
Apache ShardingSphere document

Apache ShardingSphere

Jul 18, 2022

Contents

1	Introduction	2
1.1	ShardingSphere-JDBC	2
1.2	ShardingSphere-Proxy	2
1.3	ShardingSphere-Sidecar(TODO)	4
1.4	Hybrid Architecture	5
2	Solution	6
3	Roadmap	7
4	Quick Start	8
4.1	ShardingSphere-JDBC	8
4.1.1	Scenarios	8
4.1.2	Limitations	8
4.1.3	Requirements	8
4.1.4	Procedure	8
4.2	ShardingSphere-Proxy	9
4.2.1	Scenarios	9
4.2.2	Limitations	10
4.2.3	Requirements	10
4.2.4	Procedure	10
5	Concepts	12
5.1	Adaptor	12
5.1.1	ShardingSphere-JDBC	12
5.1.2	ShardingSphere-Proxy	13
5.1.3	Hybrid Adaptors	14
5.2	Mode	16
5.2.1	Background	16
5.2.2	Standalone mode	16
5.2.3	Cluster mode	16
5.3	DistSQL	16

5.3.1	Background	16
5.3.2	Challenges	16
5.3.3	Goal	17
5.3.4	Notice	17
5.4	Pluggable Architecture	17
5.4.1	Background	17
5.4.2	Challenges	17
5.4.3	Goal	17
5.4.4	Implementation	18
	L1 Kernel Layer	18
	L2 Feature Layer	18
	L3 Ecosystem Layer	19
6	Features	20
6.1	DB Compatibility	20
6.1.1	Background	20
6.1.2	Challenges	20
6.1.3	Goal	21
6.1.4	SQL Parser	21
	MySQL	21
	openGauss	22
	PostgreSQL	22
	SQLServer	23
	Oracle	23
	SQL92	23
6.1.5	DB Protocol	23
6.1.6	Feature Support	23
	MySQL	24
	PostgreSQL	24
	SQLServer	25
	Oracle	25
	SQL92	25
6.2	DB Gateway	25
6.2.1	Background	25
6.2.2	Challenges	25
6.2.3	Goal	25
6.2.4	Current State	26
6.3	Cluster Management	26
6.3.1	Background	26
6.3.2	Challenges	26
6.3.3	Goal	26
6.3.4	Core Concept	26
	Circuit Breaker	26
	Request Limit	27
6.4	Sharding	27

6.4.1	Background	27
	Vertical Sharding	28
	Horizontal Sharding	29
6.4.2	Challenges	30
6.4.3	Goal	30
6.4.4	Core Concept	30
	Overview	30
	Table	30
	Data Node	32
	Sharding	33
	Inline Expression	34
	Distributed Primary Key	37
	Hint Sharding Route	40
6.4.5	Use Norms	40
	Background	40
	SQL	41
	Pagination	45
6.5	Distributed Transaction	47
6.5.1	Definition	47
6.5.2	Related Concepts	48
	XA Protocol	48
6.5.3	Limitations	48
	LOCAL Transaction	48
	XA Transaction	49
	BASE Transaction	49
6.5.4	How it works	50
	LOCAL Transaction	50
	XA Transaction	50
	BASE Transaction	52
6.5.5	Related references	53
6.6	ReadWrite-splitting	53
6.6.1	Definition	53
6.6.2	Related Concepts	53
	Primary database	53
	Secondary database	53
	Primary-Secondary synchronization	53
	Load balancer policy	53
6.6.3	Impact on the System	53
6.6.4	Limitations	54
6.6.5	How it works	54
6.6.6	相关参考	54
6.7	HA	54
6.7.1	Definition	54
6.7.2	Related Concepts	56
	High Availability Type	56

Dynamic Read/Write Splitting	56
6.7.3 Limitations	56
Supported	56
Not supported	56
6.7.4 How it works	56
6.7.5 Related References	57
6.8 Scaling	57
6.8.1 Background	57
6.8.2 Challenges	57
6.8.3 Goal	58
6.8.4 Status	59
6.8.5 Core Concept	59
Scaling Job	59
Inventory Data	59
Incremental Data	59
6.8.6 User Norms	59
Supported	59
Unsupported	59
6.9 Encryption	59
6.9.1 Definition	59
6.9.2 Related Concepts	60
Logic column	60
Cipher column	60
Query assistant column	60
Plain column	60
6.9.3 Impact on the system	60
6.9.4 Limitations	60
6.9.5 How it works	61
Overall architecture	61
Encryption rules	62
Encryption process	63
6.9.6 Related References	63
6.10 Shadow	63
6.10.1 Definition	63
6.10.2 Related Concepts	65
Production Database	65
Shadow Database	65
Shadow Algorithm	65
6.10.3 Limitations	66
Hint based shadow algorithm	66
Column based shadow algorithm	66
6.11 Observability	66
6.11.1 Definition	66
6.11.2 Related Concepts	67
Agent	67

APM	67
Tracing	67
Metrics	67
Logging	67
6.11.3 How it works	67
6.11.4 Related References	69
6.11.5 Use Norms	69
Compile source code	69
Agent configuration	69
Used in ShardingSphere-Proxy	72
6.12 DistSQL	72
6.12.1 Definition	72
6.12.2 Related Concepts	72
RDL	73
RQL	73
RAL	73
6.12.3 Impact on the System	73
Before	73
After	74
6.12.4 Limitations	74
6.12.5 How it works	74
6.12.6 Related References	74
7 User Manual	76
7.1 ShardingSphere-JDBC	76
7.1.1 Java API	77
Overview	77
Usage	77
Mode Configuration	78
Data Source	81
Rules	82
7.1.2 YAML Configuration	102
Overview	102
Usage	102
YAML Syntax Explanation	104
Mode Configuration	104
Data Source	106
Rules	107
7.1.3 JDBC Driver	123
Background	123
Parameters	123
Procedure	124
Sample	125
7.1.4 Spring Boot Starter	125
Overview	125

Usage	125
Mode Configuration	126
Data Source	128
Rules	130
7.1.5 Spring Namespace	145
Overview	145
Usage	145
Mode Configuration	147
Data Source	150
Rules	151
7.1.6 Properties Configuration	173
Background	173
Parameters	174
Procedure	175
Sample	175
7.1.7 Builtin Algorithm	175
Introduction	175
Usage	175
Metadata Repository	175
Sharding Algorithm	177
Key Generate Algorithm	183
Load Balance Algorithm	185
Encryption Algorithm	188
Shadow Algorithm	190
SQL Translator	192
7.1.8 Special API	193
Sharding	193
Readwrite Splitting	196
Transaction	198
Observability	209
7.1.9 Unsupported Items	214
DataSource Interface	214
Connection Interface	214
Statement and PreparedStatement Interface	214
ResultSet Interface	215
JDBC 4.1	215
7.2 ShardingSphere-Proxy	215
7.2.1 Startup	215
Use Binary Tar	216
Use Docker	218
Use Helm	219
7.2.2 Yaml Configuration	223
Authority	224
Properties	225
Rules	227

7.2.3	DistSQL	227
Syntax		227
Usage		270
7.2.4	Logging	277
To distinguish databases in the same log		277
To distinguish databases and users in the same log		278
To split into different log files		278
7.2.5	Scaling	279
Introduction		279
Build		279
Manual		283
7.3	ShardingSphere-Sidecar	295
7.3.1	Introduction	295
7.3.2	Comparison	295
8	Dev Manual	297
8.1	Mode	297
8.1.1	SPI Interface	297
8.1.2	Sample	297
StandalonePersistRepository		297
ClusterPersistRepository		298
GovernanceWatcher		298
8.2	Configuration	298
8.2.1	RuleBuilder	298
Fully-qualified class name		298
Definition		298
Implementation classes		299
8.2.2	YamlRuleConfigurationSwapper	300
Fully-qualified class name		300
Definition		300
Implementation classes		301
8.2.3	ShardingSphereYamlConstruct	302
Fully-qualified class name		302
Definition		302
Implementation classes		302
8.3	Kernel	302
8.3.1	SPI Interface	302
8.3.2	Sample	303
SQLRouter		303
SQLRewriteContextDecorator		303
SQLExecutionHook		303
ResultProcessEngine		303
StoragePrivilegeHandler		303
DynamicDataSourceStrategy		304
8.4	DataSource	304

8.4.1	SPI Interface	304
8.4.2	Sample	304
	DatabaseType	304
	DialectTableMetaDataLoader	304
	DataSourcePoolMetaData	305
	DataSourcePoolActiveDetector	305
8.5	SQL Parser	305
8.5.1	DatabaseTypedSQLParserFacade	305
	Fully-qualified class name	305
	Definition	305
	Implementation classes	307
8.5.2	SQLVisitorFacade	308
	Fully-qualified class name	308
	Definition	308
	Implementation classes	308
8.6	Proxy	308
8.6.1	DatabaseProtocolFrontendEngine	308
	Fully-qualified class name	308
	Definition	309
	Implementation classes	309
8.6.2	AuthorityProvideAlgorithm	310
	Fully-qualified class name	310
	Definition	310
	Implementation classes	310
8.7	Data Sharding	311
8.7.1	SPI Interface	311
8.7.2	Sample	311
	ShardingAlgorithm	311
	KeyGenerateAlgorithm	312
	ShardingAuditAlgorithm	312
	DatetimeService	312
	DatabaseSQLEntry	312
8.8	Readwrite-splitting	312
8.8.1	ReadQueryLoadBalanceAlgorithm	312
	Fully-qualified class name	312
	Definition	313
	Implementation classes	314
8.9	HA	315
8.9.1	DatabaseDiscoveryProviderAlgorithm	315
	Fully-qualified class name	315
	Definition	315
	Implementation classes	316
8.10	Distributed Transaction	317
8.10.1	ShardingSphereTransactionManager	317
	Fully-qualified class name	317

Definition	317
Implementation classes	318
8.10.2 XATransactionManagerProvider	319
Fully-qualified class name	319
Definition	319
Implementation classes	320
8.10.3 XADataSourceDefinition	321
Fully-qualified class name	321
Definition	321
Implementation classes	322
8.10.4 DataSourcePropertyProvider	323
Fully-qualified class name	323
Definition	323
Implementation classes	323
8.11 SQL Checker	324
8.11.1 SQLChecker	324
Fully-qualified class name	324
Definition	324
Implementation classes	325
8.12 Encryption	326
8.12.1 EncryptAlgorithm	326
Fully-qualified class name	326
Definition	326
Implementation classes	327
8.13 Shadow DB	328
8.13.1 ShadowAlgorithm	328
Fully-qualified class name	328
Definition	328
Implementation classes	329
8.14 Observability	330
8.14.1 PluginBootService	330
Fully-qualified class name	330
Definition	330
Implementation classes	331
8.14.2 PluginDefinitionService	332
Fully-qualified class name	332
Definition	332
Implementation classes	333
9 Test Manual	334
9.1 Integration Test	334
9.2 Module Test	334
9.3 Performance Test	334
9.4 Sysbench Test	335
9.5 Integration Test	335

9.5.1	Process	335
Configuration	335	
Environment Configuration	335	
Assertion Configuration	337	
Running Integration Tests	338	
9.5.2	Notice	338
9.6	Performance Test	338
9.7	Module Test	338
9.7.1	SQL Parser Test	339
Prepare Data	339	
9.7.2	SQL Rewrite Test	340
Target	340	
9.8	Scaling Integration Test	342
9.8.1	Objective	342
9.8.2	Environment	342
9.8.3	Guide	342
Configuration File	342	
Run Test Cases	343	
10	Reference	345
10.1	DistSQL	345
10.1.1	Syntax	345
RDL Syntax	345	
RQL Syntax	361	
RAL Syntax	372	
Reserved word	372	
10.2	Management	373
10.2.1	Data Structure in Registry Center	373
.rules	374	
.props	374	
.metadata/.databaseName/.versions/{versionNumber}/dataSources	374	
.metadata/.databaseName/.versions/{versionNumber}/rules	375	
.metadata/.databaseName/.schemas/{schemaName}/tables	375	
.nodes/compute_nodes	376	
.nodes/storage_nodes	376	
10.3	Sharding	376
10.3.1	SQL Parsing	377
10.3.2	SQL Route	377
10.3.3	SQL Rewrite	377
10.3.4	SQL Execution	378
10.3.5	Result Merger	378
10.3.6	Query Optimization	378
10.3.7	Parse Engine	378
Abstract Syntax Tree	378	
SQL Parser	379	

10.3.8	Route Engine	383
Sharding Route	383	
Broadcast Route	385	
10.3.9	Rewrite Engine	386
Correctness Rewrite	387	
Identifier Rewrite	387	
Column Derivation	389	
Pagination Revision	390	
Batch Split	392	
Optimization Rewrite	393	
10.3.10	Execute Engine	395
Connection Mode	395	
Automatic Execution Engine	396	
10.3.11	Merger Engine	400
Iteration Merger	400	
Order-by Merger	400	
Group-by Merger	403	
Aggregation Merger	405	
Pagination Merger	405	
10.4	Transaction	406
10.4.1	Navigation	406
10.4.2	XA Transaction	406
Transaction Begin	406	
Execute actual sharding SQL	407	
Commit or Rollback	407	
10.4.3	Seata BASE transaction	408
Init Seata Engine	409	
Transaction Begin	409	
Execute actual sharding SQL	409	
Commit or Rollback	409	
10.5	Scaling	409
10.5.1	Principle Description	409
10.5.2	Phase Description	410
Preparing Phase	410	
Inventory Phase	411	
Incremental Phase	411	
Switching Phase	411	
10.6	Encryption	412
10.6.1	Process Details	412
Overall Architecture	412	
Encryption Rule	413	
Encryption Process	414	
10.6.2	Detailed Solution	416
New Business	416	
Online Business Transformation	417	

10.6.3	The advantages of Middleware encryption service	422
10.6.4	Solution	422
EncryptAlgorithm		422
10.7	Shadow	423
10.7.1	How it works	423
DML sentence		423
DDL sentence		424
10.8	FAQ	424
10.8.1	[JDBC] Why there may be an error when configure both shardingsphere-jdbc-spring-boot-starter and a spring-boot-starter of certain datasource pool(such as druid)?	424
10.8.2	[JDBC] Why is xsd unable to be found when Spring Namespace is used?	424
10.8.3	[JDBC] Found a JtaTransactionManager in spring boot project when integrating with transaction of XA	425
10.8.4	[Proxy] In Windows environment, could not find or load main class org.apache.shardingsphere.proxyBootstrap, how to solve it?	425
10.8.5	[Proxy] How to add a new logic database dynamically when use ShardingSphere-Proxy?	425
10.8.6	[Proxy] How to use a suitable database tools connecting ShardingSphere-Proxy?	426
10.8.7	[Proxy] When using a client such as Navicat to connect to ShardingSphere-Proxy, if ShardingSphere-Proxy does not create a database or does not add a resource, the client connection will fail?	426
10.8.8	[Sharding] How to solve Cloud not resolve placeholder ...in string value ... error?	426
10.8.9	[Sharding] Why does float number appear in the return result of inline expression?	427
10.8.10	[Sharding] If sharding database is partial, should tables without sharding database & table configured in sharding rules?	427
10.8.11	[Sharding] When generic Long type SingleKeyTableShardingAlgorithm is used, why doesClassCastException: Integer can not cast to Long exception appear?	427
10.8.12	[Sharding:raw-latex:PROXY] When implementing the StandardShardingAlgorithm custom algorithm, the specific type of Comparable is specified as Long, and the field type in the database table is bigint, a ClassCastException: Integer can not cast to Long exception occurs.	427
10.8.13	[Sharding] Why are the default distributed auto-augment key strategy provided by ShardingSphere not continuous and most of them end with even numbers?	428
10.8.14	[Sharding] How to allow range query with using inline sharding strategy(BETWEEN AND, >, <, >=, <=)?	428
10.8.15	[Sharding] Why does my custom distributed primary key do not work after implementing KeyGenerateAlgorithm interface and configuring type property?	428
10.8.16	[Sharding] In addition to internal distributed primary key, does ShardingSphere support other native auto-increment keys?	429
10.8.17	[Encryption] How to solve that data encryption can't work with JPA?	429
10.8.18	[DistSQL] How to set custom JDBC connection properties or connection pool properties when adding a data source using DistSQL?	429

10.8.19 [DistSQL] How to solve Resource [xxx] is still used by [SingleTableRule]. exception when dropping a data source using DistSQL? . . .	430
10.8.20 [DistSQL] How to solve Failed to get driver instance for jdbcURL=xxx. exception when adding a data source using DistSQL?	430
10.8.21 [Other] How to debug when SQL can not be executed rightly in ShardingSphere?	430
10.8.22 [Other] Why do some compiling errors appear? Why did not the IDEA index the generated codes?	430
10.8.23 [Other] In SQLSever and PostgreSQL, why does the aggregation column without alias throw exception?	431
10.8.24 [Other] Why does Oracle database throw “Order by value must implements Comparable” exception when using Timestamp Order By?	431
10.8.25 [Other] In Windows environment,when cloning ShardingSphere source code through Git, why prompt filename too long and how to solve it?	432
10.8.26 [Other] How to solve Type is required error?	433
10.8.27 [Other] How to speed up the metadata loading when service starts up?	433
10.8.28 [Other] The ANTLR plugin generates codes in the same level directory as src, which is easy to commit by mistake. How to avoid it?	433
10.8.29 [Other] Why is the database sharding result not correct when using Proxool? . .	435
10.8.30 [Other] The property settings in the configuration file do not take effect when integrating ShardingSphere with Spring Boot 2.x ?	435
10.8.31 [ShardingSphere-JDBC] The tableName and columnName configured in yaml or properties leading incorrect result when loading Oracle metadata?	437

11 Downloads	438
11.1 Latest Releases	438
11.1.1 Apache ShardingSphere - Version: 5.1.2 (Release Date: June 17th, 2022)	438
11.2 All Releases	438
11.3 Verify the Releases	438

Stargazers Over Time

Contributors Over Time

Apache ShardingSphere is positioned as a Database Plus, and aims at building a standard layer and ecosystem above heterogeneous databases. It focuses on how to reuse existing databases and their respective upper layer, rather than creating a new database. The goal is to minimize or eliminate the challenges caused by underlying databases fragmentation.

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

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

ShardingSphere became an [Apache Top-Level Project](#) on April 16, 2020.

Welcome to interact with community via the official [mail list](#) and the ShardingSphere Slack.

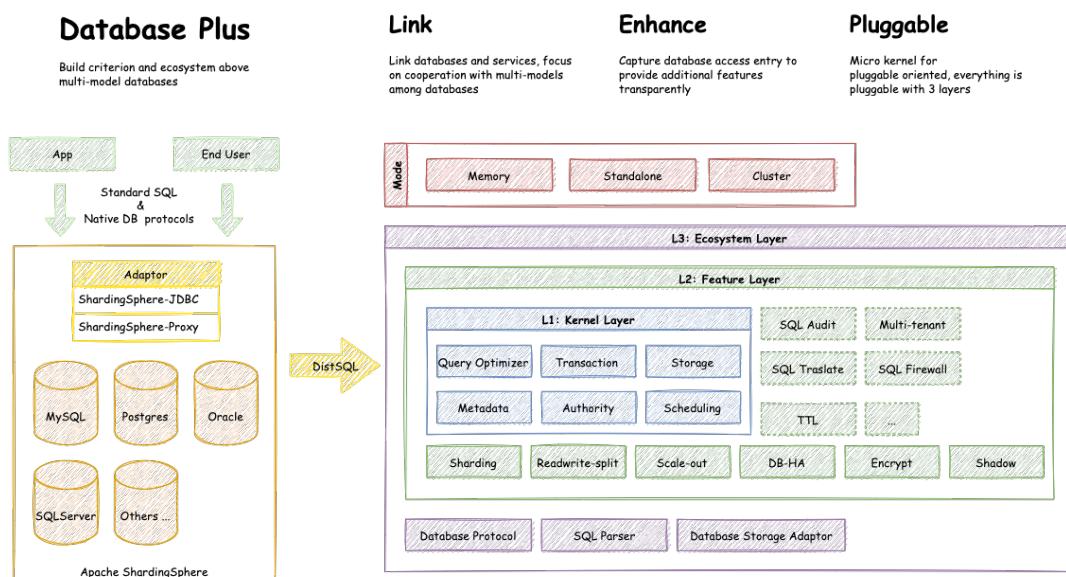


Figure1: Overview

Introduction

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

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

1.1 ShardingSphere-JDBC

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

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

1.2 ShardingSphere-Proxy

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

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



Figure1: ShardingSphere-JDBC Architecture



Figure2: ShardingSphere-Proxy Architecture

1.3 ShardingSphere-Sidecar(TODO)

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

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

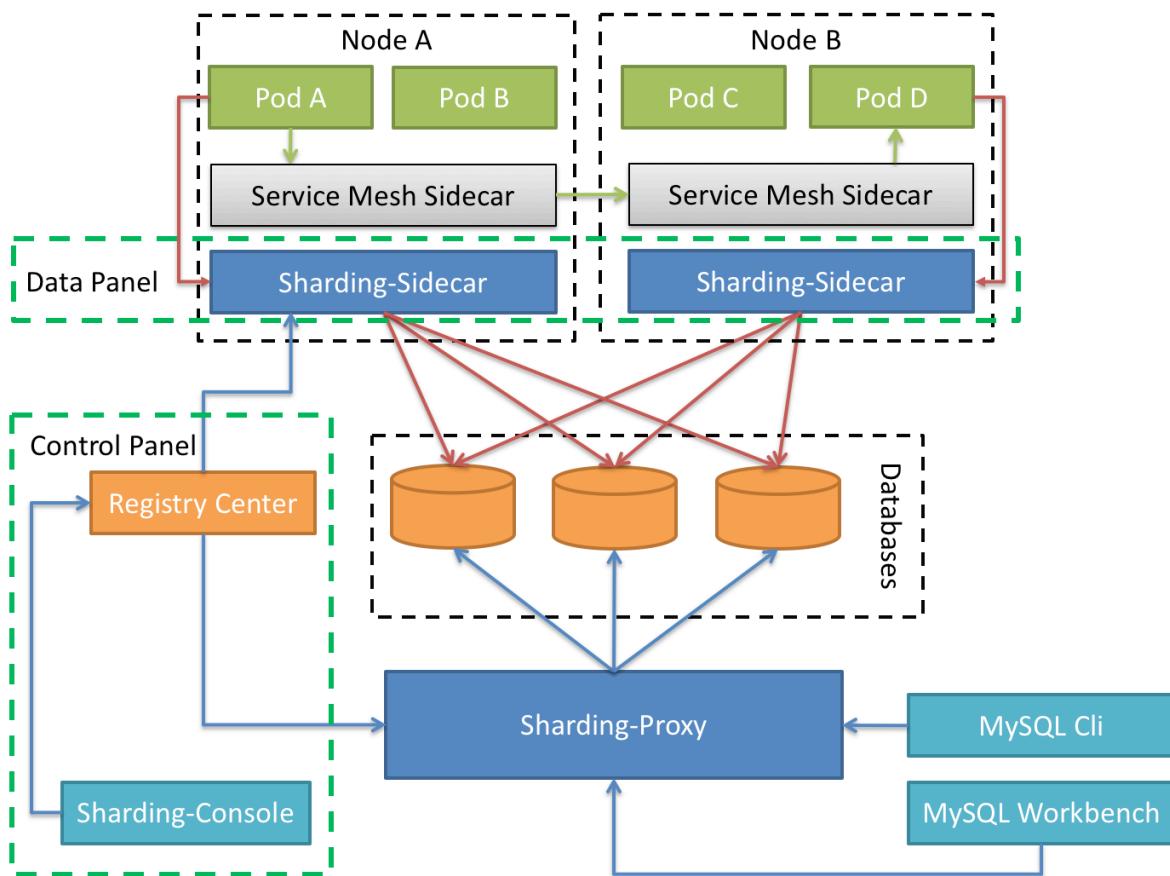


Figure3: ShardingSphere-Sidecar Architecture

	<i>ShardingSphere-JDBC</i>	<i>ShardingSphere-Proxy</i>	<i>ShardingSphere-Sidecar</i>
Database	Any	MySQL/PostgreSQL	MySQL/PostgreSQL
Connections Count Cost	High	Low	High
Supported Languages	Java Only	Any	Any
Performance	Low loss	Relatively High loss	Low loss
De centralization	Yes	No	No
Static Entry	No	Yes	No

1.4 Hybrid Architecture

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

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

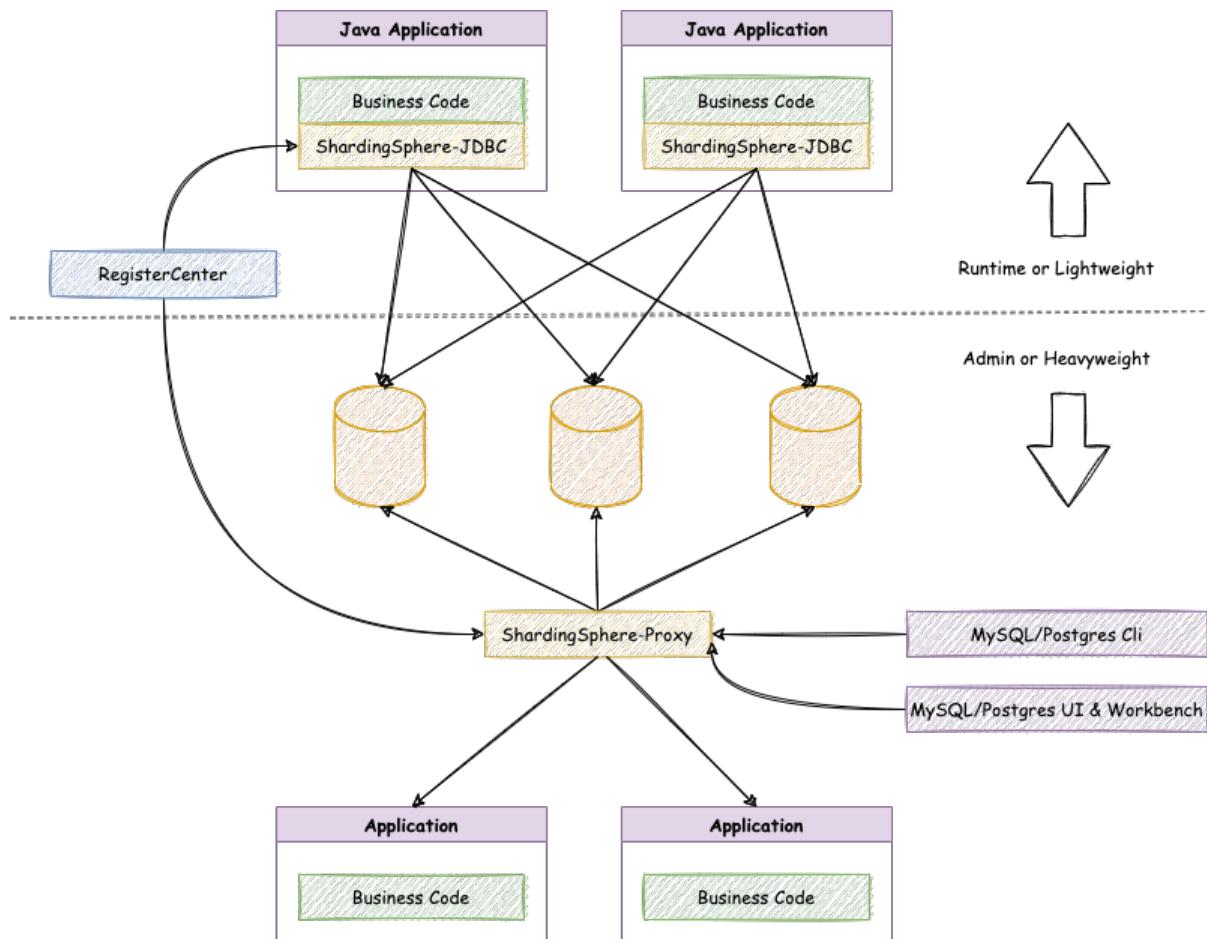


Figure4: ShardingSphere Hybrid Architecture

2

Solution

Solutions/ Features	Distributed Database*	Data Security	Database Gateway	Stress Testing
	Data Sharding	Data Encrypt	Heterogeneous Databases Supported	Shadow Database
	Readwrite splitting	Row Authority (TODO)	SQL Dialect Translate (TODO)	Observability
	Distributed Transaction	SQL Audit (TODO)		
	Elastic Scale-out	SQL Firewall (TODO)		
	Highly Available			

Roadmap

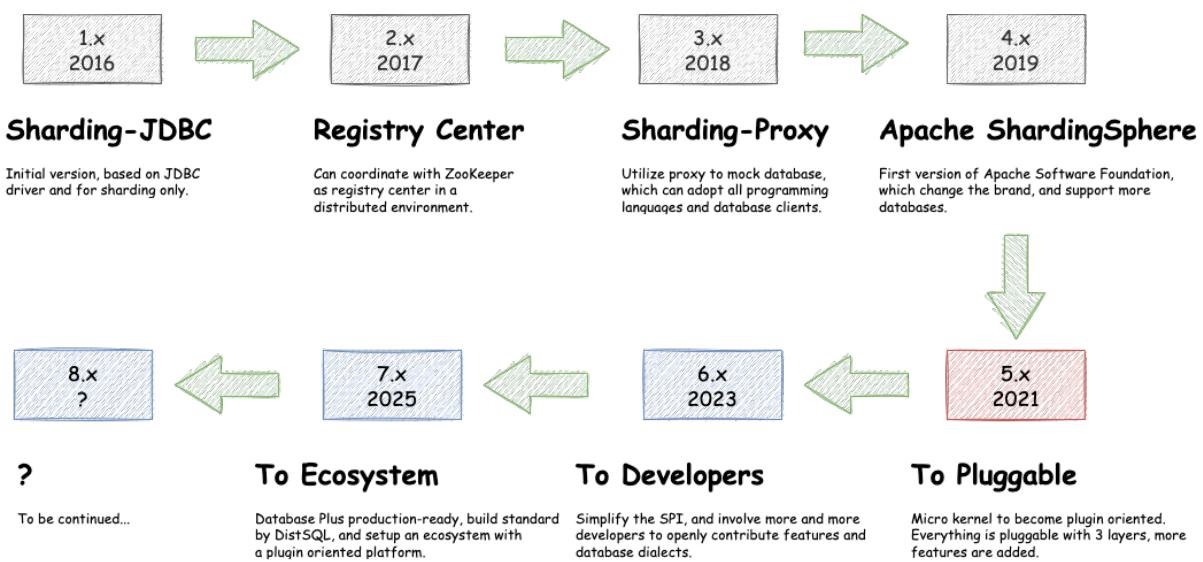


Figure1: Roadmap

4

Quick Start

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

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

4.1 ShardingSphere-JDBC

4.1.1 Scenarios

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

4.1.2 Limitations

Currently only Java language is supported.

4.1.3 Requirements

The development environment requires Java JRE 8 or later.

4.1.4 Procedure

1. Rules configuration.

Please refer to [User Manual](#) for more details.

2. Import Maven dependency

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

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

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

3. Edit application.yml.

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

4.2 ShardingSphere-Proxy

4.2.1 Scenarios

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



Figure1: shardingsphere-proxy

4.2.2 Limitations

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

4.2.3 Requirements

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

4.2.4 Procedure

1. Get ShardingSphere-Proxy.

ShardingSphere-Proxy is available at: - [Binary Distribution](#) - [Docker](#) - [Helm](#)

2. Rule configuration.

Edit %SHARDINGSPHERE_PROXY_HOME%/conf/server.yaml.

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

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

Please refer to [Configuration Manual](#) for more details.

3. Import dependencies.

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

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

4. Start server.

- Use the default configuration to start

```
sh %SHARDINGSPHERE_HOME%/bin/start.sh
```

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

- Customize port and profile directory

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

5. Use ShardingSphere-Proxy.

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

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

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

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

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

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

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

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

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

This chapter describes concepts about Apache ShardingSphere.

5.1 Adaptor

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

5.1.1 ShardingSphere-JDBC

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

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

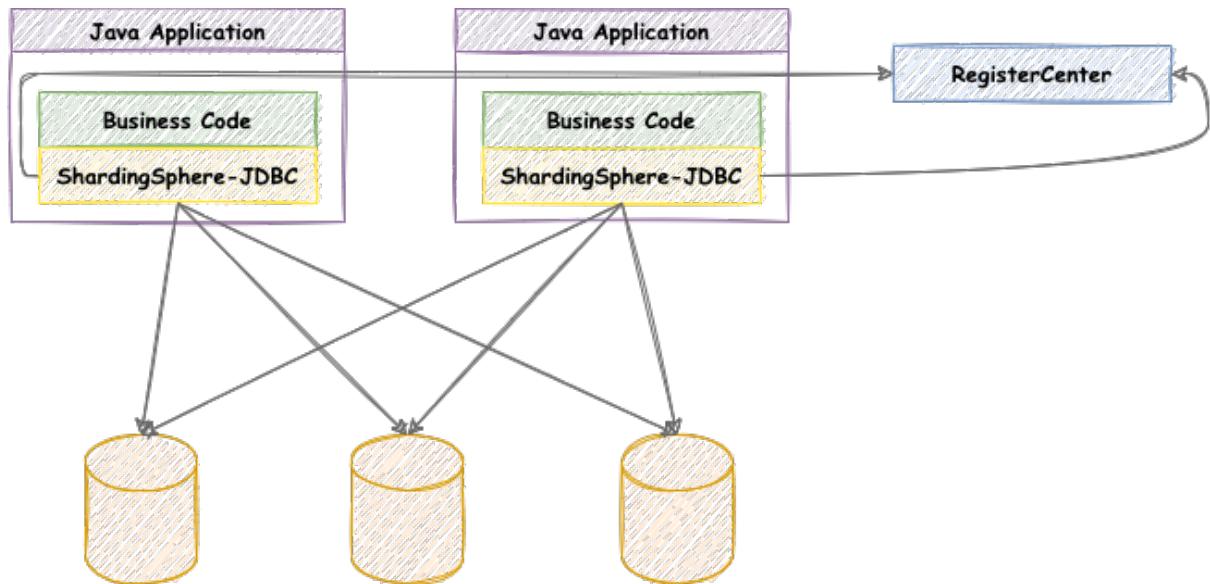


Figure1: ShardingSphere-JDBC Architecture

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

ShardingSphere-JDBC is suitable for java application.

Source Codes: <https://github.com/apache/shardingsphere/tree/master/shardingsphere-jdbc>

5.1.2 ShardingSphere-Proxy

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

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

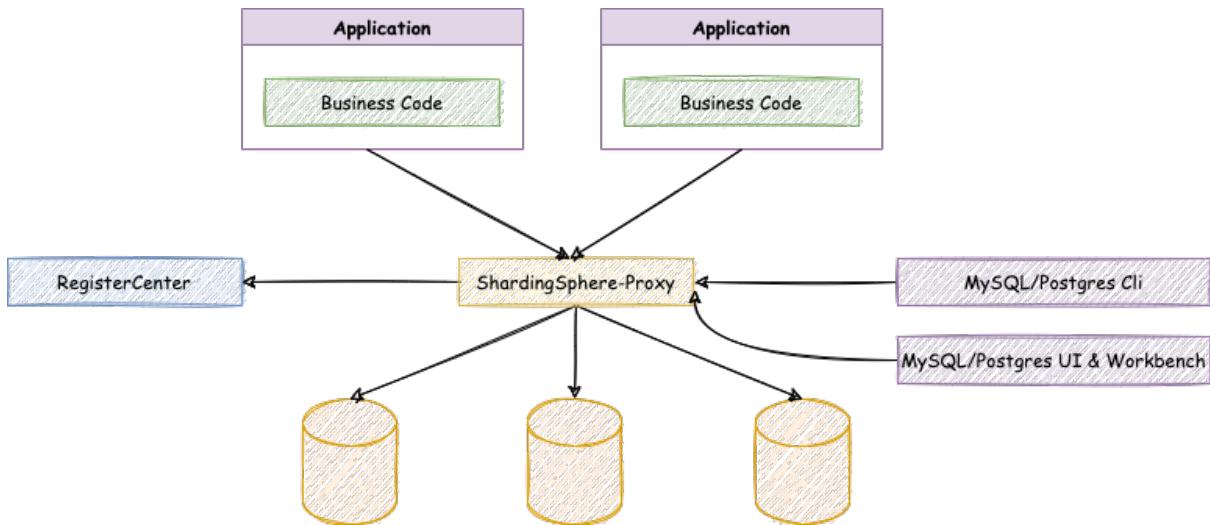


Figure2: ShardingSphere-Proxy Architecture

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

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

Source Codes: <https://github.com/apache/shardingsphere/tree/master/shardingsphere-proxy>

5.1.3 Hybrid Adaptors

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

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

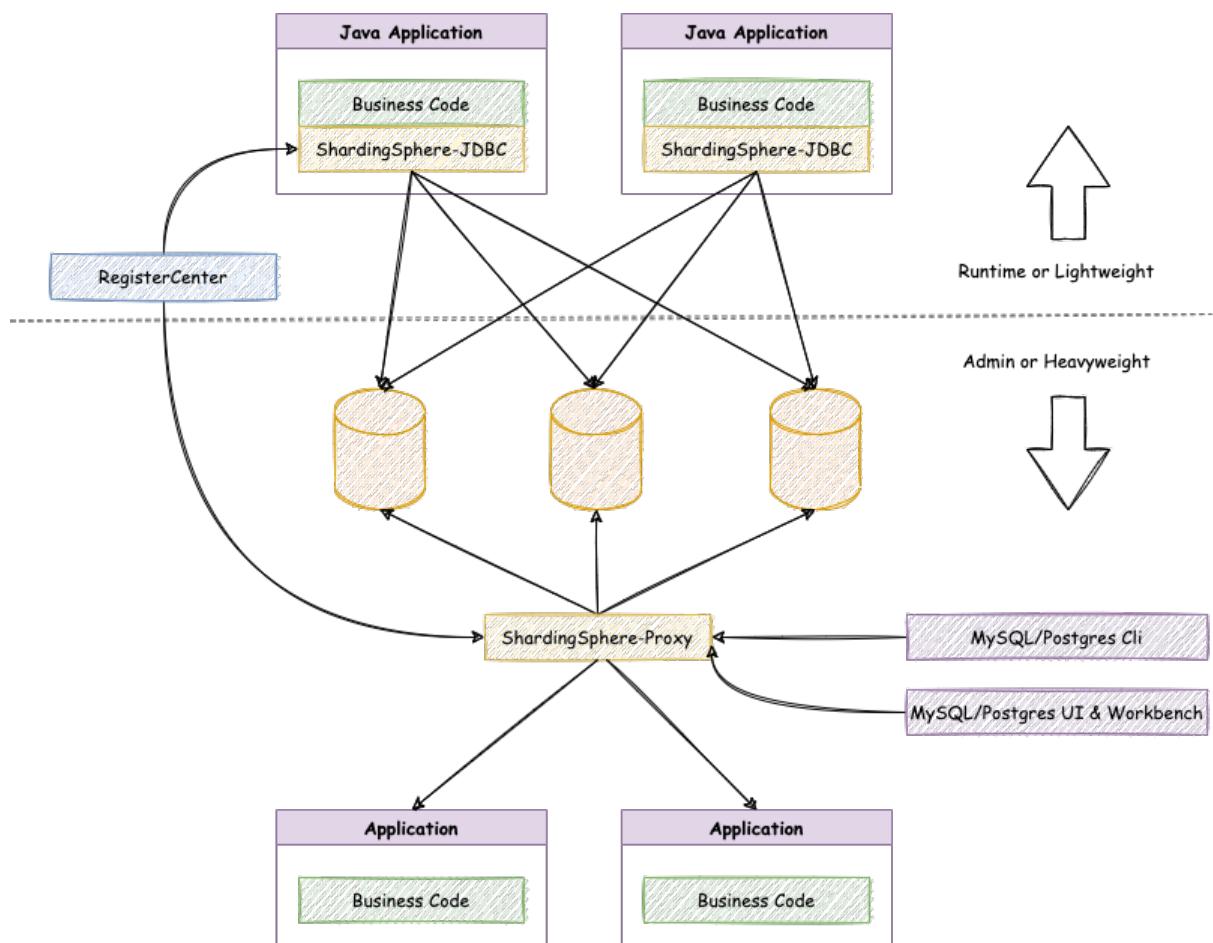


Figure3: Hybrid Architecture

5.2 Mode

5.2.1 Background

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

5.2.2 Standalone mode

Suitable in a standalone environment, through which data sources, rules, and metadata can be persisted. Will use H2 database to store configuration data by default.

5.2.3 Cluster mode

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

Source Codes: <https://github.com/apache/shardingsphere/tree/master/shardingsphere-mode>

5.3 DistSQL

5.3.1 Background

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

5.3.2 Challenges

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

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

DistSQL is divided into RDL, RQL and RAL.

- RDL (Resource & Rule Definition Language) responsible for the definition of resources and rules;
- RQL (Resource & Rule Query Language) responsible for the query of resources and rules;
- RAL (Resource & Rule Administration Language) responsible for the added-on administrator feature of hint, transaction type switch, sharding execute planning and so on.

5.3.3 Goal

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

5.3.4 Notice

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

Source Codes: <https://github.com/apache/shardingsphere/tree/master/shardingsphere-distsql>

5.4 Pluggable Architecture

5.4.1 Background

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

5.4.2 Challenges

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

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

5.4.3 Goal

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

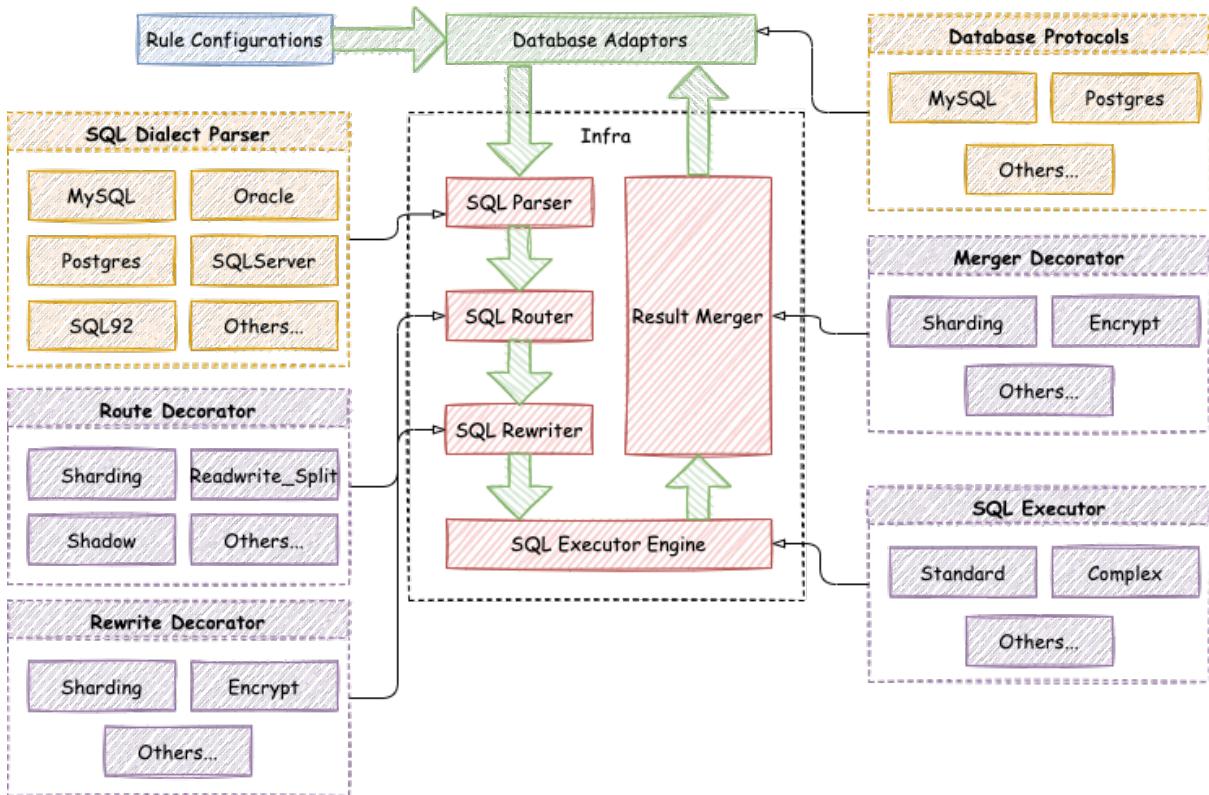


Figure4: Pluggable Platform

5.4.4 Implementation

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

L1 Kernel Layer

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

L2 Feature Layer

Used to provide enhanced capability. All components are optional and can contain zero or multiple components. Components isolate each other and multiple components can be used together superimposed. It includes data sharding, readwrite-splitting, database highly availability, data encryption, shadow database and so on. The user-defined feature can be fully customized and extended for the top-level interface defined by Apache ShardingSphere without changing kernel codes.

L3 Ecosystem Layer

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

6

Features

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

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

6.1 DB Compatibility

6.1.1 Background

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

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

The trend of database fragmentation is coming.

6.1.2 Challenges

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

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

6.1.3 Goal

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

6.1.4 SQL Parser

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

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

This chapter has listed unsupported SQLs reference for users.

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

Source Codes: <https://github.com/apache/shardingsphere/tree/master/shardingsphere-sql-parser>

MySQL

The unsupported SQL list for MySQL are as follows:

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

Source Codes: <https://github.com/apache/shardingsphere/tree/master/shardingsphere-sql-parser/shardingsphere-sql-parser-dialect/shardingsphere-sql-parser-mysql>

openGauss

The unsupported SQL list for openGauss are as follows:

SQL
CREATE type avg_state AS (total bigint, count bigint);
CREATE AGGREGATE my_avg(int4) (stype = avg_state, sfunc = avg_transfn, finalfunc = avg_finalfn)
CREATE TABLE agg_data_2k AS SELECT g FROM generate_series(0, 1999) g;
CREATE SCHEMA alt_nsp1;
ALTER AGGREGATE alt_agg3(int) OWNER TO regress_alter_generic_user2;
CREATE CONVERSION alt_conv1 FOR ‘LATIN1’ TO ‘UTF8’ FROM iso8859_1_to_utf8;
CREATE FOREIGN DATA WRAPPER alt_fdw1
CREATE SERVER alt_fserv1 FOREIGN DATA WRAPPER alt_fdw1
CREATE LANGUAGE alt_lang1 HANDLER plpgsql_call_handler
CREATE STATISTICS alt_stat1 ON a, b FROM alt_regress_1
CREATE TEXT SEARCH DICTIONARY alt_ts_dict1 (template=simple)
CREATE RULE def_view_test_ins AS ON INSERT TO def_view_test DO INSTEAD INSERT INTO def_test SELECT new.*
ALTER TABLE alterlock SET (toast.autovacuum_enabled = off)
CREATE PUBLICATION pub1 FOR TABLE alter1.t1, ALL TABLES IN SCHEMA alter2

Source Codes: <https://github.com/apache/shardingsphere/tree/master/shardingsphere-sql-parser/shardingsphere-sql-parser-dialect/shardingsphere-sql-parser-opengauss>

PostgreSQL

The unsupported SQL list for PostgreSQL are as follows:

SQL
CREATE type avg_state AS (total bigint, count bigint);
CREATE AGGREGATE my_avg(int4) (stype = avg_state, sfunc = avg_transfn, finalfunc = avg_finalfn)
CREATE TABLE agg_data_2k AS SELECT g FROM generate_series(0, 1999) g;
CREATE SCHEMA alt_nsp1;
ALTER AGGREGATE alt_agg3(int) OWNER TO regress_alter_generic_user2;
CREATE CONVERSION alt_conv1 FOR ‘LATIN1’ TO ‘UTF8’ FROM iso8859_1_to_utf8;
CREATE FOREIGN DATA WRAPPER alt_fdw1
CREATE SERVER alt_fserv1 FOREIGN DATA WRAPPER alt_fdw1
CREATE LANGUAGE alt_lang1 HANDLER plpgsql_call_handler
CREATE STATISTICS alt_stat1 ON a, b FROM alt_regress_1
CREATE TEXT SEARCH DICTIONARY alt_ts_dict1 (template=simple)
CREATE RULE def_view_test_ins AS ON INSERT TO def_view_test DO INSTEAD INSERT INTO def_test SELECT new.*
ALTER TABLE alterlock SET (toast.autovacuum_enabled = off)
CREATE PUBLICATION pub1 FOR TABLE alter1.t1, ALL TABLES IN SCHEMA alter2

Source Codes: <https://github.com/apache/shardingsphere/tree/master/shardingsphere-sql-parser/shardingsphere-sql-parser-dialect/shardingsphere-sql-parser-postgresql>

SQLServer

The unsupported SQL list for SQLServer are as follows:

TODO

Source Codes: <https://github.com/apache/shardingsphere/tree/master/shardingsphere-sql-parser/shardingsphere-sql-parser-dialect/shardingsphere-sql-parser-sqlserver>

Oracle

The unsupported SQL list for Oracle are as follows:

TODO

Source Codes: <https://github.com/apache/shardingsphere/tree/master/shardingsphere-sql-parser/shardingsphere-sql-parser-dialect/shardingsphere-sql-parser-oracle>

SQL92

The unsupported SQL list for SQL92 are as follows:

TODO

Source Codes: <https://github.com/apache/shardingsphere/tree/master/shardingsphere-sql-parser/shardingsphere-sql-parser-dialect/shardingsphere-sql-parser-sql92>

6.1.5 DB Protocol

Apache ShardingSphere implements MySQL and PostgreSQL Protocol.

Source Codes: <https://github.com/apache/shardingsphere/tree/master/shardingsphere-db-protocol>

6.1.6 Feature Support

Apache ShardingSphere provides the ability of distributed collaboration for the database, and abstracts part of the database features to the upper layer for unified management to reduce the difficulty of users.

Therefore, for the unified provided features, the native SQL will no longer be transferred to the database, and it will be prompted that the operation is not supported. User can use the feature provided by ShardingSphere to replace it.

This chapter has listed unsupported database features and related SQLs reference for users.

There are some unsupported SQLs maybe missing, welcome to finish them.

MySQL

The unsupported SQL list for MySQL are as follows:

User & Role

SQL
CREATE USER 'finley' @ 'localhost' IDENTIFIED BY 'password'
ALTER USER 'finley' @ 'localhost' IDENTIFIED BY 'new_password'
DROP USER 'finley' @ 'localhost' ;
CREATE ROLE 'app_read'
DROP ROLE 'app_read'
SHOW CREATE USER finley
SET PASSWORD = 'auth_string'
SET ROLE DEFAULT;

Authorization

SQL
GRANT ALL ON db1.* TO 'jeffrey' @ 'localhost'
GRANT SELECT ON world.* TO 'role3' ;
GRANT 'role1' , 'role2' TO 'user1' @ 'localhost'
REVOKE INSERT ON . FROM 'jeffrey' @ 'localhost'
REVOKE 'role1' , 'role2' FROM 'user1' @ 'localhost'
REVOKE ALL PRIVILEGES, GRANT OPTION FROM user_or_role
SHOW GRANTS FOR 'jeffrey' @ 'localhost'
SHOW GRANTS FOR CURRENT_USER
FLUSH PRIVILEGES

PostgreSQL

The unsupported SQL list for PostgreSQL are as follows:

TODO

SQLServer

The unsupported SQL list for SQLServer are as follows:

TODO

Oracle

The unsupported SQL list for Oracle are as follows:

TODO

SQL92

The unsupported SQL list for SQL92 are as follows:

TODO

6.2 DB Gateway

6.2.1 Background

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

6.2.2 Challenges

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

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

6.2.3 Goal

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

6.2.4 Current State

SQL translation in Apache ShardingSphere is in the **experimental stage** currently.

It has supported auto translation with MySQL/PostgreSQL, engineers can use the SQL and protocol of MySQL to access PostgreSQL, vice versa.

Source Codes: <https://github.com/apache/shardingsphere/tree/master/shardingsphere-kernel/shardingsphere-sql-translator>

6.3 Cluster Management

6.3.1 Background

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

6.3.2 Challenges

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

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

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

6.3.3 Goal

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

6.3.4 Core Concept

Circuit Breaker

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

Request Limit

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

6.4 Sharding

6.4.1 Background

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

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

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

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

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

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

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

Vertical Sharding

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

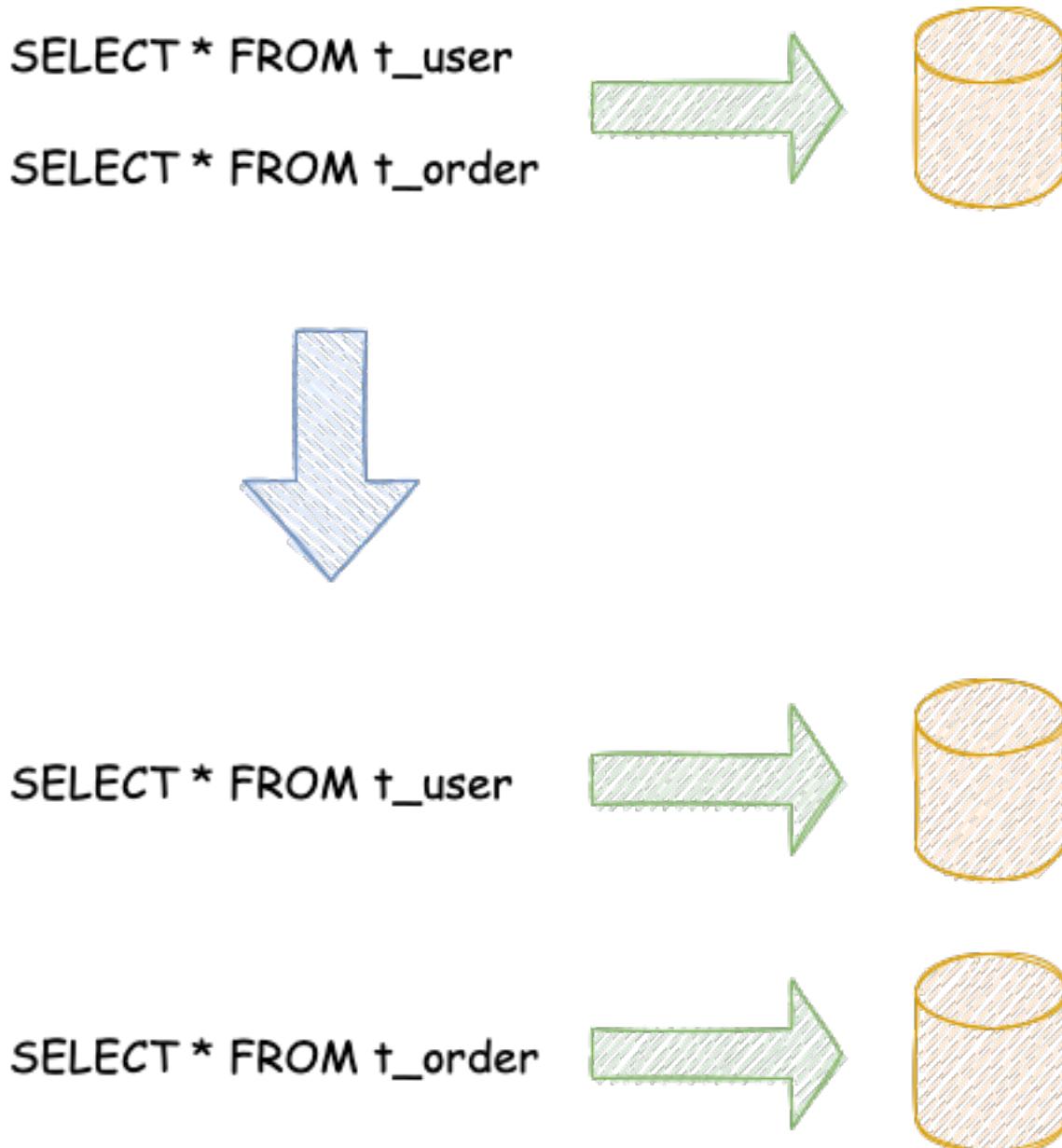


Figure1: Vertical Sharding

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

exceeds the single node threshold, it should be further processed by horizontal sharding.

Horizontal Sharding

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



Figure2: Horizontal Sharding

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

6.4.2 Challenges

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

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

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

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

6.4.3 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.

Source Codes: <https://github.com/apache/shardingsphere/tree/master/shardingsphere-features/shardingsphere-sharding>

6.4.4 Core Concept

Overview

This chapter is to introduce core concepts of data sharding.

Table

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

Logic Table

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

Actual Table

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

Binding Table

It refers to a group of sharding tables with the same sharding rules. When using binding tables in multi-table correlating query, you must use the sharding key for correlation, otherwise Cartesian product correlation or cross-database correlation will appear, which will affect query efficiency. For example, `t_order` and `t_order_item` are both sharded by `order_id`, and use `order_id` to correlate, so they are binding tables with each other. Cartesian product correlation will not appear in the multi-tables correlating query, so the query efficiency will increase greatly. Take this one for example, if SQL is:

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

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

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

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

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

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

With binding table configuration and use `order_id` to correlate, there should be 2 SQLs after routing:

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

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

In them, since table `t_order` specifies sharding conditions, it will be taken by ShardingSphere as the primary table of query. 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`.

Broadcast Table

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

Single Table

It refers to only one table that exists in all sharding database sources. It is suitable for little data in table without sharding. Users need to ensure that single table is unique. If single table is repeated under the same schema in the same database, ShardingSphere will only load the first single table for sql route.

Data Node

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

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

Uniform topology

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

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

The data node configurations will be as follows:

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

User-defined topology

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

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

The data node configurations will be as follows:

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

Sharding

Sharding Key

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

Sharding Algorithm

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

Auto Sharding Algorithm

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

User-Defined Sharding Algorithm

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

- Standard Sharding Algorithm

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

- Complex Keys Sharding Algorithm

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

- Hint Sharding Algorithm

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

Sharding Strategy

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

SQL Hint

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

Inline Expression

Motivation

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

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

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

Syntax Explanation

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

`${begin..end}` means range

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

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

For example, the following inline expression:

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

Will be parsed as:

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

Configuration

Data Node

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

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

It can be simplified by inline expression as:

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

Or

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

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

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

```
└── t_order2  
└── t_order3  
└── t_order4
```

It can be simplified by inline expression as:

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

Or

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

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

```
db0  
└── t_order_00  
└── t_order_01  
└── t_order_02  
└── t_order_03  
└── t_order_04  
└── t_order_05  
└── t_order_06  
└── t_order_07  
└── t_order_08  
└── t_order_09  
└── t_order_10  
└── t_order_11  
└── t_order_12  
└── t_order_13  
└── t_order_14  
└── t_order_15  
└── t_order_16  
└── t_order_17  
└── t_order_18  
└── t_order_19  
└── t_order_20  
  
db1  
└── t_order_00  
└── t_order_01  
└── t_order_02  
└── t_order_03  
└── t_order_04  
└── t_order_05  
└── t_order_06  
└── t_order_07  
└── t_order_08  
└── t_order_09  
└── t_order_10
```

```

└── t_order_11
└── t_order_12
└── t_order_13
└── t_order_14
└── t_order_15
└── t_order_16
└── t_order_17
└── t_order_18
└── t_order_19
└── t_order_20

```

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

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

Or

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

Sharding Algorithm

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

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

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

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

Or

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

Distributed Primary Key

Motivation

In the development of traditional database software, the automatic sequence generation technology is a basic requirement. All kinds of databases have provided corresponding support for this requirement, such as MySQL auto-increment key, Oracle auto-increment sequence and so on. It is a tricky problem that there is only one sequence generated by different data nodes after sharding. Auto-increment keys in different physical tables in the same logic table can not perceive each other and thereby generate

repeated sequences. It is possible to avoid clashes by restricting the initiative value and increasing the step of auto-increment key. But introducing extra operation rules can make the solution lack integrity and scalability.

Currently, there are many third-party solutions that can solve this problem perfectly, (such as UUID and others) relying on some particular algorithms to generate unrepeatable 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.

Nanoid

Generate a string of length 21 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^{41} to the power of milliseconds. One year can use $365 * 24 * 60 * 60 * 1000$ milliseconds. We can see from the calculation:

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

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

- work ID bit (10bit)

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

- sequence number bit (12bit)

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

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

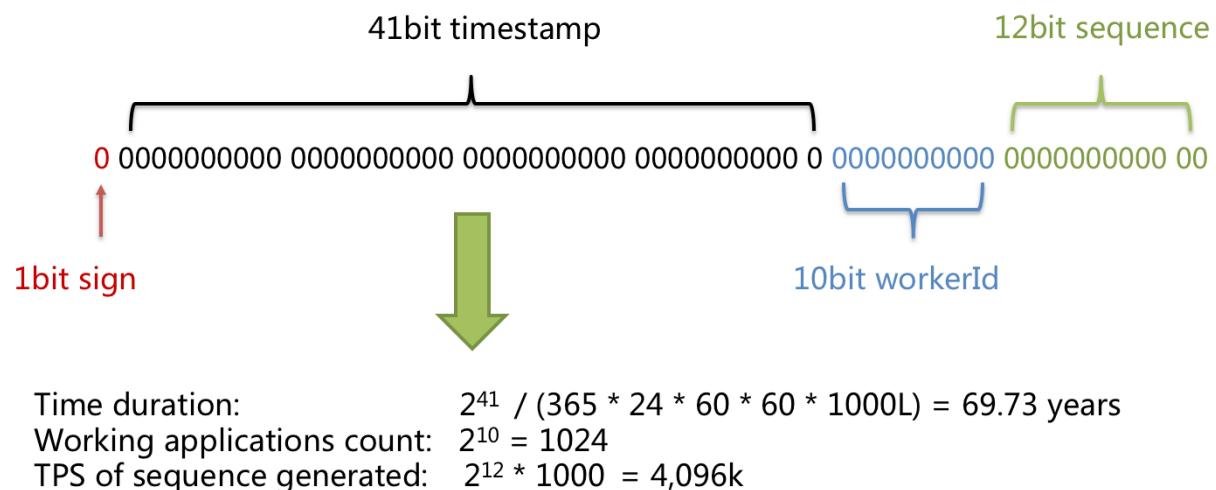


Figure3: Snowflake

Clock-Back

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

Hint Sharding Route

Motivation

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

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

Mechanism

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

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

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

6.4.5 Use Norms

Background

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

SQL

SQL Supporting Status

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

Totally Supported

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

Regular Query

- SELECT Clause

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

- select_expr

```
*
```

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

- table_reference

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

Subquery

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

For example:

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

Stable supported for subquery with [pagination](#).

For example:

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

Sharding value in expression

Sharding value in calculated expressions will lead to full routing.

For example, if `create_time` is sharding value:

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

Experimental Supported

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

Subquery

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

For example:

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

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

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

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

Join with cross databases

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

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

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

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

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

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

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

Unsupported

CASE WHEN can not support as following:

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

SQL Example

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

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

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

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

Pagination

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

Pagination Performance

Performance Bottleneck

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

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

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

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

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

Optimization of ShardingSphere

ShardingSphere has optimized in two ways.

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

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

6.5 Distributed Transaction

6.5.1 Definition

Four properties of transactions: ACID (Atomicity、Consistency、Isolation、Durability).

- Atomicity: transactions are executed as a whole, and either all or none is executed.
- Consistency: transactions should ensure that the state of data remains consistent after the transaction.

- Isolation: when multiple transactions execute concurrently, the execution of one transaction should not affect the execution of others.
- Durability: when a transaction committed modifies data, the operation will be saved persistently.

Distributed transactions guarantee the ACID properties in distributed scenarios, where a single transaction involves operations on multiple data nodes.

6.5.2 Related Concepts

XA Protocol

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

6.5.3 Limitations

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

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

LOCAL Transaction

Supported

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

Unsupported

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

XA Transaction

Supported

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

Unsupported

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

BASE Transaction

Supported

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

Unsupported

- Do not support other isolation level except RC.

To Be Optimized

- SQL parsed twice by Apache ShardingSphere and SEATA.

6.5.4 How it works

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

LOCAL Transaction

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

XA Transaction

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

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

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

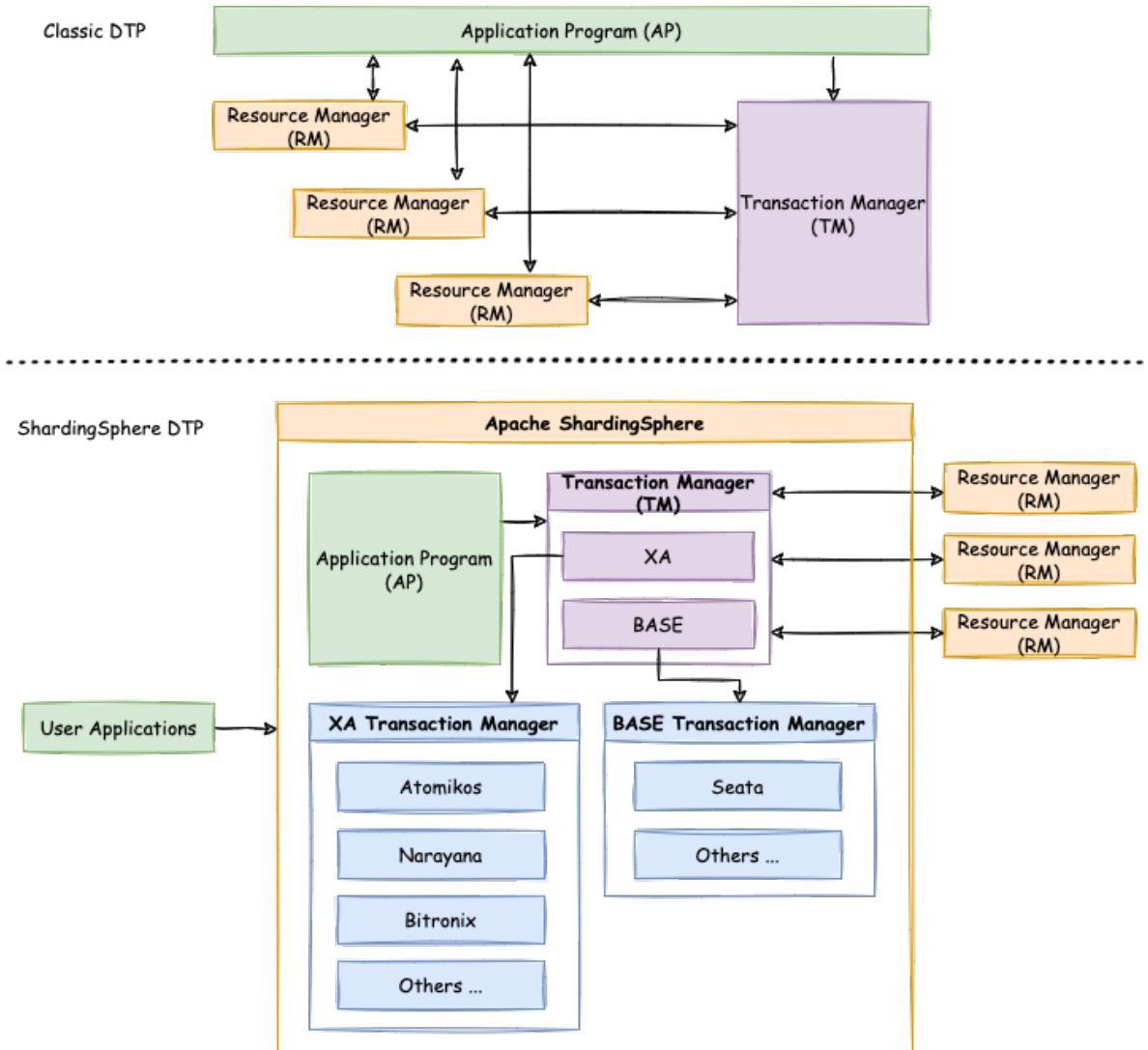


Figure4: Two-phase commit model

BASE Transaction

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

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

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

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

	<i>LOCAL</i>	<i>XA</i>	<i>BASE</i>
Business transformation	None	None	Seata server needed
Consistency	Not supported	Not supported	Final consistency
Isolation	Not supported	Supported	Business side guaranteed
Concurrent performance	no loss	severe loss	slight loss
Applied scenarios	Inconsistent processing by the business side	short transaction & low-level concurrency	long transaction & high concurrency

6.5.5 Related references

- YAML distributed transaction configuration

6.6 Readwrite-splitting

6.6.1 Definition

Read/write splitting is to split the database into primary and secondary databases. The primary database is responsible for handling transactional operations including additions, deletions and changes. And the secondary database is responsible for the query operation of database architecture.

6.6.2 Related Concepts

Primary database

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

Secondary database

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

Primary-Secondary synchronization

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

Load balancer policy

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

6.6.3 Impact on the System

There may be complex primary-secondary relational database clusters in users' systems, so applications need to access multiple data sources, which increases the cost of system maintenance and the difficulty of business development. ShardingSphere enables users to use database clusters like a database through read/write splitting function, and the impact of read/write splitting will be transparent to users.

6.6.4 Limitations

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

6.6.5 How it works

ShardingSphere's read/write splitting mainly relies on the related functions of its kernel, including a parsing engine and a routing engine. The parsing engine converts the user's SQL into Statement information that can be identified by ShardingSphere, and the routing engine performs SQL routing according to the read/write type of SQL and transactional status. The routing from the secondary database supports a variety of load balancing algorithms, including polling algorithm, random access algorithm, weight access algorithm, etc. Users can also expand the required algorithm according to the SPI mechanism. As shown in the figure below, ShardingSphere identifies read and write operations and routes them to different database instances respectively.

6.6.6 相关参考

[Java API](#)

[YAML Configuration](#)

[Spring Boot Starter](#)

[Spring Namespace](#)

6.7 HA

6.7.1 Definition

High availability is the most basic requirement for modern systems. It is also an essential element of the database, which in turn is the cornerstone of any system. In a distributed database system, storage nodes and compute nodes are different in terms of their high availability schemes. Stateful storage nodes are required to have capabilities such as data consistency and synchronization, liveness probe, and primary-node election. Stateless compute nodes need to sense storage nodes' changes, setup load balancers independently, and enable service discovery and request distribution. Apache ShardingSphere's high availability module (HA) is mainly designed to ensure a 24/7 database service as much as possible.

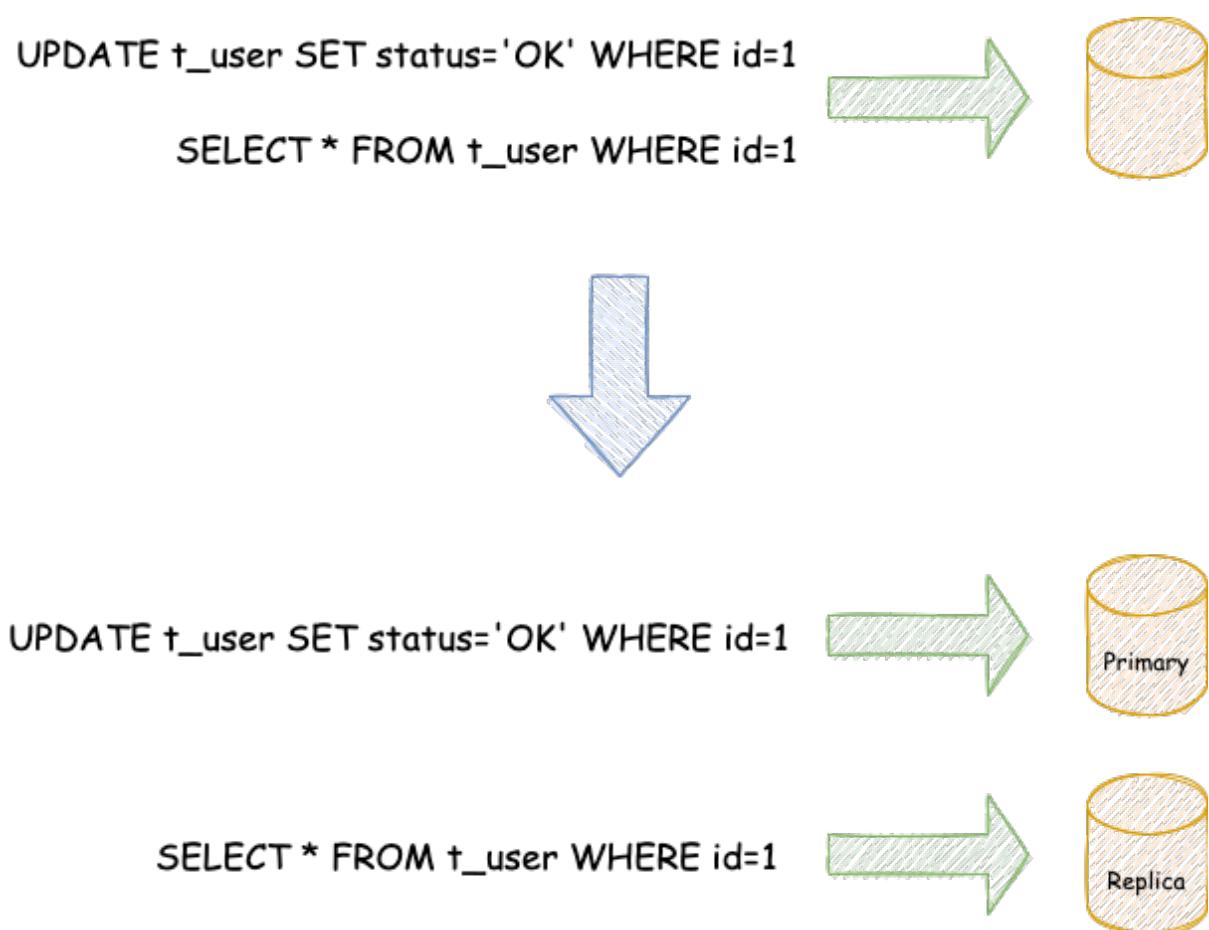


Figure5: 原理介绍

6.7.2 Related Concepts

High Availability Type

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

Dynamic Read/Write Splitting

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

6.7.3 Limitations

Supported

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

Not supported

- MySQL MGR Multi-primary mode

6.7.4 How it works

The high availability solution provided by Apache ShardingSphere allows you to carry out secondary custom development and achieve expansion, which is mainly divided into four steps: pre-check, primary database dynamic discovery, secondary database dynamic discovery and configuration synchronization.



Figure6: Overview

6.7.5 Related References

[Java API](#)

[YAML Configuration](#)

[Spring Boot Starter](#)

[Spring Namespace](#)

[Source Code](#)

6.8 Scaling

6.8.1 Background

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

6.8.2 Challenges

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

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

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

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

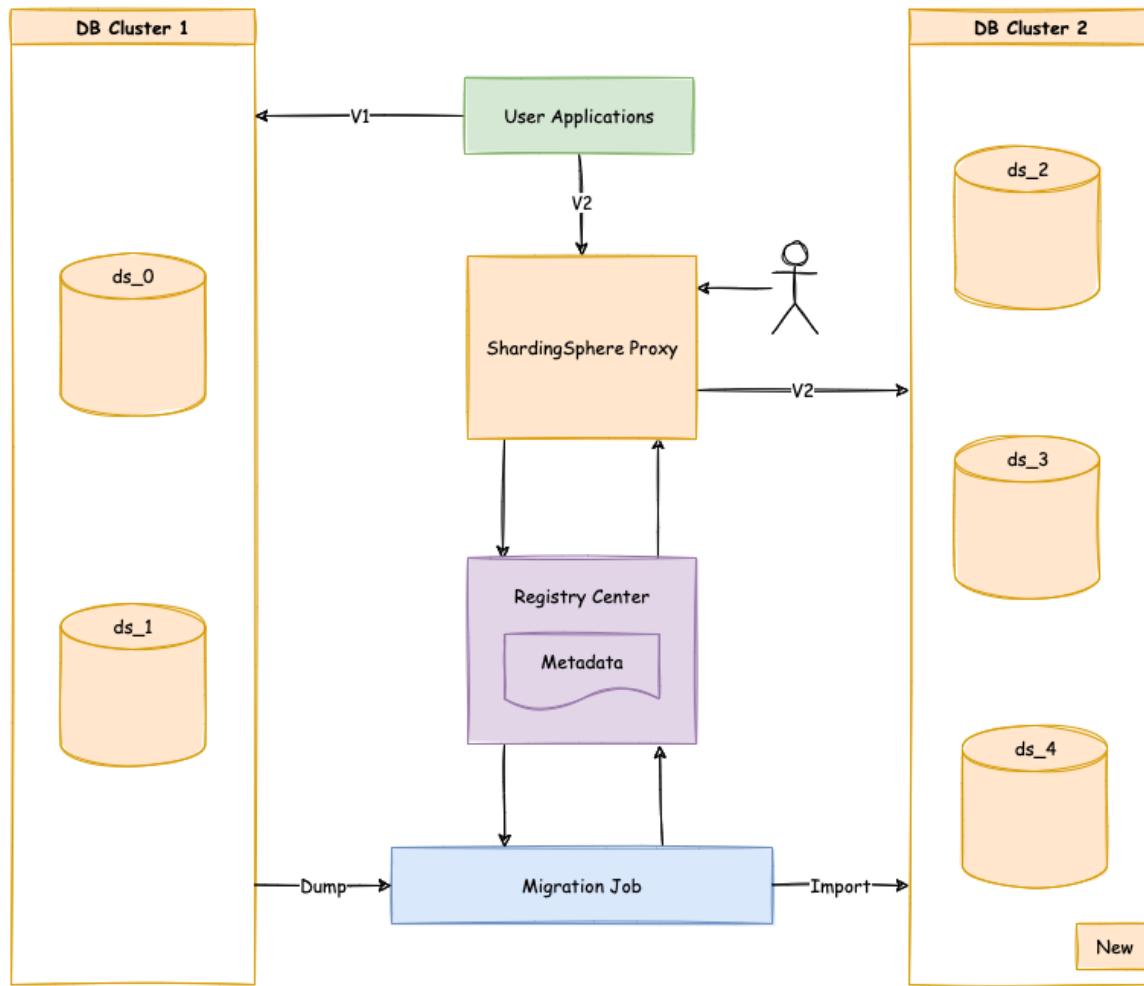


Figure7: Overview

6.8.3 Goal

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

6.8.4 Status

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

Source Codes: <https://github.com/apache/shardingsphere/tree/master/shardingsphere-kernel/shardingsphere-data-pipeline>

6.8.5 Core Concept

Scaling Job

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

Inventory Data

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

Incremental Data

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

6.8.6 User Norms

Supported

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

Unsupported

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

6.9 Encryption

6.9.1 Definition

Data encryption refers to the modification of some sensitive information through encryption rules in order to offer reliable protection to sensitive private data. Data related to customer security or some

sensitive commercial data, such as ID number, mobile phone number, card number, customer number, and other personal information, shall be encrypted according to the regulations of respective regulations.

6.9.2 Related Concepts

Logic column

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

Cipher column

Encrypted data columns.

Query assistant column

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

Plain column

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

6.9.3 Impact on the system

In real business scenarios, service development teams need to implement and maintain a set of encryption and decryption systems based on the requirements of the security department. When the encryption scenario changes, the self-maintained encryption system often faces the risk of reconstruction or modification. Additionally, for services that have been launched, it is relatively complicated to achieve seamless encrypted transformation in a transparent and secure manner without modifying business logic and SQL.

6.9.4 Limitations

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

- Calculation operations are not supported for encrypted fields, such as AVG, SUM, and computation expressions.

6.9.5 How it works

Apache ShardingSphere parses the SQL entered by users and rewrites the SQL according to the encryption rules provided by users, to encrypt the source data and store the source data (optional) and ciphertext data in the underlying database. When a user queries data, it only retrieves ciphertext data from the database, decrypts it, and finally returns the decrypted source data to the user. Apache ShardingSphere achieves a transparent and automatic data encryption process. Users can use encrypted data as normal data without paying attention to the implementation details of data encryption.

Overall architecture

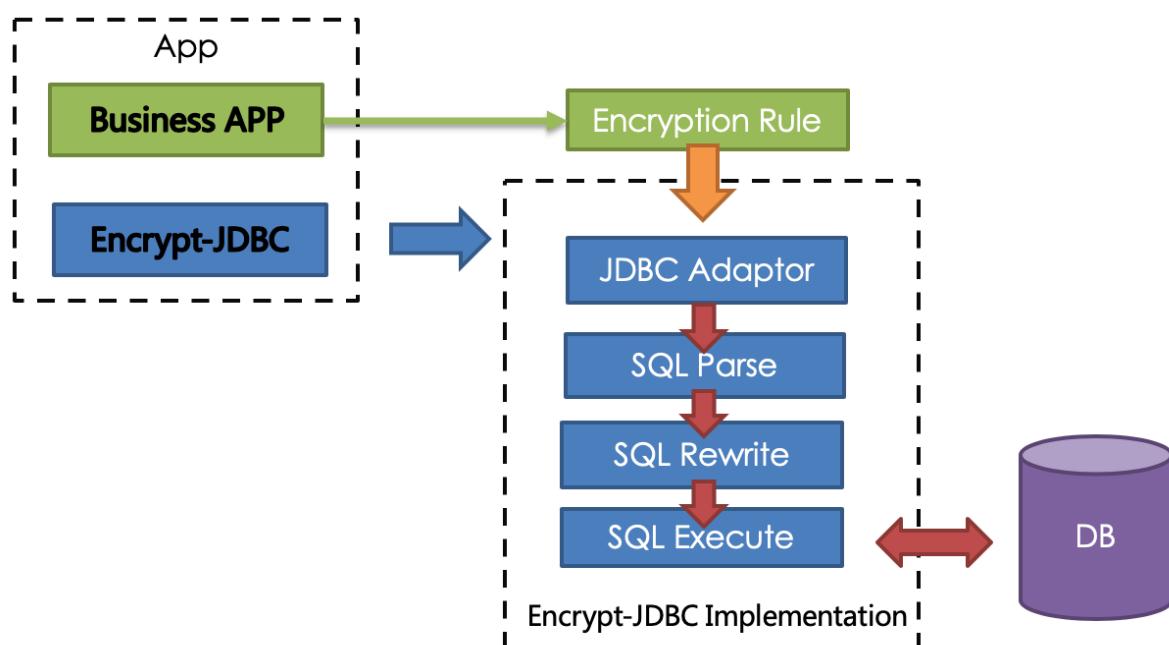


Figure8: 1

The encrypted module intercepts the SQL initiated by the user and parses and understands the SQL behavior through the SQL grammar parser. Then it finds out the fields to be encrypted and the encryption and decryption algorithm according to the encryption rules introduced by the user and interacts with the underlying database. Apache ShardingSphere will encrypt the plaintext requested by users and store it in the underlying database. When the user queries, the ciphertext is extracted from the database, decrypted, and returned to the terminal user. By shielding the data encryption process, users do not need to operate the SQL parsing process, data encryption, and data decryption.

Encryption rules

Before explaining the whole process, we need to understand the encryption rules and configuration. Encryption configuration is mainly divided into four parts: data source configuration, encryptor configuration, encryption table configuration and query attribute configuration, as shown in the figure below:

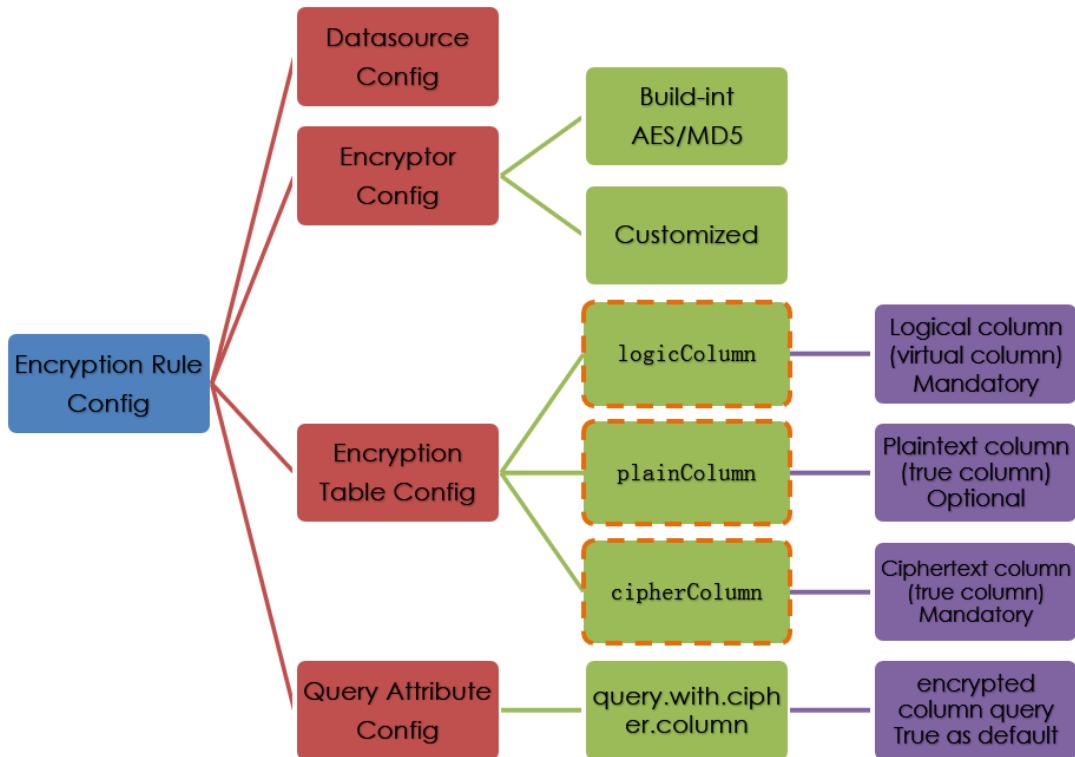


Figure9: 2

Data source configuration: literally the configuration of the data source.

Encryptor configuration: refers to the encryption algorithm used for encryption and decryption. Currently, ShardingSphere has three built-in encryption and decryption algorithms: AES, MD5 and RC4. Users can also implement a set of encryption and decryption algorithms by implementing the interfaces provided by ShardingSphere.

Encryption table configuration: it is used to tell ShardingSphere which column in the data table is used to store ciphertext data (cipherColumn), which column is used to store plaintext data (plainColumn), and which column the user would like to use for SQL writing (logicColumn).

What does it mean by “which column the user would like to use for SQL writing (logicColumn)” ?

We have to know first why the encrypted module exists. The goal of the encrypted module is to shield the underlying data encryption process, which means we don’t want users to know how data is encrypted and decrypted, and how to store plaintext data into plainColumn and ciphertext data into cipherColumn. In other words, we don’t want users to know there is a plainColumn and cipherColumn or how they are used. Therefore, we need to provide the

user with a conceptual column that can be separated from the real column in the underlying database. It may or may not be a real column in the database table so that users can change the column names of plainColumn and cipherColumn of the underlying database at will. Or we can delete plainColumn and never store plaintext, only ciphertext. The only thing we have to ensure is that the user's SQL is written towards the logical column, and the correct mapping relation between logicColumn, plainColumn and cipherColumn can be seen in the encryption rules.

Query attribute configuration: if both plaintext and ciphertext data are stored in the underlying database table, this attribute can be used to determine whether to query the plaintext data in the database table and return it directly, or query the ciphertext data and return it after decryption through Apache ShardingSphere. This attribute can be configured at the table level and the entire rule level. The table-level has the highest priority.

Encryption process

For example, if there is a table named `t_user` in the database, and there actually are two fields in the table: `pwd_plain` for storing plaintext data and `pwd_cipher` for storing ciphertext data, and `logicColumn` is defined as `pwd`, then users should write SQL for `logicColumn`, that is `INSERT INTO t_user SET pwd = '123'`. Apache ShardingSphere receives the SQL and finds that the `pwd` is the `logicColumn` based on the encryption configuration provided by the user. Therefore, it encrypts the logical column and its corresponding plaintext data. **Apache ShardingSphere transforms the column names and data encryption mapping between the logical columns facing users and the plaintext and ciphertext columns facing the underlying database.** As shown in the figure below:

The user's SQL is separated from the underlying data table structure according to the encryption rules provided by the user so that the user's SQL writing does not depend on the real database table structure. The connection, mapping, and transformation between the user and the underlying database are handled by Apache ShardingSphere. The picture below shows the processing flow and conversion logic when the encryption module is used to add, delete, change and check, as shown in the figure below.

6.9.6 Related References

- Configuration: Data Encryption
- Developer Guide: Data Encryption

6.10 Shadow

6.10.1 Definition

Solution for stress testing data governance at the database level, under the online full link stress testing scenario of Apache ShardingSphere.

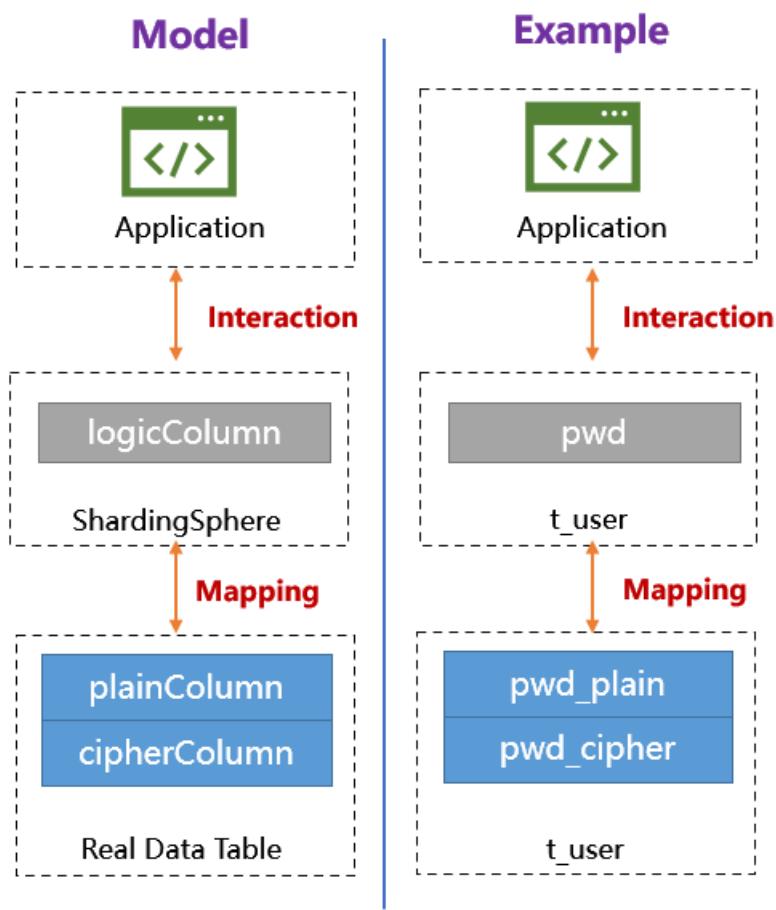


Figure10: 3



Figure11: 4

6.10.2 Related Concepts

Production Database

Database for production data

Shadow Database

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

Shadow Algorithm

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

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

6.10.3 Limitations

Hint based shadow algorithm

No

Column based shadow algorithm

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

- INSERT

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

- SELECT/UPDATE/DELETE

condition categories	SQL	<i>support or not</i>
=	SELECT/UPDATE/DELETE … WHERE column = value	support
LIKE/NOT LIKE	SELECT/UPDATE/DELETE … WHERE column LIKE/NOT LIKE value	support
IN/NOT IN	SELECT/UPDATE/DELETE … WHERE column IN/NOT IN (value1, value2, …)	support
BETWEEN	SELECT/UPDATE/DELETE … WHERE column BETWEEN value1 AND value2	do not support
GROUP BY … HAVING …	SELECT/UPDATE/DELETE … WHERE … GROUP BY column HAVING column > value	do not support
Sub Query	SELECT/UPDATE/DELETE … WHERE column = (SELECT column FROM table WHERE column = value)	do not support

6.11 Observability

6.11.1 Definition

Observing a cluster's operation status in order to quickly grasp the system's current status and efficiently be able to carry out maintenance work, represents a new challenge for distributed systems.

The point-to-point operation and maintenance method of logging into a specific server cannot be applied to scenarios facing a large number of distributed servers.

Telemetry of system-observable data is the recommended way of operating and maintaining distributed systems.

6.11.2 Related Concepts

Agent

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

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

APM

APM is an acronym for Application Performance Monitoring.

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

Tracing

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

Metrics

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

Logging

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

6.11.3 How it works

ShardingSphere-Agent module provides an observable framework for ShardingSphere, which is implemented based on Java Agent.

Metrics, tracing and logging functions are integrated into the agent through plugins, as shown in the following figure:

- The Metrics plugin is used to collect and display statistical indicators for the entire cluster. Apache ShardingSphere supports Prometheus by default.

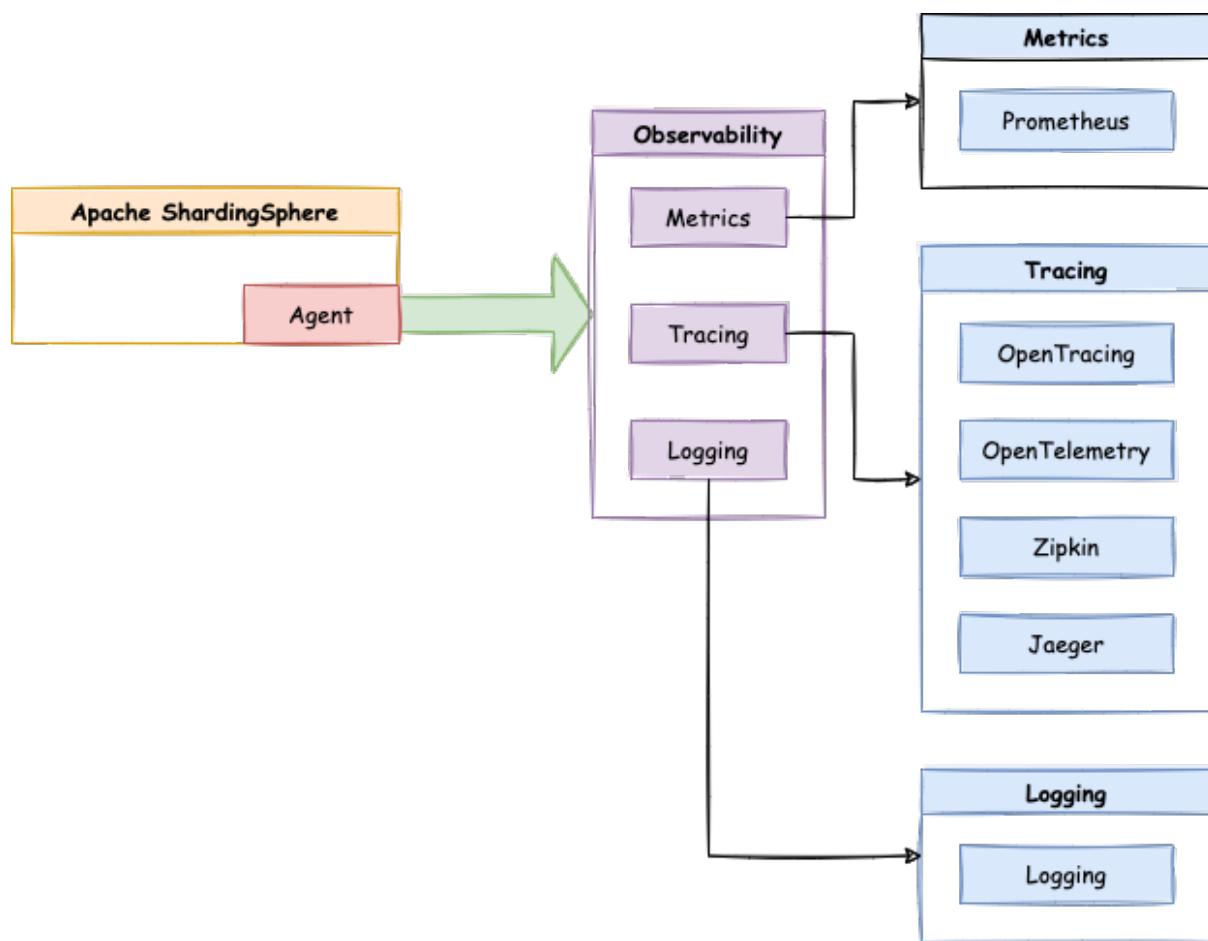


Figure12: Overview

- The tracing plugin is used to obtain the link trace information of SQL parsing and SQL execution. Apache ShardingSphere provides support for Jaeger, OpenTelemetry, OpenTracing(SkyWalking) and Zipkin by default. It also supports users developing customized tracing components through plugin.
- The default logging plugin shows how to record additional logs in ShardingSphere. In practical applications, users need to explore according to their own needs.

6.11.4 Related References

Special API: Observability

6.11.5 Use Norms

Compile source code

Download Apache ShardingSphere from GitHub, Then compile.

```
git clone --depth 1 https://github.com/apache/shardingsphere.git
cd shardingsphere
mvn clean install -Dmaven.javadoc.skip=true -Dcheckstyle.skip=true -Drat.skip=true
-Djacoco.skip=true -DskipITs -DskipTests -Prelease
```

Output directory: shardingsphere-agent/shardingsphere-agent-distribution/target/apache-shardingsphere-\${latest.release.version}-shardingsphere-agent-bin.tar.gz

Agent configuration

- Directory structure

Create agent directory, and unzip agent distribution package to the directory. ``shell mkdir agent tar -zvxf apache-shardingsphere-latest.release.version – shardingsphere – agent – bin.tar.gz – Cagentcdagenttree.¶conf¶ agent.yaml¶ logback.xml¶plugins¶ shardingsphere – agent – logging – base-[latest.release.version].jar ¶ shardingsphere-agent-metrics-prometheus-latest.release.version.jar¶ shardingsphere – agent – tracing – jaeger-[latest.release.version].jar¶ shardingsphere – agent – tracing – opentelemetry-latest.release.version.jar¶ shardingsphere – agent – tracing – opentracing-[latest.release.version].jar¶ shardingsphere-agent-tracing-zipkin-[latest.release.version].jar¶ shardingsphere-agent.jar

* Configuration file

agent.yaml is a configuration file. The plug-ins include Jaeger, opentracing, Zipkin, opentelemetry, logging and Prometheus.

Remove the corresponding plug-in in ignoredpluginnames to start the plug-in.

```

```yaml
applicationName: shardingsphere-agent
ignoredPluginNames:
 - Jaeger
 - OpenTracing
 - Zipkin
 - OpenTelemetry
 - Logging
 - Prometheus

plugins:
 Prometheus:
 host: "localhost"
 port: 9090
 props:
 JVM_INFORMATION_COLLECTOR_ENABLED : "true"
 Jaeger:
 host: "localhost"
 port: 5775
 props:
 SERVICE_NAME: "shardingsphere-agent"
 JAEGER_SAMPLER_TYPE: "const"
 JAEGER_SAMPLER_PARAM: "1"
 Zipkin:
 host: "localhost"
 port: 9411
 props:
 SERVICE_NAME: "shardingsphere-agent"
 URL_VERSION: "/api/v2/spans"
 SAMPLER_TYPE: "const"
 SAMPLER_PARAM: "1"
 OpenTracing:
 props:
 OPENTRACING_TRACER_CLASS_NAME: "org.apache.skywalking.apm.toolkit.
opentracing.SkywalkingTracer"
 OpenTelemetry:
 props:
 otel.resource.attributes: "service.name=shardingsphere-agent"
 otel.traces.exporter: "zipkin"
 Logging:
 props:
 LEVEL: "INFO"
```

```

- Parameter description:

| Name | Description | Value range | Default value |
|------------------------------------|--------------------------------------|--|-----------------------------------|
| JVM_IN_FORMATION_COLLECTOR_ENABLED | Start JVM collector | true, false | true |
| SERVICE_NAME | Tracking service name | Custom | shardingsphere-agent |
| JAEGER_SAMPLER_TYPE | Jaeger sample rate type | const, probabilistic, ratelimiting, remote | const |
| JAEGER_SAMPLER_PARAMETER | Jaeger sampling rate parameter | const:0, 1, probabilistic:0.0 - 1.0, ratelimiting: > 0, Customize the number of acquisitions per second, remote: need to customize the remote service address,JAEGER_SAMPLER_MANAGER_HOST_PORT | 1 (const type) |
| SAMPLER_TYPE | Zipkin sample rate type | const, counting, ratelimiting, boundary | const |
| SAMPLER_PARAM | Zipkin sampling rate parameter | const:0, 1, counting:0.01 - 1.0, ratelimiting: > 0, boundary:0.0001 - 1.0 | 1 (const type) |
| otel.resource.attributes | open-telemetry properties | String key value pair (, split) | service.name=shardingsphere-agent |
| otel.traces.exporter | Tracing exporter | zipkin, jaeger | zipkin |
| otel.traces.sampler | Open-telemetry sample rate type | always_on, always_off, traceidratio | always_on |
| otel.traces.sampler.args | Open-telemetry sample rate parameter | traceidratio: 0.0 - 1.0 | 1.0 |

Used in ShardingSphere-Proxy

- Startup script

Configure the absolute path of shardingsphere-agent.jar to the start.sh startup script of shardingsphere proxy.

```
nohup java ${JAVA_OPTS} ${JAVA_MEM_OPTS} \
-javaagent:/xxxxx/agent/shardingsphere-agent.jar \
-classpath ${CLASS_PATH} ${MAIN_CLASS} >> ${STDOUT_FILE} 2>&1 &
```

- Launch plugin

```
bin/start.sh
```

After normal startup, you can view the startup log of the plugin in the shardingsphere proxy log, and you can view the data at the configured address.

6.12 DistSQL

6.12.1 Definition

DistSQL (Distributed SQL) is Apache ShardingSphere's specific SQL, providing additional operation capabilities compared to standard SQL.

Flexible rule configuration and resource management & control capabilities are one of the characteristics of Apache ShardingSphere.

When using 4.x and earlier versions, developers can operate data just like using a database, but they need to configure resources and rules through YAML file (or registry center). However, the YAML file format and the changes brought by using the registry center made it unfriendly to DBAs.

Starting from version 5.x, DistSQL enables users to operate Apache ShardingSphere just like a database, transforming it from a framework and middleware for developers to a database product for DBAs.

6.12.2 Related Concepts

DistSQL is divided into RDL, RQL, and RAL.

RDL

Resource & Rule Definition Language, is responsible for the definition of resources and rules.

RQL

Resource & Rule Query Language, is responsible for the query of resources and rules.

RAL

Resource & Rule Administration Language, is responsible for the added-on administrator features of hint, transaction type switch, sharding execute planning, etc.

6.12.3 Impact on the System

Before

Before having DistSQL, users used SQL to operate data while using YAML configuration files to manage ShardingSphere, as shown below:

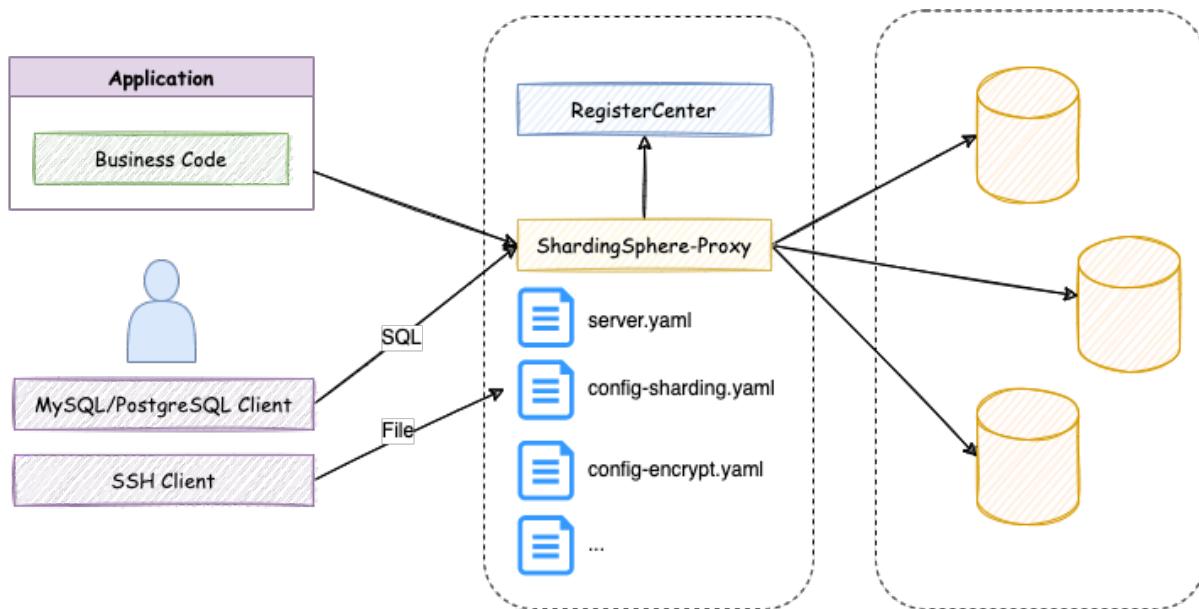


Figure13: Before

At that time, users faced the following problems:

- Different types of clients are required to operate data and manage ShardingSphere configuration.
- Multiple logical databases require multiple YAML files.
- Editing a YAML file requires writing permissions.
- Need to restart ShardingSphere after editing YAML.

After

With the advent of DistSQL, the operation of ShardingSphere has also changed:

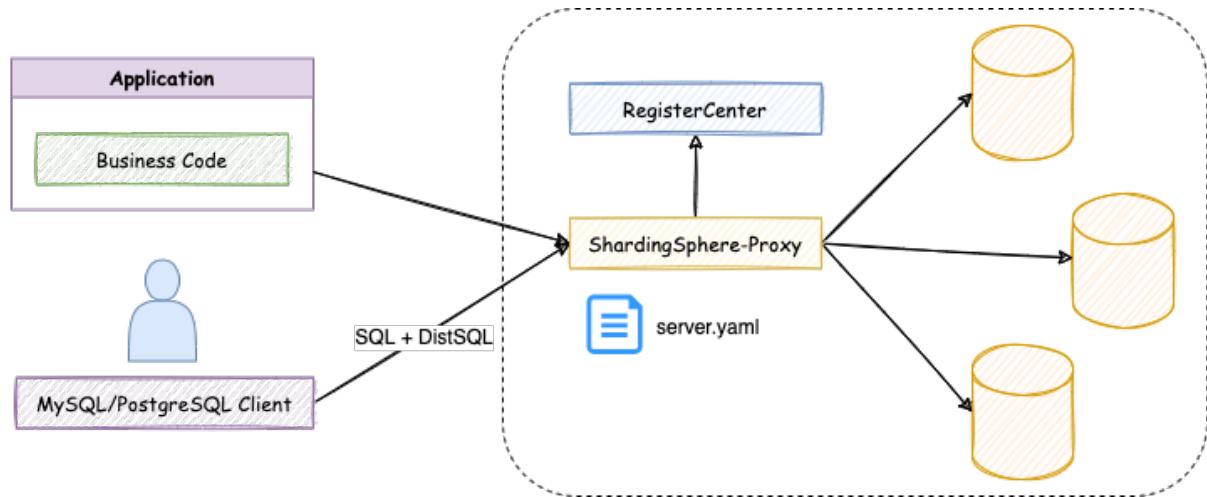


Figure14: After

Now, the user experience has been greatly improved:

- Uses the same client to operate data and ShardingSphere configuration.
- No need for additional YAML files, and the logical databases are managed through DistSQL.
- Editing permissions for files are no longer required, and configuration is managed through DistSQL.
- Configuration changes take effect in real-time without restarting ShardingSphere.

6.12.4 Limitations

DistSQL can be used only with ShardingSphere-Proxy, not with ShardingSphere-JDBC for now.

6.12.5 How it works

Like standard SQL, DistSQL is recognized by the parsing engine of ShardingSphere. It converts the input statement into an abstract syntax tree and then generates the Statement corresponding to each grammar, which is processed by the appropriate Handler.

6.12.6 Related References

User Manual: DistSQL

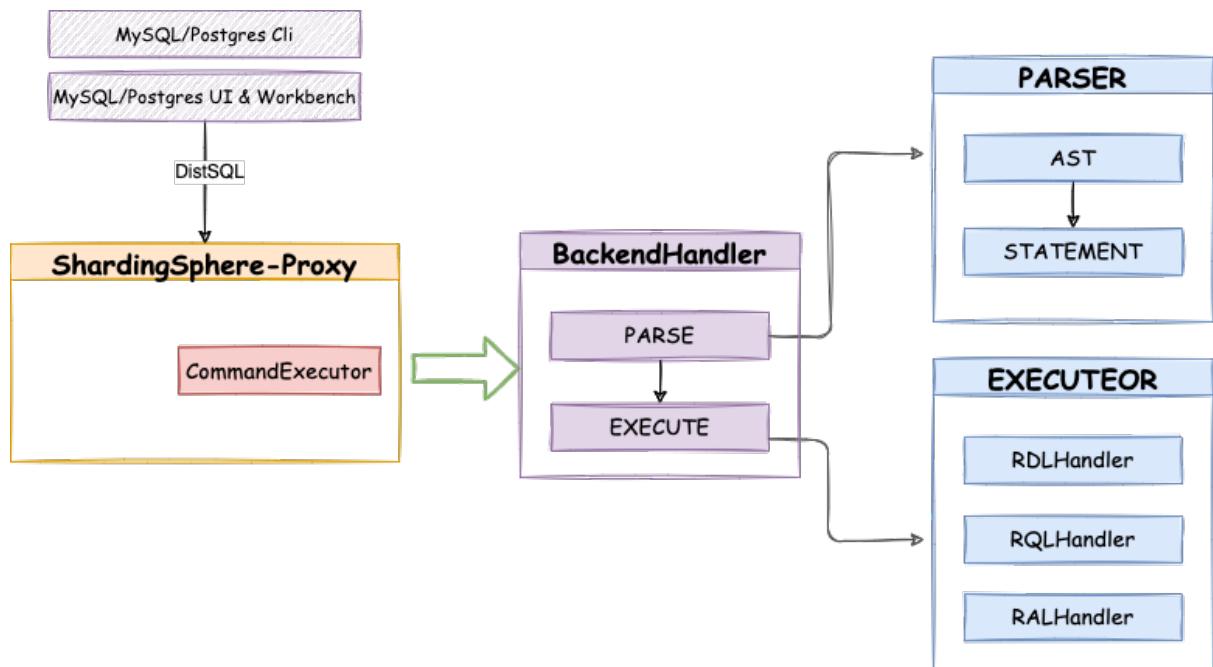


Figure15: Overview

This chapter describes how to use projects of Apache ShardingSphere.

7.1 ShardingSphere-JDBC

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

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

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

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

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

Please refer to [Example](#) for more details.

7.1.1 Java API

Overview

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

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

Usage

Import Maven Dependency

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

Create Data Source

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

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

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

Please refer to [Mode Configuration](#) for more mode details.

Please refer to [Data Source Configuration](#) for more data source details.

Please refer to [Rules Configuration](#) for more rule details.

Use Data Source

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

Take native JDBC usage as an example:

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

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

Mode Configuration

Background

Build the running mode through Java API.

Parameters

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

Attributes:

Name *	<i>Data Type</i>	<i>Description</i>	<i>Default Value</i>
type	String	Type of mode configurationValues could be: Standalone or Cluster	Standalone
repository	PersistentRepositoryConfiguration	Persistent repository configuration Standalone type uses StandalonePersistentRepositoryConfiguration Cluster type uses ClusterPersistentRepositoryConfiguration	
overwrite	boolean	Whether overwrite persistent configuration with local configuration	false

Standalone Persist Configuration

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

Attributes:

Name	<i>Data Type</i>	<i>Description</i>
type	String	Type of persist repository
props	Properties	Properties of persist repository

Cluster Persist Configuration

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

Attributes:

Name	<i>Data Type</i>	<i>Description</i>
type	String	Type of persist repository
namespace	String	Namespace of registry center
server-lists	String	Server lists of registry center
props	Properties	Properties of persist repository

Notes

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

Procedure

Introduce Maven Dependency

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

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

Sample

Standalone Mode

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

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

Cluster Mode (Recommended)

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

private ModeConfiguration createModeConfiguration() {
```

```

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

```

Related References

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

Data Source

Background

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

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

Procedure

1. Import Maven dependency.

```

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

```

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

Sample

```

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

private Map<String, DataSource> createDataSources() {
    Map<String, DataSource> dataSourceMap = new HashMap<>();
    // Configure the 1st data source
    HikariDataSource dataSource1 = new HikariDataSource();
    dataSource1.setDriverClassName("com.mysql.jdbc.Driver");
}

```

```
dataSource1.setJdbcUrl("jdbc:mysql://localhost:3306/ds_1");
dataSource1.setUsername("root");
dataSource1.setPassword("");
dataSourceMap.put("ds_1", dataSource1);

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

Rules

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

Sharding

Background

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

Parameters

Root Configuration

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

Attributes:

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

Sharding Table Configuration

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

Attributes:

Name*	<i>DataType</i>	<i>Description</i>	<i>Default Value</i>
logic Table	String	Name of sharding logic table	.
actualDataNodes (?)	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
databaseShardingStrategy (?)	ShardingStrategyConfiguration	Databases sharding strategy	Use default databases sharding strategy
tableShardingStrategy (?)	ShardingStrategyConfiguration	Tables sharding strategy	Use default tables sharding strategy
keyGeneratorStrategy (?)	KeyGeneratorConfiguration	Key generator configuration	Use default key generator

Sharding Auto Table Configuration

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

Attributes:

<i>Name</i>	<i>DataType</i>	<i>Description</i>	<i>Default Value</i>
logicTable	String	Name of sharding logic table	.
actualDataSources (?)	String	Data source names. Multiple data nodes split by comma	Use all configured data sources
sharding Strategy (?)	ShardingStrategyConfiguration	Sharding strategy	Use default sharding strategy
key Generate Strategy (?)	KeyGeneratorConfiguration	Key generator configuration	Use default key generator

Sharding Strategy Configuration

Standard Sharding Strategy Configuration

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

Attributes:

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

Complex Sharding Strategy Configuration

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

Attributes:

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

Hint Sharding Strategy Configuration

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

Attributes:

Name	DataType	Description
shardingAlgorithmName	String	Sharding algorithm name

None Sharding Strategy Configuration

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

Attributes: None

Please refer to [Built-in Sharding Algorithm List](#) for more details about type of algorithm.

Distributed Key Strategy Configuration

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

Attributes:

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

Please refer to [Built-in Key Generate Algorithm List](#) for more details about type of algorithm.

Procedure

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

Sample

```
public final class ShardingDatabasesAndTablesConfigurationPrecise implements
ExampleConfiguration {

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

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

```

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

private ShardingTableRuleConfiguration get0rderTableRuleConfiguration() {
    ShardingTableRuleConfiguration result = new ShardingTableRuleConfiguration(
"t_order", "demo_ds_${0..1}.t_order_${[0, 1]}");
    result.setKeyGenerateStrategy(new KeyGenerateStrategyConfiguration("order_
id", "snowflake"));
    return result;
}

private ShardingTableRuleConfiguration get0rderItemTableRuleConfiguration() {
    ShardingTableRuleConfiguration result = new ShardingTableRuleConfiguration(
"t_order_item", "demo_ds_${0..1}.t_order_item_${[0, 1]}");
    result.setKeyGenerateStrategy(new KeyGenerateStrategyConfiguration("order_
item_id", "snowflake"));
    return result;
}

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

```

Related References

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

Readwrite-splitting

Background

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

Parameters Explained

Entry

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

Configurable Properties:

Name*	<i>DataType</i>	Description
dataSources (+)	Collection<ReadwriteSplittingDataSourceRuleConfiguration>	Data sources of write and reads
loadBalancers (*)	Map<String, ShardingSphereAlgorithmConfiguration>	Load balance algorithm name and configurations of replica data sources

Primary-secondary Data Source Configuration

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

Configurable Properties:

Name	<i>DataType</i>	Description	<i>Default Value</i>
name	String	Readwrite-splitting data source name	.
staticStrategy	String	Static Readwrite-splitting configuration	.
dynamicStrategy	Properties	Dynamic Readwrite-splitting configuration	.
loadBalancerName (?)	String	Load balance algorithm name of replica sources	Round robin load balance algorithm

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

Configurable Properties:

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

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

Configurable Properties:

Name	DataType	Description	Default Value
autoAwareDataSourceName	String	Database discovery logic data source name	.
writeDataSourceQueryEnabled (?)	String	All read data source are offline, write data source whether the data source is responsible for read traffic	true

Please refer to [Built-in Load Balance Algorithm List](#) for details on algorithm types. Please refer to [ReadWrite splitting-Core features](#) for more details about query consistent routing.

Operating Procedures

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

Configuration Examples

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

```

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

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

```

References

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

Distributed Transaction

Root Configuration

org.apache.shardingsphere.transaction.config.TransactionRuleConfiguration

Attributes:

<i>name</i>	<i>DataType</i>	<i>Description</i>
defaultType	String	Default transaction type
providerType (?)	String	Transaction provider type
props (?)	Properties	Transaction properties

HA**Background**

Build high availability rule configuration through Java API.

Parameters**Root Configuration**

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

Attributes:

Name	Data Type	Description
dataSources (+)	Collection<DatabaseDiscoveryDataSourceRuleConfiguration>	Data source configuration
discoverEveryHeartbeats (+)	Map<String, DatabaseDiscoveryHeartBeatConfiguration>	Detect heartbeat configuration
discoveryTypes (+)	Map<String, ShardingSphereAlgorithmConfiguration>	Database discovery type configuration

Data Source Configuration

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

Attributes:

Name	Data Type	Description	Default Value
groupName (+)	String	Database discovery group name	.
dataSourceNames (+)	Collection<String>	Data source names, multiple data source names separated with comma. Such as: ds_0, ds_1	.
discoverEveryHeartbeatName (+)	String	Detect heartbeat name	.
discoveryTypeName (+)	String	Database discovery type name	.

Detect Heartbeat Configuration

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

Attributes:

Name	Data Type*	Description	Default Value
propops (+)	Properties	Detect heartbeat attribute configuration, keep-alive-cron configuration, cron expression. Such as: '0/5 * * * * ?'	.

Database Discovery Type Configuration

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

Attributes:

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

Procedure

1. Import Maven dependency.

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

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

Sample

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

DataSource dataSource = ShardingSphereDataSourceFactory.createDataSource("database_discovery_db", dataSourceMap, Arrays.asList(createDatabaseDiscoveryConfiguration(), createReadWriteSplittingConfiguration()), null);

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

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

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

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

```

        discoveryHeartBeatConfiguration.put("mgr-heartbeat", new
DatabaseDiscoveryHeartBeatConfiguration(props));
        return discoveryHeartBeatConfiguration;
}

```

Related References

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

Encryption

Root Configuration

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

Attributes:

Name	DataType	Description	Default Value
tables (+)	Collection<EncryptTableRuleConfiguration>	Encrypt table rule configurations	
encryptors (+)	Map<String, ShardingSphereAlgorithmConfiguration>	Encrypt algorithm name and configurations	
queryWithCipherColumn (?)	boolean	Whether query with cipher column for data encrypt. User you can use plaintext to query if have	true

Encrypt Table Rule Configuration

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

Attributes:

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

Encrypt Column Rule Configuration

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

Attributes:

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

Encrypt Algorithm Configuration

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

Attributes:

Name	<i>DataType</i>	<i>Description</i>
name	String	Encrypt algorithm name
type	String	Encrypt algorithm type
properties	Properties	Encrypt algorithm properties

Please refer to [Built-in Encrypt Algorithm List](#) for more details about type of algorithm.

Shadow DB

Root Configuration

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

Attributes:

Name	DataType	Description	Default Value
dataSources	Map<String, ShadowDataSourceConfiguration>	Shadow data source mapping name and configuration	
tables	Map<String, ShadowTableConfiguration>	Shadow table name and configuration	
defaultShadowAlgorithmName	String	Default shadow algorithm name	
shadowAlgorithms	Map<String, ShardingSphereAlgorithmConfiguration>	Shadow algorithm name and configuration	

Shadow Data Source Configuration

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

Attributes:

Name	DataType	Description
sourceDataSourceName	String	Production data source name
shadowDataSourceName	String	Shadow data source name

Shadow Table Configuration

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

Attributes:

Name	DataType	Description
dataSourceNames	Collection<String>	Shadow table location shadow data source mapping names
shadowAlgorithmNames	Collection<String>	Shadow table location shadow algorithm names

Shadow Algorithm Configuration

Please refer to [Built-in Shadow Algorithm List](#).

SQL Parser

Background

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

Parameters

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

Attributes:

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

Cache option Configuration

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

Attributes:

<i>name</i>	• Data Type *	<i>Description</i>	<i>Default Value</i>
initialCapacity	int	Initial capacity of local cache	parser syntax tree local cache default value 128, SQL statement cache default value 2000
maximumSize (?)	long	Maximum capacity of local cache	The default value of local cache for parsing syntax tree is 1024, and the default value of sql statement cache is 65535

Procedure

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

Sample

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

Related References

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

SQL Translator

Root Configuration

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

Attributes:

<i>name</i>	<i>Data Type</i>	<i>Description</i>
<i>type</i>	String	SQL translator type
<i>useOriginalSQLWhenTranslatingFailed</i> (?)	boolean	Whether use original SQL when translating failed

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 read00fwriteDataSource0 = new HikariDataSource();
// ...Omit specific configuration.

HikariDataSource read10fwriteDataSource0 = new HikariDataSource();
// ...Omit specific configuration.

HikariDataSource read00fwriteDataSource1 = new HikariDataSource();
// ...Omit specific configuration.

HikariDataSource read10fwriteDataSource1 = new HikariDataSource();
// ...Omit specific configuration.

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

datasourceMaps.put("write_ds0", writeDataSource0);
datasourceMaps.put("write_ds0_read0", read00fwriteDataSource0);
datasourceMaps.put("write_ds0_read1", read10fwriteDataSource0);
```

```

datasourceMaps.put("write_ds1", writeDataSource1);
datasourceMaps.put("write_ds1_read0", read0fwriteDataSource1);
datasourceMaps.put("write_ds1_read1", read1fwriteDataSource1);

/* 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}");
tOrderRuleConfiguration.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}");
tOrderRuleConfiguration.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();
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("username", "username", "", "username_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 */
Properties readwriteProps1 = new Properties();
readwriteProps1.setProperty("write-data-source-name", "write_ds0");
readwriteProps1.setProperty("read-data-source-names", "write_ds0_read0, write_ds0_
read1");
ReadwriteSplittingDataSourceRuleConfiguration dataSourceConfiguration1 = new
ReadwriteSplittingDataSourceRuleConfiguration("ds_0", "Static", readwriteProps1,
"roundRobin");
Properties readwriteProps2 = new Properties();
readwriteProps2.setProperty("write-data-source-name", "write_ds0");
readwriteProps2.setProperty("read-data-source-names", "write_ds1_read0, write_ds1_
read1");
ReadwriteSplittingDataSourceRuleConfiguration dataSourceConfiguration2 = new
ReadwriteSplittingDataSourceRuleConfiguration("ds_1", "Static", readwriteProps2,
"roundRobin");
```

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

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

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

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

7.1.2 YAML Configuration

Overview

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

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

Usage

Import Maven Dependency

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

YAML Format

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

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

```
# JDBC logic database name. Through this parameter to connect ShardingSphere-JDBC and ShardingSphere-Proxy.  
# Default value: logic_db  
databaseName (?):  
  
mode:  
  
dataSources:  
  
rules:  
- !FOO_XXX  
  ...  
- !BAR_XXX  
  ...  
  
props:  
  key_1: value_1  
  key_2: value_2
```

Please refer to [Mode Configuration](#) for more mode details.

Please refer to [Data Source Configuration](#) for more data source details.

Please refer to [Rules Configuration](#) for more rule details.

Create Data Source

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

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

Use Data Source

Same with Java API.

YAML Syntax Explanation

- ! ! means instantiation of that class
- ! means self-defined alias
- means one or multiple can be included
- [] means array, can substitutable with - each other

Mode Configuration

Parameters

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

Standalone Mode

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

Cluster Mode (recommended)

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

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

Notes

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

Sample

Standalone Mode

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

Cluster Mode (recommended)

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

Related References

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

Data Source

Background

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

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

Parameters

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

Sample

```
dataSources:
  ds_1:
    dataSourceClassName: com.zaxxer.hikari.HikariDataSource
    driverClassName: com.mysql.jdbc.Driver
    jdbcUrl: jdbc:mysql://localhost:3306/ds_1
    username: root
    password:
  ds_2:
    dataSourceClassName: com.zaxxer.hikari.HikariDataSource
    driverClassName: com.mysql.jdbc.Driver
    jdbcUrl: jdbc:mysql://localhost:3306/ds_2
    username: root
    password:
# Configure other data sources
```

Rules

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

Sharding

Background

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

Parameters

```
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
defaultShardingColumn: # Default sharding column name

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

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

```

Procedure

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

Sample

The YAML configuration sample of data sharding is as follows:

```

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

```

```
jdbcUrl: jdbc:mysql://localhost:3306/demo_ds_1?serverTimezone=UTC&useSSL=false&
useUnicode=true&characterEncoding=UTF-8
username: root
password:

rules:
- !SHARDING
tables:
  t_order:
    actualDataNodes: ds_${0..1}.t_order_${0..1}
    tableStrategy:
      standard:
        shardingColumn: order_id
        shardingAlgorithmName: t-order-inline
    keyGenerateStrategy:
      column: order_id
      keyGeneratorName: snowflake
  t_order_item:
    actualDataNodes: ds_${0..1}.t_order_item_${0..1}
    tableStrategy:
      standard:
        shardingColumn: order_id
        shardingAlgorithmName: t_order-item-inline
    keyGenerateStrategy:
      column: order_item_id
      keyGeneratorName: snowflake
  t_account:
    actualDataNodes: ds_${0..1}.t_account_${0..1}
    tableStrategy:
      standard:
        shardingAlgorithmName: t-account-inline
    keyGenerateStrategy:
      column: account_id
      keyGeneratorName: snowflake
defaultShardingColumn: account_id
bindingTables:
- t_order,t_order_item
broadcastTables:
- t_address
defaultDatabaseStrategy:
standard:
  shardingColumn: user_id
  shardingAlgorithmName: database-inline
defaultTableStrategy:
none:

shardingAlgorithms:
  database-inline:
```

```

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

props:
  sql-show: false

```

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

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

Related References

- [Core Feature: Data Sharding](#)
- [Developer Guide: Data Sharding](#)

Readwrite-splitting

Background

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

Parameters

Static Readwrite-splitting

```

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

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

```

Dynamic Readwrite-splitting

```

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

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

```

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

Procedure

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

Sample

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

Related References

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

Distributed Transaction

Background

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

Parameters

- defaultType: transaction mode, optional value LOCAL/XA/BASE.
- providerType: specific implementation of the mode.

Procedure

Use LOCAL Mode

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

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

Use XA Mode

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

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

To manually add Narayana-related dependencies:

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

Use BASE Mode

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

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

Build a Seata Server, add relevant configuration files and Seata dependencies, see [ShardingSphere Integrates Seata Flexible Transactions](#)

HA

Background

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

Parameters

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

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

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

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

Sample

```
databaseName: database_discovery_db

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

rules:
  - !READWRITE_SPLITTING
    dataSources:
      replica_ds:
        dynamicStrategy:
          autoAwareDataSourceName: readwrite_ds
  - !DB_DISCOVERY
    dataSources:
      readwrite_ds:
        dataSourceNames:
          - ds_0
          - ds_1
```

```

- ds_2
discoveryHeartbeatName: mgr-heartbeat
discoveryTypeName: mgr
discoveryHeartbeats:
  mgr-heartbeat:
    props:
      keep-alive-cron: '0/5 * * * * ?'
discoveryTypes:
  mgr:
    type: MySQL.MGR
    props:
      group-name: 558edd3c-02ec-11ea-9bb3-080027e39bd2

```

Related References

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

Encryption

Background

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

Parameters

```

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

```

```

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

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

```

Please refer to [Built-in Encrypt Algorithm List](#) for more details about type of algorithm.

Procedure

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

Sample

The data encryption YAML configurations are as follows:

```

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

rules:
- !ENCRYPT
  tables:
    t_user:
      columns:
        username:
          plainColumn: username_plain
          cipherColumn: username
          encryptorName: name-encryptor
      pwd:
        cipherColumn: pwd
        assistedQueryColumn: assisted_query_pwd
        encryptorName: pwd_encryptor

```

```

encryptors:
  name-encryptor:
    type: AES
    props:
      aes-key-value: 123456abc
  pwd_encryptor:
    type: assistedTest

```

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

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

Related References

- [Core Feature: Data Encryption](#)
- [Developer Guide: Data Encryption](#)

Shadow DB

Background

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

Parameters

Configuration entry

```

rules:
- !SHADOW

```

Configurable attributes

Name	Description	Default
dataSources	shadow DB logical data source mapping the configuration list	none
tables	shadow table configuration list	none
d efaultShadowAlgorithm-Name	name of default shadow algorithm	none, option
shadowAlgorithms	shadow algorithm configuration list	none

Shadow data source configuration

<i>Name</i>	<i>Description</i>	<i>Default</i>
dataSourceName	shadow DB logical data source name	无
sourceDataSourceName	production data source name	无
shadowDataSourceName	shadow data source name	无

Shadow table configuration

<i>Name</i>	<i>Description</i>	<i>De-fault</i>
dataSourceNames	shadow table associates shadow DB logical data source name list	无
shadowAlgorithmNames	shadow table associates shadow algorithm name list	无

Shadow algorithm configuration

<i>Name</i>	<i>Description</i>	<i>Default</i>
type	shadow algorithm type	none
props	shadow algorithm configuration	none

Please refer to [Built-in shadow algorithm list](#) for more details.

Procedure

1. Create production and shadow data sources.
2. Configure shadow rules.
 - Configure the shadow data source.
 - Configure the shadow table.
 - Configure the shadow algorithm.

Sample

```

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

```

Related References

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

SQL-parser

Background

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

Parameters

```

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

```

```
initialCapacity: # Initial capacity of local cache  
maximumSize: # Maximum capacity of local cache
```

Procedure

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

Sample

```
rules:  
- !SQL_PARSER  
  sqlCommentParseEnabled: true  
  sqlStatementCache:  
    initialCapacity: 2000  
    maximumSize: 65535  
  parseTreeCache:  
    initialCapacity: 128  
    maximumSize: 1024
```

Related References

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

SQL Translator

Configuration Item Explanation

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

Mixed Rules

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.
      type: Static
      props:
        write-data-source-name: write_ds # Use the real data source name 'write_
ds'.
        read-data-source-names: read_ds_0, read_ds_1 # Use the real data source
name 'read_ds_0', 'read_ds_1'.
        loadBalancerName: roundRobin
      loadBalancers:
        roundRobin:
          type: ROUND_ROBIN

  props:
    sql-show: true

```

7.1.3 JDBC Driver

Background

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

Parameters

Driver Class Name

`org.apache.shardingsphere.driver.ShardingSphereDriver`

URL Configuration

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

Procedure

1. Import Maven Dependency

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

2. Use drive

- Use native drivers:

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

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

- Use database connection pool:

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

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

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

```
// ...
}
}
}
```

Sample

Load JDBC URL of config.yaml profile in classpath:

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

Load JDBC URL of config.yaml profile in absolute path

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

7.1.4 Spring Boot Starter

Overview

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

Usage

Import Maven Dependency

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

Configure Spring Boot Properties

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

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

Please refer to [Mode Configuration](#) for more mode details.

Please refer to [Data Source Configuration](#) for more data source details.

Please refer to [Rules Configuration](#) for more rule details.

Use Data Source

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

Take native JDBC usage as an example:

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

Mode Configuration

Parameters

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

Standalone Mode

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

Cluster Mode (recommended)

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

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

Notes

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

Sample

Standalone Mode

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

Cluster Mode (recommended)

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

Related References

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

Data Source

Background information

Use local datasource

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

Use datasource JNDI

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

Parameters Explanation

Using local datasource

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

Using JNDI datasource

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

Configuration Examples

Using local datasource

```
# configure actual datasource  
spring.shardingsphere.datasource.names=ds1,ds2  
  
# configure the first datasource  
spring.shardingsphere.datasource.ds1.type=com.zaxxer.hikari.HikariDataSource  
spring.shardingsphere.datasource.ds1.driver-class-name=com.mysql.jdbc.Driver  
spring.shardingsphere.datasource.ds1.jdbc-url=jdbc:mysql://localhost:3306/ds1  
spring.shardingsphere.datasource.ds1.username=root  
spring.shardingsphere.datasource.ds1.password=  
  
# configure the second datasource  
spring.shardingsphere.datasource.ds2.type=com.zaxxer.hikari.HikariDataSource  
spring.shardingsphere.datasource.ds2.driver-class-name=com.mysql.jdbc.Driver  
spring.shardingsphere.datasource.ds2.jdbc-url=jdbc:mysql://localhost:3306/ds2  
spring.shardingsphere.datasource.ds2.username=root  
spring.shardingsphere.datasource.ds2.password=
```

Using JNDI datasource

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

Rules

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

Sharding

Background

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

Parameters

```
spring.shardingsphere.datasource.names= # Omit the data source configuration,  
please refer to the usage  
  
# Standard sharding table configuration  
spring.shardingsphere.rules.sharding.tables.<table-name>.actual-data-nodes= #  
Describe data source names and actual tables, delimiter as point, multiple data  
nodes separated with comma, support inline expression. Absent means sharding  
databases only.  
  
# Databases sharding strategy, use default databases sharding strategy if absent.  
sharding strategy below can choose only one.  
  
# For single sharding column scenario  
spring.shardingsphere.rules.sharding.tables.<table-name>.database-strategy.  
standard.sharding-column= # Sharding column name  
spring.shardingsphere.rules.sharding.tables.<table-name>.database-strategy.  
standard.sharding-algorithm-name= # Sharding algorithm name  
  
# For multiple sharding columns scenario  
spring.shardingsphere.rules.sharding.tables.<table-name>.database-strategy.complex.  
sharding-columns= # Sharding column names, multiple columns separated with comma  
spring.shardingsphere.rules.sharding.tables.<table-name>.database-strategy.complex.  
sharding-algorithm-name= # Sharding algorithm name  
  
# Sharding by hint  
spring.shardingsphere.rules.sharding.tables.<table-name>.database-strategy_hint.  
sharding-algorithm-name= # Sharding algorithm name  
  
# Tables sharding strategy, same as database sharding strategy  
spring.shardingsphere.rules.sharding.tables.<table-name>.table-strategy.xxxx= #  
Omitted
```

```

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

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

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

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

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

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

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

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

```

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

Attention: Inline expression identifier can use \${...} or \$->{...}, but \${...} is conflict

with spring placeholder of properties, so use \${...} on spring environment is better.

Procedure

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

Sample

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

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

spring.shardingsphere.datasource.ds-0.jdbc-url=jdbc:mysql://localhost:3306/demo_ds_0?serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=UTF-8
spring.shardingsphere.datasource.ds-0.type=com.zaxxer.hikari.HikariDataSource
spring.shardingsphere.datasource.ds-0.driver-class-name=com.mysql.jdbc.Driver
spring.shardingsphere.datasource.ds-0.username=root
spring.shardingsphere.datasource.ds-0.password=

spring.shardingsphere.datasource.ds-1.jdbc-url=jdbc:mysql://localhost:3306/demo_ds_1?serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=UTF-8
spring.shardingsphere.datasource.ds-1.type=com.zaxxer.hikari.HikariDataSource
spring.shardingsphere.datasource.ds-1.driver-class-name=com.mysql.jdbc.Driver
spring.shardingsphere.datasource.ds-1.username=root
spring.shardingsphere.datasource.ds-1.password=

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

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

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

```

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

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

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

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

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

```

Related References

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

Readwrite splitting

Background

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

Parameters Explained

Static Readwrite-splitting

```

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

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

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

```

Dynamic Readwrite-splitting

```

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

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

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

```

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

Operating Procedure

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

Configuration Examples

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

References

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

HA

Background

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

Parameters

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

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

```

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

```

Procedure

1. Import MAVEN dependency.

```

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

```

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

Sample

```

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

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

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

```

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

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

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

Related References

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

Encryption

Background

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

Parameters

```

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

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

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

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

```

Please refer to [Built-in Encrypt Algorithm List](#) for more details about type of algorithm.

Procedure

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

Sample

```

spring.shardingsphere.datasource.names=ds

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

spring.shardingsphere.rules.encrypt.encryptors.name-encryptor.type=AES

```

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

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

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

Related References

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

Shadow DB

Background

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

Parameters

Root Configuration

```
spring.shardingsphere.rules.shadow
```

Configurable attributes

Name	Description	Default Value
data-sources	Shadow DB logical data source mapping configuration list	none
tables	Shadow table configuration list	none
shadowAlgorithms	Shadow algorithm configuration list	none
default -shadow-algorithm-name	Default shadow algorithm name	none, options

Shadow Data Source Configuration

Name	Description	Default Value
source-data-source-name	Production data source name	none
shadow-data-source-name	Shadow data source name	none

Shadow Table Configuration

Name	Description	Default Value
data-source-names	影子表关联影子库逻辑数据源名称列表	无
shadow-algorithm-names	影子表关联影子算法名称列表	无

Shadow Algorithm Configuration

Name	Description	Default Value
type	Shadow algorithm type	none
props	Shadow algorithm configuration	none

For details, see [list of built-in shadow algorithms](#)

Procedure

1. Create production and shadow data sources.
2. Configure shadow rules:
 - Configure shadow data sources
 - Configure shadow tables
 - Configure shadow algorithm

Sample

```
spring.shardingsphere.datasource.names= # Omit the data source configuration,  
please refer to the usage  
  
spring.shardingsphere.rules.shadow.data-sources.shadow-data-source.source-data-  
source-name= # Production data source name  
spring.shardingsphere.rules.shadow.data-sources.shadow-data-source.shadow-data-  
source-name= # Shadow data source name  
  
spring.shardingsphere.rules.shadow.tables.<table-name>.data-source-names= # Shadow  
table location shadow data source names (multiple values are separated by ",")  
spring.shardingsphere.rules.shadow.tables.<table-name>.shadow-algorithm-names= #  
Shadow table location shadow algorithm names (multiple values are separated by ",")  
  
spring.shardingsphere.rules.shadow.defaultShadowAlgorithmName= # Default shadow  
algorithm name, optional item.  
  
spring.shardingsphere.rules.shadow.shadow-algorithms.<shadow-algorithm-name>.type= #  
Shadow algorithm type  
spring.shardingsphere.rules.shadow.shadow-algorithms.<shadow-algorithm-name>.props.  
xxx= # Shadow algorithm property configuration
```

Related References

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

SQL Parser

Background

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

Parameters

```
spring.shardingsphere.rules.sql-parser.sql-comment-parse-enabled= # Whether to  
parse SQL comments  
  
spring.shardingsphere.rules.sql-parser.sql-statement-cache.initial-capacity= #  
Initial capacity of SQL statement local cache  
spring.shardingsphere.rules.sql-parser.sql-statement-cache.maximum-size= # Maximum  
capacity of SQL statement local cache  
  
spring.shardingsphere.rules.sql-parser.parse-tree-cache.initial-capacity= # Initial  
capacity of parse tree local cache  
spring.shardingsphere.rules.sql-parser.parse-tree-cache.maximum-size= # Maximum  
local cache capacity of parse tree
```

Procedure

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

Sample

```
spring.shardingsphere.rules.sql-parser.sql-comment-parse-enabled=true  
  
spring.shardingsphere.rules.sql-parser.sql-statement-cache.initial-capacity=2000  
spring.shardingsphere.rules.sql-parser.sql-statement-cache.maximum-size=65535  
  
spring.shardingsphere.rules.sql-parser.parse-tree-cache.initial-capacity=128  
spring.shardingsphere.rules.sql-parser.parse-tree-cache.maximum-size=1024
```

Related References

- [JAVA API: SQL Parser](#)
- [YAML Configuration: SQL Parser](#)
- [Spring Namespace: SQL Parser](#)

Mixed Rules

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

# 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_${0..1}` is the name of the logical data
# source configured with readwrite-splitting
spring.shardingsphere.rules.sharding.sharding-algorithms.default-database-strategy-
inline.algorithm-expression=ds_${0..1}
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

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

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

```

7.1.5 Spring Namespace

Overview

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

Usage

Import Maven Dependency

```

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

```

Configure Spring Bean

Configuration Item Explanation

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

<shardingsphere:data-source />

Name	Type *	Description
id	Attribute	Spring Bean Id
database-name (?)	Attribute	JDBC data source alias
data-sources-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
mode (?)	Tag	Mode configuration
props (?)	Tag	Properties configuration, Please refer to Properties Configuration for more details

Example

```

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

```

```
</beans>
```

Use Data Source

Same with Spring Boot Starter.

Mode Configuration

Parameters Explained

Standalone Mode

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

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

Cluster Mode(Recommended)

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

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

Tips:

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

Operating Procedures

Import MAVEN dependency

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

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

Configuration Example

Standalone Mode

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:shardingsphere="http://shardingsphere.apache.org/schema/
shardingsphere/datasource"
       xmlns:standalone="http://shardingsphere.apache.org/schema/shardingsphere/
mode-repository/standalone"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-
beans.xsd
                           http://shardingsphere.apache.org/schema/shardingsphere/
datasource
                           http://shardingsphere.apache.org/schema/shardingsphere/
datasource/datasource.xsd
                           http://shardingsphere.apache.org/schema/shardingsphere/
mode-repository/standalone
                           http://shardingsphere.apache.org/schema/shardingsphere/
mode-repository/standalone/repository.xsd">
    <standalone:repository id="standaloneRepository" type="H2"/>

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

Cluster Mode

```

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

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

Relevant References

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

Data Source

Background

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

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

Configuration Examples

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

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

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

```
</beans>
```

Rules

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

Sharding

Background

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

Parameters

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

```
<sharding:rule />
```

Name	Type	Description
id	Attribute	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
default-database-strategy-ref (?)	Attribute	Default database strategy name
default-table-strategy-ref (?)	Attribute	Default table strategy name
default-key-generate-strategy-ref (?)	Attribute	Default key generate strategy name
default-sharding-column (?)	Attribute	Default sharding column 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	Attribute	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	Type	Description
id	Attribute	Complex sharding strategy name
sharding-columns	Attribute	Sharding column names, multiple columns separated with comma
algorithm-ref	Attribute	Sharding algorithm name

<sharding:hint-strategy />

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

<sharding:none-strategy />

Name	Type	Description
id	Attribute	Sharding strategy name

<sharding:key-generate-strategy />

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

<sharding:sharding-algorithm />

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

<sharding:key-generate-algorithm />

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

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

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

Procedure

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

Sample

```

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

```

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

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

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

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

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

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

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

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

```

<property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>
</bean>
</beans>

```

Related References

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

Readwrite-splitting

Background

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

Parameters Explained

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

<readwrite-splitting:rule />

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

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

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

<readwrite-splitting:static-strategy />

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

<readwrite-splitting:dynamic-strategy />

Name	Type*	Description
id	Attribute	Dynamic readwrite-splitting name
auto-aware-data-source-name	Attribute	Database discovery logic data source name
write-data-source-query-enabled	Attribute	All read data source are offline, write data source whether the data source is responsible for read traffic
load-balance-algorithm-ref	Attribute	Load balance algorithm name

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

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

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

Operating Procedures

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

Configuration Example

```

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

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

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

```

Related References

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

HA

Background

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

Parameters Explained

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

<database-discovery:rule />

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

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

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

<database-discovery:discovery-heartbeat />

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

<database-discovery:discovery-type />

Name	Type	Description
id	Property	Type name of database discovery
type	Property	Database discovery type, such as: MySQL.MGR
props (?)	Tag	Configuration of database discovery type, such as group-name property configuration of MGR

Operating Procedures

1. Introduce Maven dependency

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

Configuration Example

```

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:cluster="http://shardingsphere.apache.org/schema/shardingsphere/model-
repository/cluster"
       xmlns:shardingsphere="http://shardingsphere.apache.org/schema/
shardingsphere/datasource"
       xmlns:database-discovery="http://shardingsphere.apache.org/schema/
shardingsphere/database-discovery"
       xmlns:readwrite-splitting="http://shardingsphere.apache.org/schema/
shardingsphere/readwrite-splitting"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-
beans.xsd
                           http://shardingsphere.apache.org/schema/shardingsphere/
database-discovery
                           http://shardingsphere.apache.org/schema/shardingsphere/
database-discovery/database-discovery.xsd
                           http://shardingsphere.apache.org/schema/shardingsphere/
readwrite-splitting
                           http://shardingsphere.apache.org/schema/shardingsphere/
readwrite-splitting/readwrite-splitting.xsd
                           http://shardingsphere.apache.org/schema/shardingsphere/
mode-repository/cluster
                           http://shardingsphere.apache.org/schema/shardingsphere/
mode-repository/cluster/repository.xsd
                           http://shardingsphere.apache.org/schema/shardingsphere/
datasource
                           http://shardingsphere.apache.org/schema/shardingsphere/
datasource/datasource.xsd
                           ">
<bean id="ds_0" class="com.zaxxer.hikari.HikariDataSource" destroy-method=
"close">
    <property name="driverClassName" value="com.mysql.jdbc.Driver" />
    <property name="jdbcUrl" value="jdbc:mysql://127.0.0.1:33306/primary_demo_
ds?serverTimezone=UTC&useSSL=false&useUnicode=true&
characterEncoding=UTF-8" />
    <property name="username" value="root" />
    <property name="password" value="" />
</bean>
<bean id="ds_1" class="com.zaxxer.hikari.HikariDataSource" destroy-method=
"close">
    <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
    <property name="jdbcUrl" value="jdbc:mysql://127.0.0.1:33307/primary_demo_
ds?serverTimezone=UTC&useSSL=false&useUnicode=true&
characterEncoding=UTF-8" />
    <property name="username" value="root" />

```

```

        <property name="password" value="" />
    </bean>
    <bean id="ds_2" class="com.zaxxer.hikari.HikariDataSource" destroy-method=
"close">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <property name="jdbcUrl" value="jdbc:mysql://127.0.0.1:33308/primary_demo_
ds?useSSL=false"/>
        <property name="username" value="root"/>
        <property name="password" value="" />
    </bean>
    <cluster:repository id="clusterRepository" type="ZooKeeper" namespace=
"governance" server-lists="localhost:2181">
        <props>
            <prop key="max-retries">3</prop>
            <prop key="operation-timeout-milliseconds">3000</prop>
        </props>
    </cluster:repository>
    <readwrite-splitting:rule id="readWriteSplittingRule">
        <readwrite-splitting:data-source-rule id="replica_ds">
            <readwrite-splitting:dynamic-strategy id="dynamicStrategy" auto-aware-
data-source-name="readwrite_ds" />
        </readwrite-splitting:data-source-rule>
    </readwrite-splitting:rule>
    <database-discovery:rule id="mgrDatabaseDiscoveryRule">
        <database-discovery:data-source-rule id="readwrite_ds" data-source-names=
"ds_0,ds_1,ds_2" discovery-heartbeat-name="mgr-heartbeat" discovery-type-name="mgr"
/>
        <database-discovery:discovery-heartbeat id="mgr-heartbeat">
            <props>
                <prop key="keep-alive-cron" >0/5 * * * * ?</prop>
            </props>
        </database-discovery:discovery-heartbeat>
    </database-discovery:rule>
    <database-discovery:discovery-type id="mgr" type="MySQL.MGR">
        <props>
            <prop key="group-name">558edd3c-02ec-11ea-9bb3-080027e39bd2</prop>
        </props>
    </database-discovery:discovery-type>
    <shardingsphere:data-source id="databaseDiscoveryDataSource" schema-name=
"database-discovery-db" data-source-names="ds_0, ds_1, ds_2" rule-refs=
"readWriteSplittingRule, mgrDatabaseDiscoveryRule">
        <shardingsphere:mode repository-ref="clusterRepository" type="Cluster" />
    </shardingsphere:data-source>
</beans>
```

Related References

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

Encryption

Background

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

Parameters

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

<encrypt:rule />

Name	Type *	Description	Default Value
id	Attribute	Spring Bean Id	
queryWithCipherColumn(?)	Attribute	Whether query with cipher column for data encrypt. User you can use plaintext to query if have	true
table (+)	Tag	Encrypt table configuration	

<encrypt:table />

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

<encrypt:column />

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

<encrypt:encrypt-algorithm />

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

Please refer to [Built-in Encrypt Algorithm List](#) for more details about type of algorithm.

Procedure

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

Sample

```

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

```

```

http://shardingsphere.apache.org/schema/shardingsphere/
datasource/datasource.xsd
http://shardingsphere.apache.org/schema/shardingsphere/
encrypt
http://shardingsphere.apache.org/schema/shardingsphere/
encrypt/encrypt.xsd
">
<context:component-scan base-package="org.apache.shardingsphere.example.core.
mybatis" />

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

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

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

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

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

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

```

```

</bean>

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

```

Related References

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

Shadow DB

Background

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

Parameters

Configuration Entry

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

Configurable Properties:

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

Shadow data source configuration:

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

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

Shadow table configuration:

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

Name	Type	Description
name	At-tribute	Shadow table name
data-sources	At-tribute	Shadow table associated shadow data source name list (multiple values are separated by “,”)
algorithm (?)	Tag	Shadow table association shadow algorithm configuration

```
<shadow:algorithm />
```

Name	Type	Description
shadow-algorithm-ref	Attribute	Shadow table association shadow algorithm name

Shadow algorithm configuration:

```
<shadow:shadow-algorithm />
```

Name	Type	Description
id	Attribute	Shadow algorithm name
type	Attribute	Shadow algorithm type
props (?)	Tag	Shadow algorithm attribute configuration

Refer to [Builin Shadow Algorithm](#) for details

Procedure

1. Create production and shadow data sources.
2. Configure shadow rules.
 - Configure shadow data sources.
 - Configure shadow table.
 - Configure shadow algorithm.

Sample

```

<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:shadow="http://shardingsphere.apache.org/schema/shardingsphere/shadow" xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://shardingsphere.apache.org/schema/shardingsphere/shadow
http://shardingsphere.apache.org/schema/shardingsphere/shadow/shadow.xsd
">
    <shadow:shadow-algorithm id="user-id-insert-match-algorithm" type="VALUE_MATCH">
        <props>
            <prop key="operation">insert</prop>
            <prop key="column">user_id</prop>
            <prop key="value">1</prop>
        </props>
    </shadow:shadow-algorithm>

    <shadow:rule id="shadowRule">
        <shadow:data-source id="shadow-data-source" source-data-source-name="ds" shadow-data-source-name="ds_shadow"/>
            <shadow:shadow-table name="t_user" data-sources="shadow-data-source">
                <shadow:algorithm shadow-algorithm-ref="user-id-insert-match-algorithm"/>
            </shadow:shadow-table>
    </shadow:rule>
</beans>

```

Related References

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

SQL Parser

Background

Spring namespace's SQL parser configuration applies to traditional Spring projects. SQL parsing rules and attributes can be configured through the XML configuration files of the namespace.

Parameters

Namespace: <http://shardingsphere.apache.org/schema/shardingsphere/sql-parser/sql-parser-5.1.2.xsd>

<sql-parser:rule />

Name	Type	Description
id	Attribute	Spring Bean Id
sql-comment-parse-enable	Attribute	Whether to parse SQL comments
parse-tree-cache-ref	Attribute	Parse tree local cache name
sql-statement-cache-ref	Attribute	SQL statement local cache name

<sql-parser:cache-option />

Name	Type	Description
id	Attribute	Local cache configuration item name
initial-capacity	Attribute	Initial capacity of local cache
maximum-size	Attribute	Maximum capacity of local cache

Procedure

1. Set local cache configuration.
2. Set parser configuration.
3. Parse SQL with a parsing engine.

Sample

```
<sql-parser:rule id="sqlParseRule" sql-comment-parse-enable="true" parse-tree-
cache-ref="parseTreeCache" sql-statement-cache-ref="sqlStatementCache" />
<sql-parser:cache-option id="sqlStatementCache" initial-capacity="1024" maximum-
size="1024"/>
<sql-parser:cache-option id="parseTreeCache" initial-capacity="1024" maximum-size=
"1024"/>
```

Related References

- JAVA API: SQL Parser
- YAML Configuration: SQL Parser
- Spring Boot Starter: SQL Parser

Mixed Rules

Configuration Item Explanation

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

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

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

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

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

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

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

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

<!-- readwrite-splitting rule configuration -->
<readwrite-splitting:rule id="readWriteSplittingRule">
    <readwrite-splitting:data-source-rule id="ds_0" type="Static" load-balance-
algorithm-ref="randomStrategy">
        <props>
```

```

        <prop key="write-data-source-name">write_ds0</prop>
        <prop key="read-data-source-names">read_ds0_0, read_ds0_1</prop>
    </props>
</readwrite-splitting:data-source-rule>
<readwrite-splitting:data-source-rule id="ds_1" type="Static" load-balance-
algorithm-ref="randomStrategy">
    <props>
        <prop key="write-data-source-name">write_ds1</prop>
        <prop key="read-data-source-names">read_ds1_0, read_ds1_1</prop>
    </props>
</readwrite-splitting:data-source-rule>
</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="username" cipher-column="username" plain-column="username_plain" encrypt-algorithm-ref="name_encryptor" />
        <encrypt:column logic-column="pwd" cipher-column="pwd" assisted-query-column="assisted_query_pwd" encrypt-algorithm-ref="pwd_encryptor" />
    </encrypt:table>
</encrypt:rule>

<!-- datasource configuration -->
<!-- the element data-source-names's value is all of the datasource name -->
<shardingsphere:data-source id="readQueryDataSource" data-source-names="write_ds0, read_ds0_0, read_ds0_1, write_ds1, read_ds1_0, read_ds1_1"
    rule-refs="readWriteSplittingRule, shardingRule, encryptRule" >
    <props>
        <prop key="sql-show">true</prop>
    </props>
</shardingsphere:data-source>
</beans>
```

7.1.6 Properties Configuration

Background

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

Parameters

Name	. Data Type *	Description	. Default Value *
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
kernel-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-connection-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
sql-federation-enabled (?)	boolean	Whether enable SQL federation	false

Procedure

- Properties configuration is directly configured in the profile used by ShardingSphere-JDBC. The format is as follows:

```
props:  
    sql-show: true
```

Sample

The example of ShardingSphere warehouse contains property configurations of various scenarios. Please refer to: <https://github.com/apache/shardingsphere/tree/master/examples/shardingsphere-jdbc-example>

7.1.7 Builtin Algorithm

Introduction

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

Usage

The builtin algorithms are configured by type and props. Type is defined by the algorithm in SPI, and props is used to deliver the customized parameters of the algorithm.

No matter which configuration type is used, the configured algorithm is named and passed to the corresponding rule configuration. This chapter distinguishes and lists all the builtin algorithms of Apache ShardingSphere according to its functions for developers' reference.

Metadata Repository

H2 Repository

Type: H2

Mode: Standalone

Attributes:

Name	Type *	Description	Default Value
jdbcUrl	String	Database access URL	jdb c:h2:mem:config;DB_CLOSE_DELAY=-1; DATABASE_TO_UPPER=false;MODE=MY
user	String	Database access user-name	sa
password	String	Database access pass-word	

ZooKeeper Repository

Type: ZooKeeper

Mode: Cluster

Attributes:

Name	Type	Description	Default Value
retryIntervalMilliseconds	int	Milliseconds of retry interval	500
maxRetries	int	Max retries of client connection	3
timeToLiveSeconds	int	Seconds of ephemeral data live	60
operationTimeoutMilliseconds	int	Milliseconds of operation timeout	500
digest	String	Password of login	

Etcd Repository

Type: Etcd

Mode: Cluster

Attributes:

Name	Type	Description	Default Value
timeToLiveSeconds	long	Seconds of ephemeral data live	30
connectionTimeout	long	Seconds of connection timeout	30

Sharding Algorithm

Background

ShardingSphere built-in algorithms provide a variety of sharding algorithms, which can be divided into automatic sharding algorithms, standard sharding algorithms, composite sharding algorithms, and hint sharding algorithms, and can meet the needs of most business scenarios of users.

Additionally, considering the complexity of business scenarios, the built-in algorithm also provides a way to customize the sharding algorithm. Users can complete complex sharding logic by writing java code.

Parameters

Auto Sharding Algorithm

Modulo Sharding Algorithm

Type: MOD

Attributes:

Name	DataType	Description
sharding-count	int	Sharding count

Hash Modulo Sharding Algorithm

Type: HASH_MOD

Attributes:

Name	DataType	Description
sharding-count	int	Sharding count

Volume Based Range Sharding Algorithm

Type: VOLUME_RANGE

Attributes:

Name	DataType	Description
range-lower	long	Range lower bound, throw exception if lower than bound
range-upper	long	Range upper bound, throw exception if upper than bound
sharding-volume	long	Sharding volume

Boundary Based Range Sharding Algorithm

Type: BOUNDARY_RANGE

Attributes:

Name	Data Type	Description
sharding_ranges	String	Range of sharding border, multiple boundaries separated by commas

Auto Interval Sharding Algorithm

Type: AUTO_INTERVAL

Attributes:

Name	Data Type*	Description
datetime_lower	String	Shard datetime begin boundary, pattern: yyyy-MM-dd HH:mm:ss
datetime_upper	String	Shard datetime end boundary, pattern: yyyy-MM-dd HH:mm:ss
sharding_seconds	long	Max seconds for the data in one shard, allows sharding key timestamp format seconds with time precision, but time precision after seconds is automatically erased

Standard Sharding Algorithm

Apache ShardingSphere built-in standard sharding algorithm are:

Inline Sharding Algorithm

With Groovy expressions, `InlineShardingStrategy` provides single-key support for the sharding operation of `=` and `IN` in SQL. Simple sharding algorithms can be used through a simple configuration to avoid laborious Java code developments. For example, `t_user_$->{u_id % 8}` means table `t_user` is divided into 8 tables according to `u_id`, with table names from `t_user_0` to `t_user_7`. Please refer to [Inline Expression](#) for more details.

Type: INLINE

Attributes:

Name	<code>*</code> Data Type*	Description	Default Value
algorithm-expression	String	Inline expression sharding algorithm	.
allow-range-query-with-inline-sharding(?)	boolean	Whether range query is allowed. Note: range query will ignore sharding strategy and conduct full routing	false

Interval Sharding Algorithm

This algorithm actively ignores the time zone information of `datetime-pattern`. This means that when `datetime-lower`, `datetime-upper` and the incoming shard key contain time zone information, time zone conversion will not occur due to time zone inconsistencies. When the incoming sharding key is `java.time.Instant`, there is a special case, which will carry the time zone information of the system and convert it into the string format of `datetime-pattern`, and then proceed to the next sharding.

Type: INTERVAL

Attributes:

Name	.	Description	.
	Data Type *		Default Value *
date time -pattern	String	Timestamp pattern of sharding value, must can be transformed to Java LocalDateTime. For example: yyyy-MM-dd HH:mm:ss, yyyy-MM-dd or HH:mm:ss etc. But Gy-MM etc. related to java.time.chrono. JapaneseDate are not supported	.
da teti me-l ower	String	Datetime sharding lower boundary, pattern is defined datetime-pattern	.
da teti me-u pper (?)	String	Datetime sharding upper boundary, pattern is defined datetime-pattern	Now
sha rdin g-su ffix -pattern	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	.
date time -interval-amount (?)	int	Interval of sharding value	1
da teti me-i nter val-unit (?)	String	Unit of sharding value interval, must can be transformed to Java ChronoUnit's Enum value. For example: MONTHS	DAYs

Complex Sharding Algorithm

Complex Inline Sharding Algorithm

Please refer to [Inline Expression](#) for more details.

Type: COMPLEX_INLINE

Name	<code>.</code> Data Type*	Description	Default Value
sharding-columns (?)	String	sharing column names	.
algorithm-expression	String	Inline expression sharding algorithm	.
allow-range-query-with-inline-sharding (?)	boolean	Whether range query is allowed. Note: range query will ignore sharding strategy and conduct full routing	false

Hint Sharding Algorithm

Hint Inline Sharding Algorithm

Please refer to [Inline Expression](#) for more details.

Type: COMPLEX_INLINE

Name	DataType	Description	Default Value
algorithm-expression	String	Inline expression sharding algorithm	\${value}

Class Based Sharding Algorithm

Realize custom extension by configuring the sharding strategy type and algorithm class name. CLASS_BASED allows additional custom properties to be passed into the algorithm class. The passed properties can be retrieved through the `java.util.Properties` class instance with the property name `props`. Refer to Git's `org.apache.shardingsphere.example.extension.sharding.algorithm.classbased.fixture.ClassBasedStandardShardingAlgorithmFixture`.

Type: CLASS_BASED

Attributes:

Name	Data Type	Description
strategy	String	Sharding strategy type, support STANDARD, COMPLEX or HINT (case insensitive)
algorithmClassName	String	Fully qualified name of sharding algorithm

Procedure

- When using data sharding, configure the corresponding data sharding algorithm under the shardingAlgorithms attribute.

Sample

```

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

```

```
- t_order,t_order_item
broadcastTables:
- t_address
defaultDatabaseStrategy:
standard:
    shardingColumn: user_id
    shardingAlgorithmName: database-inline
defaultTableStrategy:
none:

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

Related References

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

Key Generate Algorithm

Snowflake

Type: SNOWFLAKE

Attributes:

Name	• Data Type *	Description	Default Value
max-tolerate-time-difference-milliseconds (?)	long	The max tolerate time for different server's time difference in milliseconds	10 milliseconds
max-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

CosId

Type: COSID

Attributes:

Name *	Data Type *	Description	Default Value
id-name	String	ID generator name	`_share_`
as-string	bool	Whether to generate a string type ID: Convert long type ID to Base-62 String type (Long.MAX_VALUE maximum string length is 11 digits), and ensure the ordering of string IDs	`false`

CosId-Snowflake

Type: COSID_SNOWFLAKE

Attributes:

Name *	Data Type *	Description	Default Value
epoch	String	EPOCH of Snowflake ID Algorithm	`147792960000`
as-string	bool	Whether to generate a string type ID: Convert long type ID to Base-62 String type (Long.MAX_VALUE maximum string length is 11 digits), and ensure the ordering of string IDs	`false`

Load Balance Algorithm

Background

ShardingSphere built-in provides a variety of load balancer algorithms, including polling algorithm, random access algorithm and weight access algorithm, which can meet users' needs in most business scenarios.

Moreover, considering the complexity of the business scenario, the built-in algorithm also provides an extension mode. Users can implement the load balancer algorithm they need based on SPI interface.

Parameters

Type	Describe	Limitations
ROUND_ROBIN	Within the transaction, read query are routed to the primary, and outside the transaction, the round-robin strategy is used to route to the replica	
RANDOM	Within the transaction, read query are routed to the primary, and outside the transaction, the random strategy is used to route to the replica	
WEIGHT	Within the transaction, read query are routed to the primary, and outside the transaction, the weight strategy is used to route to the replica	Attributes need to be configured, attribute name: \${replica-name}, data type: double, attribute name uses the name of the replica, and the parameter fills in the weight value corresponding to the replica. Weight parameter range min > 0, total <= Double.MAX_VALUE.
TRANSACTION_RANDOM	Display/non-display open transaction, read query are routed to multiple replicas using random strategy	
TRANSACTION_ROUND_ROBIN	Display/non-display open transaction, read query are routed to multiple replicas using round-robin strategy	
TRANSACTION_WEIGHT	Display/non-display open transaction, read query are routed to multiple replicas using weight strategy	Attributes need to be configured, attribute name: \${replica-name}, data type: double, attribute name uses the name of the replica, and the parameter fills in the weight value corresponding to the replica. Weight parameter range min > 0, total <= Double.MAX_VALUE.
FIXED_RANDOM	Open transaction displayed, and the replica is routed to a fixed replica using random strategy; otherwise, each read traffic is routed to a different replica using random strategy	
FIXED_ROUND_ROBIN	Open transaction displayed, and the replica is routed to a fixed replica using round-robin strategy; otherwise, each read traffic is routed to a different replica using round-robin strategy	
FIXED_WEIGHT	Open transaction displayed, and the replica is routed to a fixed replica using weight strategy; otherwise, each read traffic is routed to a different replica using weight strategy	Attributes need to be configured, attribute name: \${replica-name}, data type: double, attribute name uses the name of the replica, and the parameter fills in the weight value corresponding to the replica. Weight parameter range min > 0, total <= Double.MAX_VALUE.

Procedure

1. Configure a load balancer algorithm for the loadBalancers attribute to use read/write splitting.

Sample

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

Related References

- Core Feature: Read/Write Splitting
- Developer Guide: Read/Write Splitting

Encryption Algorithm

Background

Encryption algorithms are the algorithms used by the encryption features of Apache ShardingSphere. A variety of algorithms are built-in to make it easy for users to fully leverage the feature.

Parameters

MD5 Encrypt Algorithm

Type: MD5

Attributes: None

AES Encrypt Algorithm

Type: AES

Attributes:

Name	DataType	Description
aes-key-value	String	AES KEY

RC4 Encrypt Algorithm

Type: RC4

Attributes:

Name	DataType	Description
rc4-key-value	String	RC4 KEY

SM3 Encrypt Algorithm

Type: SM3

Attributes:

Name	DataType	Description
sm3-salt	String	SM3 SALT (should be blank or 8 bytes long)

SM4 Encrypt Algorithm

Type: SM4

Attributes:

Name	DataType	Description
sm4-key	String	SM4 KEY (should be 16 bytes)
sm4-mode	String	SM4 MODE (should be CBC or ECB)
sm4-iv	String	SM4 IV (should be specified on CBC, 16 bytes long)
sm4-padding	String	SM4 PADDING (should be PKCS5Padding or PKCS7Padding, NoPadding excepted)

Operating Procedures

1. Configure encryptors in an encryption rule.
2. Use relevant algorithm types in encryptors.

Configuration Examples

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

Related References

- Core Feature: Data Encrypt
- Developer Guide: Data Encrypt

Shadow Algorithm

Background

The shadow DB feature carries out shadow measurement to SQL statements executed. Shadow measurement supports two types of algorithms, and users can choose one or a combination of them based on actual business needs.

Parameters

Column-based shadow algorithm

Column value matching shadow algorithm

Type: VALUE_MATCH

<i>Attribute Name</i>	<i>Data Type</i>	<i>Description</i>
column	String	shadow column
operation	String	SQL operation type (INSERT, UPDATE, DELETE, SELECT)
value	String	value matched by shadow column

Column-based Regex matching algorithm

Type: REGEX_MATCH

<i>Attribute Name</i>	<i>Data Type</i>	<i>Description</i>
column	String	match a column
operation	String	SQL operation type (INSERT, UPDATE, DELETE, SELECT)
regex	String	shadow column matching Regex

Hint-based shadow algorithm

Simple Hint matching shadow algorithm

Type: SIMPLE_HINT

<i>Attribute Name</i>	<i>Data Type</i>	<i>Description</i>
foo	String	bar

Configuration sample

- Java API

```
public final class ShadowConfiguration {
    // ...

    private ShardingSphereAlgorithmConfiguration
createShadowAlgorithmConfiguration() {
    Properties userIdInsertProps = new Properties();
    userIdInsertProps.setProperty("operation", "insert");
    userIdInsertProps.setProperty("column", "user_id");
    userIdInsertProps.setProperty("value", "1");
    return new ShardingSphereAlgorithmConfiguration("VALUE_MATCH",
userIdInsertProps);
}

// ...
}
```

- YAML:

```
shadowAlgorithms:
  user-id-insert-algorithm:
    type: VALUE_MATCH
    props:
      column: user_id
      operation: insert
      value: 1
```

- Spring Boot Starter:

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

- Spring Namespace:

```
<shadow:shadow-algorithm id="user-id-insert-algorithm" type="VALUE_MATCH">
  <props>
    <prop key="operation">insert</prop>
    <prop key="column">user_id</prop>
    <prop key="value">1</prop>
  </props>
</shadow:shadow-algorithm>
```

SQL Translator

Native SQL translator

Type: NATIVE

Attributes:

None

Default SQL translator, does not implement yet.

JooQ SQL translator

Type: JOOQ

Attributes:

None

Because of it need JooQ dependency, ShardingSphere does not include the module, please use below XML to import it by Maven.

```
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-sql-translator-jooq-provider</artifactId>
    <version>${project.version}</version>
</dependency>
```

7.1.8 Special API

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

Sharding

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

Hint

Background

Apache ShardingSphere uses ThreadLocal to manage sharding key values for mandatory routing. A sharding value can be added by programming to the HintManager that takes effect only within the current thread. Apache ShardingSphere can also do mandatory routing by adding comments to SQL.

Main application scenarios for Hint: - The sharding fields do not exist in the SQL and database table structure but in the external business logic. - Certain data operations are forced to be performed in given databases.

Procedure

1. Call HintManager.getInstance() to obtain an instance of HintManager.
2. Use HintManager.addDatabaseShardingValue, HintManager.addTableShardingValue to set the sharding key value.
3. Execute SQL statements to complete routing and execution.
4. Call HintManager.close to clean up the contents of ThreadLocal.

Sample

Sharding with Hint

Hint Configuration

Hint algorithms require users to implement the interface of `org.apache.shardingsphere.api.sharding_hint.HintShardingAlgorithm`. Apache ShardingSphere will acquire sharding values from HintManager to route.

Take the following configurations for reference:

```
rules:
- !SHARDING
  tables:
    t_order:
      actualDataNodes: demo_ds_${0..1}.t_order_${0..1}
      databaseStrategy:
        hint:
          algorithmClassName: xxx.xxx.xxx.HintXXXAlgorithm
      tableStrategy:
        hint:
          algorithmClassName: xxx.xxx.xxx.HintXXXAlgorithm
      defaultTableStrategy:
        none:
      defaultKeyGenerateStrategy:
        type: SNOWFLAKE
        column: order_id

  props:
    sql-show: true
```

Get HintManager

```
HintManager hintManager = HintManager.getInstance();
```

Add Sharding Value

- Use `hintManager.addDatabaseShardingValue` to add sharding key value of data source.
 - Use `hintManager.addTableShardingValue` to add sharding key value of table.
- Users can use `hintManager.setDatabaseShardingValue` to add sharding in hint route to some certain sharding database without sharding tables.

Clean Hint Values

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

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

Codes:

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

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

Related References

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

Readwrite Splitting

This chapter will introduce the Readwrite Splitting API of ShardingSphere-JDBC.

Hint

Background

Apache ShardingSphere uses ThreadLocal to manage primary database routing marks for mandatory routing. A primary database routing mark can be added to HintManager through programming, and this value is valid only in the current thread. Apache ShardingSphere can also route the primary database by adding comments to SQL.

Hint is mainly used to perform mandatory data operations in the primary database under the read/write splitting scenarios.

Procedure

1. Call `HintManager.getInstance()` to obtain HintManager instance.
2. Call `HintManager.setWriteRouteOnly()` method to set the primary database routing marks.
3. Execute SQL statements to complete routing and execution.
4. Call `HintManager.close()` to clear the content of ThreadLocal.

Sample

Primary Route with Hint

Use manual programming

Get HintManager

Be the same as sharding based on hint.

Configure Primary Database Route

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

Clean Hint Value

Be the same as data sharding based on hint.

Codes:

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

Use special SQL comments

Terms of Use

To use SQL Hint function, users need to set `sqlCommentParseEnabled` to true. The comment format only supports `/* */` for now. The content needs to start with `ShardingSphere hint:`, and the attribute name needs to be `writeRouteOnly`.

Codes:

```
/* ShardingSphere hint: writeRouteOnly=true */
SELECT * FROM t_order;
```

Related References

- Core Feature: Readwrite Splitting
- Developer Guide: Readwrite Splitting

Transaction

Using distributed transaction through Apache ShardingSphere is no different from local transaction. In addition to transparent use of distributed transaction, Apache ShardingSphere can switch distributed transaction types every time the database accesses.

Supported transaction types include local, XA and BASE. It can be set before creating a database connection, and default value can be set when Apache ShardingSphere startup.

Use Java API

Background

With ShardingSphere-JDBC, XA and BASE mode transactions can be used through the API.

Prerequisites

Introducing Maven dependency

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

<!-- This module is required when using XA transactions -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-xa-core</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>

<!-- This module is required when using XA's Narayana mode -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-xa-narayana</artifactId>
    <version>${project.version}</version>
</dependency>

<!-- This module is required when using BASE transactions -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-base-seata-at</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>
```

Procedure

1. Set the transaction type
2. Perform the business logic

Sample

```
TransactionTypeHolder.set(TransactionType.XA); // support TransactionType.LOCAL,
TransactionType.XA, TransactionType.BASE
try (Connection conn = dataSource.getConnection()) { // use
ShardingSphereDataSource
    conn.setAutoCommit(false);
    PreparedStatement ps = conn.prepareStatement("INSERT INTO t_order (user_id,
status) VALUES (?, ?)");
    ps.setObject(1, 1000);
    ps.setObject(2, "init");
    ps.executeUpdate();
    conn.commit();
}
```

Use Spring Boot Starter

Background

ShardingSphere-JDBC can be used through spring boot starter. ## Prerequisites

Introducing Maven dependency

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

<!-- This module is required when using XA transactions -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-xa-core</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>

<!-- This module is required when using XA's Narayana mode -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-xa-narayana</artifactId>
    <version>${project.version}</version>
```

```
</dependency>

<!-- This module is required when using BASE transactions -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-base-seata-at</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>
```

Procedure

1. Configure the transaction Type
2. Use distributed transactions

Sample

Configure the transaction Type

```
@Configuration
@EnableTransactionManagement
public class TransactionConfiguration {

    @Bean
    public PlatformTransactionManager txManager(final DataSource dataSource) {
        return new DataSourceTransactionManager(dataSource);
    }

    @Bean
    public JdbcTemplate jdbcTemplate(final DataSource dataSource) {
        return new JdbcTemplate(dataSource);
    }
}
```

Use distributed transactions

```
@Transactional
@ShardingSphereTransactionType(TransactionType.XA) // 支持 TransactionType.LOCAL,
                                                    TransactionType.XA, TransactionType.BASE
public void insert() {
    jdbcTemplate.execute("INSERT INTO t_order (user_id, status) VALUES (?, ?)",
(PreparedStatementCallback<Object>) ps -> {
    ps.setObject(1, i);
    ps.setObject(2, "init");
    ps.executeUpdate();
```

```
});  
}
```

Use Spring Namespace

Background

ShardingSphere-JDBC can be used through spring namespace.

Prerequisites

Introducing Maven dependency

```
<dependency>  
    <groupId>org.apache.shardingsphere</groupId>  
    <artifactId>shardingsphere-jdbc-core-spring-namespace</artifactId>  
    <version>${shardingsphere.version}</version>  
</dependency>  
  
<!-- This module is required when using XA transactions -->  
<dependency>  
    <groupId>org.apache.shardingsphere</groupId>  
    <artifactId>shardingsphere-transaction-xa-core</artifactId>  
    <version>${shardingsphere.version}</version>  
</dependency>  
  
<!-- This module is required when using XA's Narayana mode -->  
<dependency>  
    <groupId>org.apache.shardingsphere</groupId>  
    <artifactId>shardingsphere-transaction-xa-narayana</artifactId>  
    <version>${project.version}</version>  
</dependency>  
  
<!-- This module is required when using BASE transactions -->  
<dependency>  
    <groupId>org.apache.shardingsphere</groupId>  
    <artifactId>shardingsphere-transaction-base-seata-at</artifactId>  
    <version>${shardingsphere.version}</version>  
</dependency>
```

Procedure

1. Configure the transaction manager
2. Use distributed transactions

Sample

Configure the transaction manager

```
<!-- Configuration of ShardingDataSource -->
<!-- ... -->

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

<!-- Enable automatic scanning of @ShardingSphereTransactionType annotation and use Spring's native AOP for class and method enhancements -->
<sharding:tx-type-annotation-driven />
```

Use distributed transactions

```
@Transactional
@ShardingSphereTransactionType(TransactionType.XA) // support TransactionType.LOCAL, TransactionType.XA, TransactionType.BASE
public void insert() {
    jdbcTemplate.execute("INSERT INTO t_order (user_id, status) VALUES (?, ?)",
(PreparedStatementCallback<Object>) ps -> {
        ps.setObject(1, i);
        ps.setObject(2, "init");
        ps.executeUpdate();
    });
}
```

Atomikos Transaction

Background

Apache ShardingSphere provides XA transactions, and the default XA transaction manager is Atomikos.

Procedure

1. Configure the transaction type
2. Configure Atomikos

Sample

Configure the transaction type

Yaml:

```
- !TRANSACTION
  defaultType: XA
  providerType: Atomikos
```

SpringBoot:

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

Spring Namespace:

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

Configure Atomikos

Atomikos configuration items can be customized by adding `jta.properties` to the project's classpath.

See Atomikos' s official documentation for more details.

Data Recovery

`xa_tx.log` is generated in the `logs` directory of the project. This is the log required for recovering XA crash. Do not delete it.

Bitronix Transaction

background

Apache ShardingSphere provides XA transactions that integrate with the Bitronix implementation.

Prerequisites

Introducing Maven dependency

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

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

<!-- This module is required when using XA transactions -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-xa-core</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>

<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-xa-bitronix</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>

<dependency>
    <groupId>org.codehaus.btm</groupId>
    <artifactId>btm</artifactId>
    <version>${btm.version}</version>
</dependency>
```

Procedure

1. Configure the XA transaction type
2. Configure Bitronix

Sample

Configure the XA transaction type

Yaml:

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

SpringBoot:

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

Spring Namespace:

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

Configure Bitronix (Deletable)

See Bitronix's Official Documentation for more details.

Narayana Transaction

Background

Apache ShardingSphere provides XA transactions that integrate with the Narayana implementation.

Prerequisites

Introducing Maven dependency

```
<properties>
    <narayana.version>5.12.4.Final</narayana.version>
    <jboss-transaction-spi.version>7.6.0.Final</jboss-transaction-spi.version>
    <jboss-logging.version>3.2.1.Final</jboss-logging.version>
</properties>

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

<!-- This module is required when using XA transactions -->
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-xa-core</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>

<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-transaction-xa-narayana</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>
<dependency>
    <groupId>org.jboss.narayana.jta</groupId>
    <artifactId>jta</artifactId>
    <version>${narayana.version}</version>
</dependency>
<dependency>
    <groupId>org.jboss.narayana.jts</groupId>
    <artifactId>narayana-jts-integration</artifactId>
    <version>${narayana.version}</version>
</dependency>
<dependency>
    <groupId>org.jboss</groupId>
    <artifactId>jboss-transaction-spi</artifactId>
    <version>${jboss-transaction-spi.version}</version>
</dependency>
<dependency>
    <groupId>org.jboss.logging</groupId>
    <artifactId>jboss-logging</artifactId>
    <version>${jboss-logging.version}</version>
</dependency>
```

Procedure

1. Configure Narayana
2. Set the XA transaction type

Sample

Configure Narayana

Narayana configuration items can be customized by adding `jbosssts-properties.xml` to the project's classpath.

See [Narayana's Official Documentation](#) for more details.

Set the XA transaction type

Yaml:

```
- !TRANSACTION
  defaultType: XA
  providerType: 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

Background

Apache ShardingSphere provides BASE transactions that integrate the Seata implementation.

Procedure

1. Start Seata Server
2. Create the log table
3. Add the Seata configuration

Sample

Start Seata Server

Refer to [seata-work-shop](#) to download and start the Seata server.

Create undo_log table

Create the undo_log table in each shard database instance (take MySQL as an example).

```
CREATE TABLE IF NOT EXISTS `undo_log`
(
  `id`          BIGINT(20)    NOT NULL AUTO_INCREMENT COMMENT 'increment id',
  `branch_id`   BIGINT(20)    NOT NULL COMMENT 'branch transaction id',
  `xid`         VARCHAR(100)  NOT NULL COMMENT 'global transaction id',
  `context`     VARCHAR(128)  NOT NULL COMMENT 'undo_log context,such as
serialization',
  `rollback_info` LONGBLOB    NOT NULL COMMENT 'rollback info',
  `log_status`   INT(11)      NOT NULL COMMENT '0:normal status,1:defense status',
  `log_created`  DATETIME    NOT NULL COMMENT 'create datetime',
  `log_modified` DATETIME    NOT NULL COMMENT 'modify datetime',
  PRIMARY KEY (`id`),
  UNIQUE KEY `ux_undo_log` (`xid`, `branch_id`)
) ENGINE = InnoDB
AUTO_INCREMENT = 1
DEFAULT CHARSET = utf8 COMMENT ='AT transaction mode undo table';
```

Modify configuration

Add the seata.conf file to the classpath.

```
client {
  application.id = example    ## Apply the only primary key
  transaction.service.group = my_test_tx_group    ## The transaction group it
belongs to.
}
```

Modify the file.conf and registry.conf files of Seata as required.

Observability

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

Use Agent

Build

Local Build

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

Download (Not Released Yet)

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

Configuration

Found agent.yaml file:

```
applicationName: shardingsphere-agent  
ignoredPluginNames: # A collection of ignored plugins  
  - Opentracing  
  - Jaeger  
  - Zipkin  
  - Prometheus  
  - OpenTelemetry  
  - Logging  
  
plugins:  
  Prometheus:  
    host: "localhost"  
    port: 9090  
    props:  
      JVM_INFORMATION_COLLECTOR_ENABLED : "true"  
  Jaeger:  
    host: "localhost"  
    port: 5775  
    props:  
      SERVICE_NAME: "shardingsphere-agent"  
      JAAGER_SAMPLER_TYPE: "const"  
      JAAGER_SAMPLER_PARAM: "1"
```

```

    JAEGER_REPORTER_LOG_SPANS: "true"
    JAEGER_REPORTER_FLUSH_INTERVAL: "1"

Zipkin:
  host: "localhost"
  port: 9411
  props:
    SERVICE_NAME: "shardingsphere-agent"
    URL_VERSION: "/api/v2/spans"

Opentracing:
  props:
    OPENTRACING_TRACER_CLASS_NAME: "org.apache.skywalking.apm.toolkit.
opentracing.SkywalkingTracer"

OpenTelemetry:
  props:
    otel.resource.attributes: "service.name=shardingsphere-agent" # Multiple
configurations can be split by ','
    otel.traces.exporter: "zipkin"

Logging:
  props:
    LEVEL: "INFO"

```

Startup

Add arguments in startup script.

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

APM Integration

Usage

Use OpenTracing

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

Add startup arguments

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

Call initialization method.

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

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

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

Notice: when using SkyWalking OpenTracing agent, the OpenTracing plug-in of Apache ShardingSphere Agent cannot be used at the same time to prevent the two plug-ins from conflicting with each other.

Use SkyWalking's Automatic Agent

Please refer to [SkyWalking Manual](#).

Use OpenTelemetry

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

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

Result Demonstration

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

Application Architecture

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

Topology

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.

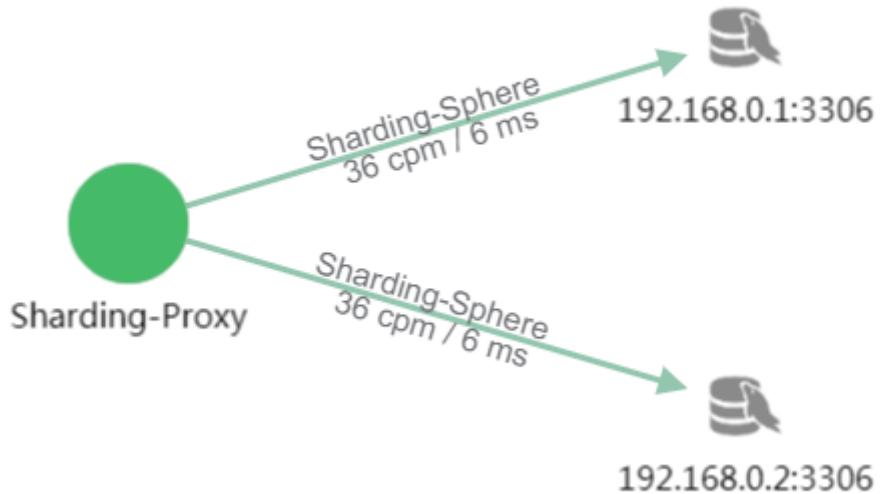


Figure1: The topology diagram

Tracking Data



Figure2: The tracking diagram

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

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

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

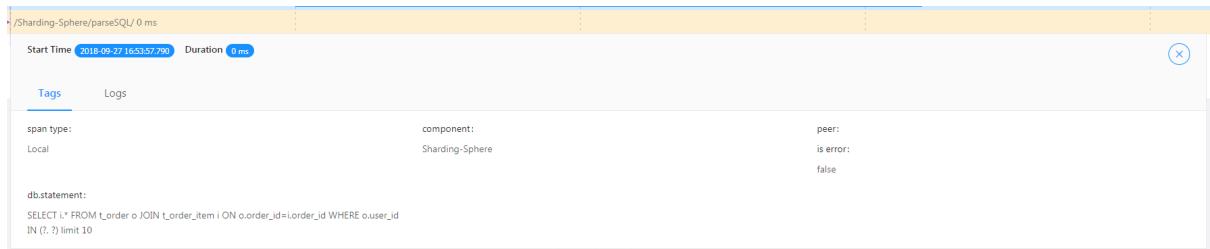


Figure3: The parsing node



Figure4: The actual access node

Exception



Figure5: Exception tracking diagram

Exception nodes can be seen from the tracing diagram.

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

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

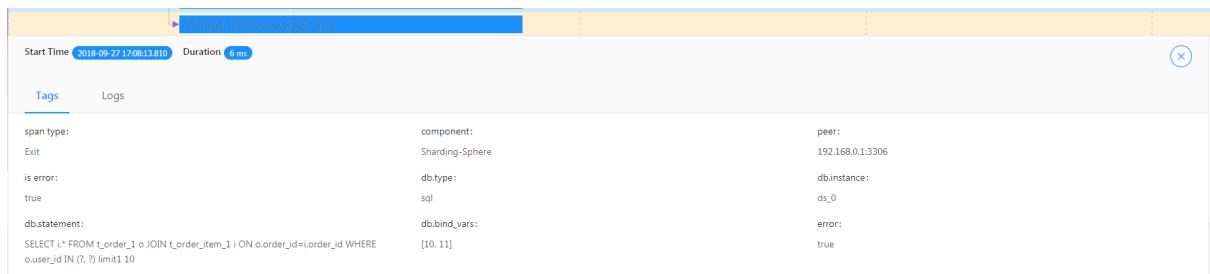


Figure6: Exception node



Figure7: Exception log

7.1.9 Unsupported Items

DataSource Interface

- Do not support timeout related operations

Connection Interface

- Do not support operations of stored procedure, function and cursor
- Do not support native SQL
- Do not support savepoint related operations
- Do not support Schema/Catalog operation
- Do not support self-defined type mapping

Statement and PreparedStatement Interface

- Do not support statements that return multiple result sets (stored procedures, multiple pieces of non-SELECT data)
- Do not support the operation of international characters

ResultSet Interface

- Do not support getting result set pointer position
- Do not support changing result pointer position through none-next method
- Do not support revising the content of result set
- Do not support acquiring international characters
- Do not support getting Array

JDBC 4.1

- Do not support new functions of JDBC 4.1 interface

For all the unsupported methods, please read `org.apache.shardingsphere.driver.jdbc.unsupported` package.

7.2 ShardingSphere-Proxy

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

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

ShardingSphere-Proxy provided YAML configuration, and used DistSQL to communicate. By configuration, application developers can flexibly use data sharding, readwrite-splitting, data encryption, shadow database or the combination of them.

Rule configuration keeps consist with YAML configuration of ShardingSphere-JDBC. DistSQL and YAML can be replaced each other.

Please refer to [Example](#) for more details.

7.2.1 Startup

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

Use Binary Tar

Background

This section describes how to start ShardingSphere-Proxy by binary release packages

Premise

Start the Proxy with a binary package requires an environment with Java JRE 8 or later.

Steps

1. Obtain the binary release package of ShardingSphere-Proxy

Obtain the binary release package of ShardingSphere-Proxy on the [download page](#).

2. Configure `conf/server.yaml`

ShardingSphere-Proxy's operational mode is configured on `server.yaml`, and its configuration mode is the same with that of ShardingSphere-JDBC. Refer to [mode of configuration](#).

Please refer to the following links for other configuration items: * [Permission configuration](#) * [Property configuration](#)

3. Configure `conf/config-*.yaml`

Modify files named with the prefix `config-` in the `conf` directory, such as `conf/config-sharding.yaml` file and configure sharding rules and read/write splitting rules. See [Configuration Manual](#) for configuration methods. The `*` part of the `config-*.yaml` file can be named whatever you want.

ShardingSphere-Proxy supports multiple logical data sources. Each YAML configuration file named with the prefix `config-` is a logical data source.

4. Introduce database driver (Optional)

If the backend is connected to a PostgreSQL or openGauss database, no additional dependencies need to be introduced.

If the backend is connected to a MySQL database, please download `mysql-connector-java-5.1.47.jar` or `mysql-connector-java-8.0.11.jar`, and put it into the `ext-lib` directory.

5. Introduce dependencies required by the cluster mode (Optional)

ShardingSphere-Proxy integrates the ZooKeeper Curator client by default. ZooKeeper is used in cluster mode without introducing other dependencies.

If the cluster mode uses Etcd, the client drivers of Etcd `jetcd-core 0.5.0` need to be copied into the `ext-lib` directory.

6. Introduce dependencies required by distributed transactions (Optional)

It is the same with ShardingSphere-JDBC. Please refer to [Distributed Transaction](#) for more details.

7. Introduce custom algorithm (Optional)

If you need to use a user-defined algorithm class, you can configure custom algorithm in the following ways:

1. Implement the algorithm implementation class defined by `ShardingAlgorithm`.
2. Create a `META-INF/services` directory under the project `resources` directory.
3. Create file `org.apache.shardingsphere.sharding.spi.ShardingAlgorithm` under the directory `META-INF/services`.
4. Writes the fully qualified class name of the implementation class to a file `org.apache.shardingsphere.sharding.spi.ShardingAlgorithm`
5. Package the above Java files into jar packages.
6. Copy the above jar package to the `ext-lib` directory.
7. Configure the Java file reference of the above custom algorithm implementation class in a YAML file, see [Configuration rule](<https://shardingsphere.apache.org/document/current/en/user-manual/shardingsphere-proxy/yaml-config/>) for more details.

8. Start ShardingSphere-Proxy

In Linux or macOS, run `bin/start.sh`. In Windows, run `bin/start.bat` to start ShardingSphere-Proxy. The default listening port is 3307 and the default configuration directory is the `conf` directory in Proxy. The startup script can specify the listening port and the configuration file directory by running the following command:

```
bin/start.sh [port] [/path/to/conf]
```

9. Connect ShardingSphere-Proxy with client

Run the MySQL/PostgreSQL/openGauss client command to directly operate ShardingSphere-Proxy.

Connect ShardingSphere-Proxy with MySQL client:

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

Connect ShardingSphere-Proxy with PostgreSQL:

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

Connect ShardingSphere-Proxy with openGauss client:

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

Sample

Please refer to samples on ShardingSphere repository for complete configuration: <https://github.com/apache/shardingsphere/tree/master/examples/shardingsphere-proxy-example>

Use Docker

Background

This chapter is an introduction about how to start ShardingSphere-Proxy via Docker

Notice

Using Docker to start ShardingSphere-Proxy does not require additional package support.

Steps

1. Acquire Docker Image

- Method 1 (Recommended): Pull from DockerHub

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

- Method 2: Acquire latest master branch image master: <https://github.com/apache/shardingsphere/pkgs/container/shardingsphere-proxy>
- Method 3: Build your own image

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

If the following problems emerge, please make sure Docker daemon Process is running.

```
I/O exception (java.io.IOException) caught when processing request to {}->unix://localhost:80: Connection refused?
```

2. Configure conf/server.yaml and conf/config-*.yaml

Configuration file template can be attained from the Docker container and can be copied to any directory on the host:

```
docker run -d --name tmp --entrypoint=bash apache/shardingsphere-proxy
docker cp tmp:/opt/shardingsphere-proxy/conf /host/path/to/conf
docker rm tmp
```

Since the network conditions inside the container may differ from those of the host, if errors such as “cannot connect to the database” occurs, please make sure that the IP of the database specified in the conf/config-*.yaml configuration file can be accessed from inside the Docker container.

For details, please refer to [ShardingSphere-Proxy quick start manual - binary distribution packages](#).

3. (Optional) Introduce third-party dependencies or customized algorithms

If you have any of the following requirements:
 * ShardingSphere-Proxy Backend use MySQL Database;
 * Implement customized algorithms; * Use Etcd as Registry Center in cluster mode.

Please create `ext-lib` directory anywhere inside the host and refer to the steps in [ShardingSphere-Proxy quick start manual - binary distribution packages](#).

4. Start ShardingSphere-Proxy container

Mount the `conf` and `ext-lib` directories from the host to the container. Start the container:

```
docker run -d \
-v /host/path/to/conf:/opt/shardingsphere-proxy/conf \
-v /host/path/to/ext-lib:/opt/shardingsphere-proxy/ext-lib \
-e PORT=3308 -p13308:3308 apache/shardingsphere-proxy:latest
```

`ext-lib` is not necessary during the process. Users can mount it at will. ShardingSphere-Proxy default portal 3307 can be designated according to environment variable `-e PORT`. Customized JVM related parameters can be set according to environment variable `JVM_OPTS`.

5. Use Client to connect to ShardingSphere-Proxy

Please refer to [ShardingSphere-Proxy quick start manual - binary distribution packages](#).

Configuration Example

For full configuration, please refer to the examples given in ShardingSphere library: <https://github.com/apache/shardingsphere/tree/master/examples/shardingsphere-proxy-example>

Use Helm

Use [Helm](#) to provide guidance for the installation of ShardingSphere-Proxy instance in Kubernetes cluster.

Quick Start

Attention: The following installation method will start ShardingSphere-Proxy with the default `server.yaml` configuration

```
helm repo add shardingsphere https://shardingsphere.apache.org/charts
helm install shardingsphere-proxy shardingsphere/apache-shardingsphere-proxy
```

Step By Step

Requirements

1. Kubernetes 1.18+
2. kubectl
3. Helm 3.2.0+

Use StorageClass to allow dynamic provisioning of Persistent Volumes (PV) for data persistent (optional).

Install

Online installation

1. Add ShardingSphere-Proxy to the local helm repo:

```
helm repo add shardingsphere https://shardingsphere.apache.org/charts
```

2. Install ShardingSphere-Proxy charts: Attention: The following installation method will start ShardingSphere-Proxy with the default server.yaml configuration

```
helm install shardingsphere-proxy shardingsphere/apache-shardingsphere-proxy
```

To modify the configuration:

```
helm pull shardingsphere/apache-shardingsphere-proxy  
tar -zxvf apache-shardingsphere-proxy-1.1.0-chart.tgz  
# Modify the serverConfig section in apache-shardingsphere-proxy/values.yaml  
helm install shardingsphere-proxy apache-shardingsphere-proxy
```

Source installation

```
cd apache-shardingsphere-proxy/charts/governance  
helm dependency build  
cd ../../  
helm dependency build  
cd ..  
helm install shardingsphere-proxy apache-shardingsphere-proxy
```

Charts will be installed with default configuration if above commands executed. Please refer configuration items description below to get more details. Execute `helm list` to acquire all installed releases.

Uninstall

```
helm uninstall shardingsphere-proxy
```

Delete all release records by default, add --keep-history to keep them.

Parameters

Governance-Node parameters

Name	Description	Value
governance.enabled	Switch to enable or disable the governance helm chart	``true``

Governance-Node ZooKeeper parameters

Name	Description	Value
governance.zookeeper.enabled	Switch to enable or disable the ZooKeeper helm chart	true
governance.zookeeper.replicaCount	Number of ZooKeeper nodes	1
governance.zookeeper.persistence.enabled	Enable persistence on ZooKeeper using PVC(s)	`false`
governance.zookeeper.persistence.storageClass	Persistent Volume storage class	""
governance.zookeeper.persistence.accessModes	Persistent Volume access modes	["ReadWriteOnce"]
governance.zookeeper.persistence.size	Persistent Volume size	8Gi
governance.zookeeper.resources.limits	The resources limits for the ZooKeeper containers	{}
governance.zookeeper.resources.requests.memory	The requested memory for the ZooKeeper containers	`256Mi`
governance.zookeeper.resources.requests.cpu	The requested cpu for the ZooKeeper containers	250m

Compute-Node ShardingSphere-Proxy parameters

Name	Description	Value
compute.image.repository	Image name of ShardingSphere-Proxy.	apache/sharding-sphere-proxy
compute.image.pullPolicy	The policy for pulling ShardingSphere-Proxy image	``IfNotPresent``
compute.image.tag	ShardingSphere-Proxy image tag	5.1.2
compute.imagePullSecrets	Specify docker-registry secret names as an array	[]
compute.resources.limits	The resources limits for the ShardingSphere-Proxy containers	{}
compute.resources.requests.memory	The requested memory for the ShardingSphere-Proxy containers	2Gi
compute.resources.requests.cpu	The requested cpu for the ShardingSphere-Proxy containers	200m
compute.replicas	Number of cluster replicas	3
compute.service.type	ShardingSphere-Proxy network mode	ClusterIP
compute.service.port	ShardingSphere-Proxy expose port	3307
compute.mysqlConnector.version	MySQL connector version	5.1.49
compute.startPort	ShardingSphere-Proxy start port	3307

Compute-Node ShardingSphere-Proxy ServerConfiguration authority parameters

Name	Description	Value
compute.serverConfig.authority.privilege.type	authority provider for storage node, the default value is ALL_PERMITTED	ALL_PRIVILEGES_PERMISSIONED
compute.serverConfig.authority.users[0].password	Password for compute node.	root
compute.serverConfig.authority.users[0].user	Username,authorized host for compute node. Format: @hostname is % or empty string means do not care about authorized host	root@%

Compute-Node ShardingSphere-Proxy ServerConfiguration mode Configuration parameters

Name	Description	Value
compute.serverConfig.mode.type	Type of mode configuration. Now only support Cluster mode	Cluster
compute.serverConfig.mode.repository.props.namespace	Namespace of registry center	governance_ds
compute.serverConfig.mode.repository.props.server-lists	Server lists of registry center	<code>{{ printf "%s-zookeeper.%s:2 181" .Release.Name .Release.Namespace }}</code>
compute.serverConfig.mode.repository.props.maxRetries	Max retries of client connection	3
compute.serverConfig.mode.repository.props.operationTimeoutMilliseconds	Milliseconds of operation timeout	5000
compute.serverConfig.mode.repository.props.retryIntervalMilliseconds	Milliseconds of retry interval	500
compute.serverConfig.mode.repository.props.timeToLiveSeconds	Seconds of ephemeral data live	60
<code>'compute.serverConfig.mode.repository.type'</code>	Type of persist repository. Now only support ZooKeeper	ZooKeeper
compute.serverConfig.mode.overwrite	Whether overwrite persistent configuration with local configuration	true

7.2.2 Yaml Configuration

The YAML configuration of ShardingSphere-JDBC is the subset of ShardingSphere-Proxy. In `server.yaml` file, ShardingSphere-Proxy can configure authority feature and more properties for Proxy only.

This chapter will introduce the extra YAML configuration of ShardingSphere-Proxy.

Authority

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

Configuration Item Explanation

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

Example

ALL_PERMITTED

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

DATABASE_PERMITTED

```
rules:
- !AUTHORITY
users:
- root@:root
- my_user@:pwd
provider:
type: DATABASE_PERMITTED
props:
user-database-mappings: root@=sharding_db, root@=test_db, my_user@127.0.0.
1=sharding_db
```

The above configuration means:

- The user `root` can access `sharding_db` when connecting from any host
- The user `root` can access `test_db` when connecting from any host
- The user `my_user` can access `sharding_db` only when connected from `127.0.0.1`

Refer to [Authority Provider](#) for more implementations.

Properties

Background

Apache ShardingSphere can configure system-level configuration through property configuration. This section describes the configuration items in `server.yaml`.

Parameters

Name	• Data Type *	Description	• Default *	• Dynamic Update *
sql-show (?)	boolean	Whether to print SQL in logs. Printing SQL can help developers quickly locate system problems. Logs contain the following contents: logical SQL, authentic SQL and SQL parsing result. If configuration is enabled, logs will use Topic Sharding-Sphere-SQL, and log level is INFO.	false	True
sql-simple (?)	boolean	Whether to print simple SQL in logs.	false	True
kernel-exe cutor-size (?)	int	Set the size of the thread pool for task processing. Each ShardingSphere-DataSource uses an independent thread pool, and different data sources on the same JVM do not share thread pools.	infinite	False
max-connections-size -per-query (?)	int	The maximum number of connections that a query request	1	True
7.2. ShardingSphere-Proxy		can use in each database instance.		226
check-tables	boolean	Whether to check table	false	True

Property configuration can be modified according to [DistSQL#RAL](#). Properties that support dynamic change can take effect immediately. Properties that do not support dynamic change take effect after a restart.

Sample

For a complete sample, please refer to `server.yaml` in ShardingSphere's repository: <https://github.com/apache/shardingsphere/blob/aac0d3026e00575114701be603ec189a02a45747/shardingsphere-proxy/shardingsphere-proxy-bootstrap/src/main/resources/conf/server.yaml#L71-L93>

Rules

Background

This section describes how to configure the rules for ShardingSphere-Proxy.

Parameters Explained

Rules configuration of ShardingSphere-Proxy is the same as that of ShardingSphere-JDBC. For details, please refer to [ShardingSphere-JDBC Rules Configuration](#).

Notice

Unlike ShardingSphere-JDBC, the following rules need to be configured in `server.yaml` of ShardingSphere-Proxy:

- [SQL Parsing](#)
- [Distributed Operations](#)
- [SQL Translator](#)

7.2.3 DistSQL

This chapter will introduce the detailed syntax of DistSQL.

Syntax

This chapter describes the syntax of DistSQL in detail, and introduces use of DistSQL with practical examples.

RDL Syntax

RDL (Resource & Rule Definition Language) responsible for definition of resources/rules.

Resource Definition

Syntax

```

ADD RESOURCE dataSource [, dataSource] ...

ALTER RESOURCE dataSource [, dataSource] ...

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

dataSource:
    simpleSource | urlSource

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

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

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

```

- Before adding resources, please confirm that a distributed database has been created, and execute the `use` command to successfully select a database
- Confirm that the added resource can be connected normally, otherwise it will not be added successfully
- Duplicate `dataSourceName` is not allowed to be added
- In the definition of a `dataSource`, the syntax of `simpleSource` and `urlSource` cannot be mixed
- `poolProperty` is used to customize connection pool properties, `key` must be the same as the connection pool property name, `value` supports int and String types
- `ALTER RESOURCE` is not allowed to change the real data source associated with this resource
- `ALTER RESOURCE` will switch the connection pool. This operation may affect the ongoing business, please use it with caution
- `DROP RESOURCE` will only delete logical resources, not real data sources
- Resources referenced by rules cannot be deleted

- If the resource is only referenced by `single table rule`, and the user confirms that the restriction can be ignored, the optional parameter `ignore single tables` can be added to perform forced deletion

Example

```
ADD RESOURCE resource_0 (
    HOST=127.0.0.1,
    PORT=3306,
    DB=db0,
    USER=root,
    PASSWORD=root
),resource_1 (
    HOST=127.0.0.1,
    PORT=3306,
    DB=db1,
    USER=root
),resource_2 (
    HOST=127.0.0.1,
    PORT=3306,
    DB=db2,
    USER=root,
    PROPERTIES("maximumPoolSize"=10)
),resource_3 (
    URL="jdbc:mysql://127.0.0.1:3306/db3?serverTimezone=UTC&useSSL=false",
    USER=root,
    PASSWORD=root,
    PROPERTIES("maximumPoolSize"=10,"idleTimeout"="30000")
);

ALTER RESOURCE resource_0 (
    HOST=127.0.0.1,
    PORT=3309,
    DB=db0,
    USER=root,
    PASSWORD=root
),resource_1 (
    URL="jdbc:mysql://127.0.0.1:3309/db1?serverTimezone=UTC&useSSL=false",
    USER=root,
    PASSWORD=root,
    PROPERTIES("maximumPoolSize"=10,"idleTimeout"="30000")
);

DROP RESOURCE resource_0, resource_1;
DROP RESOURCE resource_2, resource_3 ignore single tables;
```

Rule Definition

This chapter describes the syntax of rule definition.

Sharding

Syntax

Sharding Table Rule

```
CREATE SHARDING TABLE RULE shardingTableRuleDefinition [,  
shardingTableRuleDefinition] ...  
  
ALTER SHARDING TABLE RULE shardingTableRuleDefinition [,  
shardingTableRuleDefinition] ...  
  
DROP SHARDING TABLE RULE tableName [, tableName] ...  
  
CREATE DEFAULT SHARDING shardingScope STRATEGY (shardingStrategy)  
  
ALTER DEFAULT SHARDING shardingScope STRATEGY (shardingStrategy)  
  
DROP DEFAULT SHARDING shardingScope STRATEGY;  
  
CREATE SHARDING ALGORITHM shardingAlgorithmDefinition [,  
shardingAlgorithmDefinition] ...  
  
ALTER SHARDING ALGORITHM shardingAlgorithmDefinition [,  
shardingAlgorithmDefinition] ...  
  
DROP SHARDING ALGORITHM algorithmName [, algorithmName] ...  
  
CREATE SHARDING KEY GENERATOR keyGeneratorDefinition [, keyGeneratorDefinition] ...  
  
ALTER SHARDING KEY GENERATOR keyGeneratorDefinition [, keyGeneratorDefinition] ...  
  
DROP SHARDING KEY GENERATOR keyGeneratorName [, keyGeneratorName] ...  
  
shardingTableRuleDefinition:  
    shardingAutoTableRule | shardingTableRule  
  
shardingAutoTableRule:  
    tableName(resources, shardingColumn, algorithmDefinition [,  
keyGenerateDeclaration)  
  
shardingTableRule:  
    tableName(dataNodes [, databaseStrategy] [, tableStrategy] [,  
keyGenerateDeclaration)
```

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

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

resource:
    resourceName | inlineExpression

dataNode:
    resourceName | inlineExpression

shardingColumn:
    SHARDING_COLUMN=columnName

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

keyGenerateDeclaration:
    keyGenerateDefinition | keyGenerateConstruction

keyGenerateDefinition:
    KEY_GENERATE_STRATEGY(COLUMN=columnName, strategyDefinition)

shardingScope:
    DATABASE | TABLE

databaseStrategy:
    DATABASE_STRATEGY(shardingStrategy)

tableStrategy:
    TABLE_STRATEGY(shardingStrategy)

keyGenerateConstruction
    KEY_GENERATE_STRATEGY(COLUMN=columnName, KEY_
GENERATOR=keyGenerateAlgorithmName)

shardingStrategy:
    TYPE=strategyType, shardingColumn, shardingAlgorithm

shardingAlgorithm:
    existingAlgorithm | autoCreativeAlgorithm

existingAlgorithm:
    SHARDING_ALGORITHM=shardingAlgorithmName

autoCreativeAlgorithm:
```

```

SHARDING_ALGORITHM(algorithmDefinition)

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

shardingAlgorithmDefinition:
    shardingAlgorithmName(algorithmDefinition)

algorithmProperties:
    algorithmProperty [, algorithmProperty] ...

algorithmProperty:
    key=value

keyGeneratorDefinition:
    keyGeneratorName (algorithmDefinition)

```

- RESOURCES needs to use data source resources managed by RDL
- shardingAlgorithmType specifies the type of automatic sharding algorithm, please refer to [Auto Sharding Algorithm](#)
- keyGenerateStrategyType specifies the distributed primary key generation strategy, please refer to [Key Generate Algorithm](#)
- Duplicate tableName will not be created
- shardingAlgorithm can be reused by different Sharding Table Rule, so when executing DROP SHARDING TABLE RULE, the corresponding shardingAlgorithm will not be removed
- To remove shardingAlgorithm, please execute DROP SHARDING ALGORITHM
- strategyType specifies the sharding strategy, please refer to [Sharding Strategy](#)
- Sharding Table Rule supports both Auto Table and Table at the same time. The two types are different in syntax. For the corresponding configuration file, please refer to [Sharding](#)
- When using the autoCreativeAlgorithm way to specify shardingStrategy, a new sharding algorithm will be created automatically. The algorithm naming rule is tableName_strategyType_shardingAlgorithmType, such as t_order_database_inline

Sharding Binding Table Rule

```

CREATE SHARDING BINDING TABLE RULES bindTableRulesDefinition [, bindTableRulesDefinition] ...

ALTER SHARDING BINDING TABLE RULES bindTableRulesDefinition [, bindTableRulesDefinition] ...

DROP SHARDING BINDING TABLE RULES bindTableRulesDefinition [, bindTableRulesDefinition] ...

```

```
bindTableRulesDefinition:  
  (tableName [, tableName] ... )
```

- ALTER will overwrite the binding table configuration in the database with the new configuration

Sharding Broadcast Table Rule

```
CREATE SHARDING BROADCAST TABLE RULES (tableName [, tableName] ... )  
  
ALTER SHARDING BROADCAST TABLE RULES (tableName [, tableName] ... )  
  
DROP SHARDING BROADCAST TABLE RULES
```

- ALTER will overwrite the broadcast table configuration in the database with the new configuration

Sharding Scaling Rule

```
CREATE SHARDING SCALING RULE scalingName [scalingRuleDefinition]  
  
DROP SHARDING SCALING RULE scalingName  
  
ENABLE SHARDING SCALING RULE scalingName  
  
DISABLE SHARDING SCALING RULE scalingName  
  
scalingRuleDefinition:  
  [inputDefinition] [, outputDefinition] [, streamChannel] [, completionDetector]  
  [, dataConsistencyChecker]  
  
inputDefinition:  
  INPUT ([workerThread] [, batchSize] [, rateLimiter])  
  
outputDefinition:  
  OUTPUT ([workerThread] [, batchSize] [, rateLimiter])  
  
completionDetector:  
  COMPLETION_DETECTOR (algorithmDefinition)  
  
dataConsistencyChecker:  
  DATA_CONSISTENCY_CHECKER (algorithmDefinition)  
  
rateLimiter:  
  RATE_LIMITER (algorithmDefinition)  
  
streamChannel:
```

```

STREAM_CHANNEL (algorithmDefinition)

workerThread:
    WORKER_THREAD=intValue

batchSize:
    BATCH_SIZE=intValue

intValue:
    INT

```

- ENABLE is used to set which sharding scaling rule is enabled;
- DISABLE will disable the sharding scaling rule currently in use;
- Enabled by default when creating the first sharding scaling rule in a logical database.

Example

Sharding Table Rule

Key Generator

```

CREATE SHARDING KEY GENERATOR snowflake_key_generator (
TYPE(NAME=SNOWFLAKE)
);

ALTER SHARDING KEY GENERATOR snowflake_key_generator (
TYPE(NAME=SNOWFLAKE))
;

DROP SHARDING KEY GENERATOR snowflake_key_generator;

```

Auto Table

```

CREATE SHARDING TABLE RULE t_order (
RESOURCES(resource_0,resource_1),
SHARDING_COLUMN=order_id,TYPE(NAME=hash_mod,PROPERTIES("sharding-count"=4)),
KEY_GENERATE_STRATEGY(COLUMN=another_id,TYPE(NAME=snowflake))
);

ALTER SHARDING TABLE RULE t_order (
RESOURCES(resource_0,resource_1,resource_2,resource_3),
SHARDING_COLUMN=order_id,TYPE(NAME=hash_mod,PROPERTIES("sharding-count"=16)),
KEY_GENERATE_STRATEGY(COLUMN=another_id,TYPE(NAME=snowflake))
);

DROP SHARDING TABLE RULE t_order;

```

```
DROP SHARDING ALGORITHM t_order_hash_mod;
```

Table

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

CREATE SHARDING TABLE RULE t_order_item (
    DATANODES("resource_${0..1}.t_order_item_${0..1}"),
    DATABASE_STRATEGY(TYPE=standard,SHARDING_COLUMN=user_id,SHARDING_
    ALGORITHM(TYPE(NAME=inline,PROPERTIES("algorithm-expression"="resource_${user_id % 2}")))),
    TABLE_STRATEGY(TYPE=standard,SHARDING_COLUMN=order_id,SHARDING_ALGORITHM=table_
    inline),
    KEY_GENERATE_STRATEGY(COLUMN=another_id,KEY_GENERATOR=snowflake_key_generator)
);

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

ALTER SHARDING TABLE RULE t_order_item (
    DATANODES("resource_${0..3}.t_order_item${0..3}"),
    DATABASE_STRATEGY(TYPE=standard,SHARDING_COLUMN=user_id,SHARDING_
    ALGORITHM=database_inline),
    TABLE_STRATEGY(TYPE=standard,SHARDING_COLUMN=order_id,SHARDING_ALGORITHM=table_
    inline),
    KEY_GENERATE_STRATEGY(COLUMN=another_id,KEY_GENERATOR=snowflake_key_generator)
);

DROP SHARDING TABLE RULE t_order_item;

DROP SHARDING ALGORITHM database_inline;

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

ALTER DEFAULT SHARDING DATABASE STRATEGY (
    TYPE = standard,SHARDING_COLUMN=another_id,SHARDING_ALGORITHM=database_inline
);

DROP DEFAULT SHARDING DATABASE STRATEGY;
```

Sharding Binding Table Rule

```
CREATE SHARDING BINDING TABLE RULES (t_order,t_order_item),(t_1,t_2);

ALTER SHARDING BINDING TABLE RULES (t_order,t_order_item);

DROP SHARDING BINDING TABLE RULES;

DROP SHARDING BINDING TABLE RULES (t_order,t_order_item);
```

Sharding Broadcast Table Rule

```
CREATE SHARDING BROADCAST TABLE RULES (t_b,t_a);

ALTER SHARDING BROADCAST TABLE RULES (t_b,t_a,t_3);

DROP SHARDING BROADCAST TABLE RULES;
```

Sharding Scaling Rule

```
CREATE SHARDING SCALING RULE sharding_scaling(
INPUT(
    WORKER_THREAD=40,
    BATCH_SIZE=1000
),
OUTPUT(
    WORKER_THREAD=40,
    BATCH_SIZE=1000
),
STREAM_CHANNEL(TYPE(NAME=MEMORY, PROPERTIES("block-queue-size"=10000))),
COMPLETION_DETECTOR(TYPE(NAME=IDLE, PROPERTIES("incremental-task-idle-seconds-threshold"=1800))),
DATA_CONSISTENCY_CHECKER(TYPE(NAME=DATA_MATCH, PROPERTIES("chunk-size"=1000)))
);

ENABLE SHARDING SCALING RULE sharding_scaling;

DISABLE SHARDING SCALING RULE sharding_scaling;

DROP SHARDING SCALING RULE sharding_scaling;
```

Single Table

Definition

```
CREATE DEFAULT SINGLE TABLE RULE singleTableRuleDefinition

ALTER DEFAULT SINGLE TABLE RULE singleTableRuleDefinition

DROP DEFAULT SINGLE TABLE RULE

singleTableRuleDefinition:
    RESOURCE = resourceName
```

- RESOURCE needs to use data source resource managed by RDL

Example

Single Table Rule

```
CREATE DEFAULT SINGLE TABLE RULE RESOURCE = ds_0

ALTER DEFAULT SINGLE TABLE RULE RESOURCE = ds_1

DROP DEFAULT SINGLE TABLE RULE
```

Readwrite-Splitting

Syntax

```
CREATE READWRITE_SPLITTING RULE readwriteSplittingRuleDefinition [, 
readwriteSplittingRuleDefinition] ...

ALTER READWRITE_SPLITTING RULE readwriteSplittingRuleDefinition [, 
readwriteSplittingRuleDefinition] ...

DROP READWRITE_SPLITTING RULE ruleName [, ruleName] ...

readwriteSplittingRuleDefinition:
    ruleName ([staticReadwriteSplittingRuleDefinition | 
dynamicReadwriteSplittingRuleDefinition]
        [, loadBanlancerDefinition])

staticReadwriteSplittingRuleDefinition:
    WRITE_RESOURCE=writeResourceName, READ_RESOURCES(resourceName [, resourceName]
... )
```

```

dynamicReadWriteSplittingRuleDefinition:
    AUTO_AWARE_RESOURCE=resourceName [, WRITE_DATA_SOURCE_QUERY_
ENABLED=writeDataSourceQueryEnabled]

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

algorithmProperties:
    algorithmProperty [, algorithmProperty] ...

algorithmProperty:
    key=value

writeDataSourceQueryEnabled:
    TRUE | FALSE

```

- Support the creation of static readwrite-splitting rules and dynamic readwrite-splitting rules
- Dynamic readwrite-splitting rules rely on database discovery rules
- loadBanlancerType specifies the load balancing algorithm type, please refer to [Load Balance Algorithm](#)
- Duplicate ruleName will not be created

Example

```

// Static
CREATE READWRITE_SPLITTING RULE ms_group_0 (
    WRITE_RESOURCE=write_ds,
    READ_RESOURCES(read_ds_0,read_ds_1),
    TYPE(NAME=random)
);

// Dynamic
CREATE READWRITE_SPLITTING RULE ms_group_1 (
    AUTO_AWARE_RESOURCE=group_0,
    WRITE_DATA_SOURCE_QUERY_ENABLED=false,
    TYPE(NAME=random,PROPERTIES(read_weight='2:1'))
);

ALTER READWRITE_SPLITTING RULE ms_group_1 (
    WRITE_RESOURCE=write_ds,
    READ_RESOURCES(read_ds_0,read_ds_1,read_ds_2),
    TYPE(NAME=random,PROPERTIES(read_weight='2:0'))
);

```

```
DROP READWRITE_SPLITTING RULE ms_group_1;
```

DB Discovery

Syntax

```
CREATE DB_DISCOVERY RULE ruleDefinition [, ruleDefinition] ...

ALTER DB_DISCOVERY RULE ruleDefinition [, ruleDefinition] ...

DROP DB_DISCOVERY RULE ruleName [, ruleName] ...

CREATE DB_DISCOVERY TYPE databaseDiscoveryTypeDefinition [, databaseDiscoveryTypeDefinition] ...

ALTER DB_DISCOVERY TYPE databaseDiscoveryTypeDefinition [, databaseDiscoveryTypeDefinition] ...

DROP DB_DISCOVERY TYPE discoveryTypeName [, discoveryTypeName] ...

CREATE DB_DISCOVERY HEARTBEAT databaseDiscoveryHeartbaetDefinition [, databaseDiscoveryHeartbaetDefinition] ...

ALTER DB_DISCOVERY HEARTBEAT databaseDiscoveryHeartbaetDefinition [, databaseDiscoveryHeartbaetDefinition] ...

DROP DB_DISCOVERY HEARTBEAT discoveryHeartbeatName [, discoveryHeartbeatName] ...

ruleDefinition:
    (databaseDiscoveryRuleDefinition | databaseDiscoveryRuleConstruction)

databaseDiscoveryRuleDefinition
    ruleName (resources, typeDefinition, heartbeatDefinition)

databaseDiscoveryRuleConstruction
    ruleName (resources, TYPE = discoveryTypeName, HEARTBEAT = discoveryHeartbeatName)

databaseDiscoveryTypeDefinition
    discoveryTypeName (typeDefinition)

databaseDiscoveryHeartbaetDefinition
    discoveryHeartbeatName (PROPERTIES (properties))

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

```

typeDefinition:
    TYPE(NAME=typeName [, PROPERTIES([properties] )] )

heartbeatDefinition
    HEARTBEAT (PROPERTIES (properties))

properties:
    property [, property] ...

property:
    key=value

```

- `discoveryType` specifies the database discovery service type, ShardingSphere has built-in support for MySQL.MGR
- Duplicate `ruleName` will not be created
- The `discoveryType` and `discoveryHeartbeat` being used cannot be deleted
- Names with - need to use " " when changing
- When removing the `discoveryRule`, the `discoveryType` and `discoveryHeartbeat` used by the `discoveryRule` will not be removed

Example

When creating a `discoveryRule`, create both `discoveryType` and `discoveryHeartbeat`

```

CREATE DB_DISCOVERY RULE db_discovery_group_0 (
RESOURCES(ds_0, ds_1, ds_2),
TYPE(NAME='MySQL.MGR',PROPERTIES('group-name'='92504d5b-6dec')),
HEARTBEAT(PROPERTIES('keep-alive-cron'='0/5 * * * * ?'))
);

ALTER DB_DISCOVERY RULE db_discovery_group_0 (
RESOURCES(ds_0, ds_1, ds_2),
TYPE(NAME='MySQL.MGR',PROPERTIES('group-name'='246e9612-aaf1')),
HEARTBEAT(PROPERTIES('keep-alive-cron'='0/5 * * * * ?'))
);

DROP DB_DISCOVERY RULE db_discovery_group_0;

DROP DB_DISCOVERY TYPE db_discovery_group_0_mgr;

DROP DB_DISCOVERY HEARTBEAT db_discovery_group_0_heartbeat;

```

Use the existing `discoveryType` and `discoveryHeartbeat` to create a `discoveryRule`

```

CREATE DB_DISCOVERY TYPE db_discovery_group_1_mgr(
    TYPE(NAME='MySQL.MGR',PROPERTIES('group-name'='92504d5b-6dec'))
);

CREATE DB_DISCOVERY HEARTBEAT db_discovery_group_1_heartbeat(
    PROPERTIES('keep-alive-cron'='0/5 * * * * ?')
);

CREATE DB_DISCOVERY RULE db_discovery_group_1 (
RESOURCES(ds_0, ds_1, ds_2),
TYPE=db_discovery_group_1_mgr,
HEARTBEAT=db_discovery_group_1_heartbeat
);

ALTER DB_DISCOVERY TYPE db_discovery_group_1_mgr(
    TYPE(NAME='MySQL.MGR',PROPERTIES('group-name'='246e9612-aaf1'))
);

ALTER DB_DISCOVERY HEARTBEAT db_discovery_group_1_heartbeat(
    PROPERTIES('keep-alive-cron'='0/10 * * * * ?')
);

ALTER DB_DISCOVERY RULE db_discovery_group_1 (
RESOURCES(ds_0, ds_1),
TYPE=db_discovery_group_1_mgr,
HEARTBEAT=db_discovery_group_1_heartbeat
);

DROP DB_DISCOVERY RULE db_discovery_group_1;

DROP DB_DISCOVERY TYPE db_discovery_group_1_mgr;

DROP DB_DISCOVERY HEARTBEAT db_discovery_group_1_heartbeat;

```

Encrypt

Syntax

```

CREATE ENCRYPT RULE encryptRuleDefinition [, encryptRuleDefinition] ...

ALTER ENCRYPT RULE encryptRuleDefinition [, encryptRuleDefinition] ...

DROP ENCRYPT RULE tableName [, tableName] ...

```

```

encryptRuleDefinition:
    tableName(COLUMNS(columnDefinition [, columnDefinition] ...), QUERY_WITH_
CIPHER_COLUMN=queryWithCipherColumn)

columnDefinition:
    (NAME=columnName [, PLAIN=plainColumnName] , CIPHER=cipherColumnName,
encryptAlgorithm)

encryptAlgorithm:
    TYPE(NAME=encryptAlgorithmType [, PROPERTIES([algorithmProperties] )] )

algorithmProperties:
    algorithmProperty [, algorithmProperty] ...

algorithmProperty:
    key=value

```

- PLAIN specifies the plain column, CIPHER specifies the cipher column
- encryptAlgorithmType specifies the encryption algorithm type, please refer to [Encryption Algorithm](#)
- Duplicate tableName will not be created
- queryWithCipherColumn support uppercase or lowercase true or false

Example

```

CREATE ENCRYPT RULE t_encrypt (
COLUMNS(
(NAME=user_id,PLAIN=user_plain,CIPHER=user_cipher,TYPE(NAME=AES,PROPERTIES('aes-
key-value'='123456abc'))),
(NAME=order_id, CIPHER =order_cipher,TYPE(NAME=MD5))
), QUERY_WITH_CIPHER_COLUMN=true),
t_encrypt_2 (
COLUMNS(
(NAME=user_id,PLAIN=user_plain,CIPHER=user_cipher,TYPE(NAME=AES,PROPERTIES('aes-
key-value'='123456abc'))),
(NAME=order_id, CIPHER=order_cipher,TYPE(NAME=MD5))
), QUERY_WITH_CIPHER_COLUMN=FALSE);

ALTER ENCRYPT RULE t_encrypt (
COLUMNS(
(NAME=user_id,PLAIN=user_plain,CIPHER=user_cipher,TYPE(NAME=AES,PROPERTIES('aes-
key-value'='123456abc'))),
(NAME=order_id,CIPHER=order_cipher,TYPE(NAME=MD5))
), QUERY_WITH_CIPHER_COLUMN=TRUE);

```

```
DROP ENCRYPT RULE t_encrypt,t_encrypt_2;
```

Shadow

Syntax

```
CREATE SHADOW RULE shadowRuleDefinition [, shadowRuleDefinition] ...

ALTER SHADOW RULE shadowRuleDefinition [, shadowRuleDefinition] ...

CREATE SHADOW ALGORITHM shadowAlgorithm [, shadowAlgorithm] ...

ALTER SHADOW ALGORITHM shadowAlgorithm [, shadowAlgorithm] ...

DROP SHADOW RULE ruleName [, ruleName] ...

DROP SHADOW ALGORITHM algorithmName [, algorithmName] ...

CREATE DEFAULT SHADOW ALGORITHM NAME = algorithmName

shadowRuleDefinition: ruleName(resourceMapping, shadowTableRule [, shadowTableRule]
...)

resourceMapping: SOURCE=resourceName, SHADOW=resourceName

shadowTableRule: tableName(shadowAlgorithm [, shadowAlgorithm] ...)

shadowAlgorithm: ([algorithmName, ] TYPE(NAME=shadowAlgorithmType,
PROPERTIES([algorithmProperties] ...)))

algorithmProperties: algorithmProperty [, algorithmProperty] ...

algorithmProperty: key=value
```

- Duplicate ruleName cannot be created
- resourceMapping specifies the mapping relationship between the source database and the shadow library. You need to use the resource managed by RDL, please refer to [resource](#)
- shadowAlgorithm can act on multiple shadowTableRule at the same time
- If algorithmName is not specified, it will be automatically generated according to ruleName, tableName and shadowAlgorithmType
- shadowAlgorithmType currently supports VALUE_MATCH, REGEX_MATCH and SIMPLE_HINT
- shadowTableRule can be reused by different shadowRuleDefinition, so when executing `DROP SHADOW RULE`, the corresponding shadowTableRule will not be removed

- shadowAlgorithm can be reused by different shadowTableRule, so when executing ALTER SHADOW RULE, the corresponding shadowAlgorithm will not be removed

Example

```
CREATE SHADOW RULE shadow_rule(
SOURCE=demo_ds,
SHADOW=demo_ds_shadow,
t_order((simple_hint_algorithm, TYPE(NAME=SIMPLE_HINT, PROPERTIES("shadow"="true",
foo="bar"))),(TYPE(NAME=REGEX_MATCH, PROPERTIES("operation"="insert","column"=
"user_id", "regex"='[1]'))),
t_order_item((TYPE(NAME=VALUE_MATCH, PROPERTIES("operation"="insert","column"=
"user_id", "value"='1'))));

ALTER SHADOW RULE shadow_rule(
SOURCE=demo_ds,
SHADOW=demo_ds_shadow,
t_order((simple_hint_algorithm, TYPE(NAME=SIMPLE_HINT, PROPERTIES("shadow"="true",
foo="bar"))),(TYPE(NAME=REGEX_MATCH, PROPERTIES("operation"="insert","column"=
"user_id", "regex"='[1]'))),
t_order_item((TYPE(NAME=VALUE_MATCH, PROPERTIES("operation"="insert","column"=
"user_id", "value"='1'))));

CREATE SHADOW ALGORITHM
(simple_hint_algorithm, TYPE(NAME=SIMPLE_HINT, PROPERTIES("shadow"="true", "foo"=
"bar")),
(user_id_match_algorithm, TYPE(NAME=REGEX_MATCH,PROPERTIES("operation"="insert",
"column"="user_id", "regex"='[1]')));

ALTER SHADOW ALGORITHM
(simple_hint_algorithm, TYPE(NAME=SIMPLE_HINT, PROPERTIES("shadow"="false", "foo"=
"bar")),
(user_id_match_algorithm, TYPE(NAME=VALUE_MATCH,PROPERTIES("operation"="insert",
"column"="user_id", "value"='1')));

DROP SHADOW RULE shadow_rule;

DROP SHADOW ALGORITHM simple_hint_algorithm;

CREATE DEFAULT SHADOW ALGORITHM NAME = simple_hint_algorithm;
```

RQL Syntax

RQL (Resource & Rule Query Language) responsible for resources/rules query.

Resource Query

Syntax

```
SHOW DATABASE RESOURCES [FROM databaseName]
```

Return Value Description

Column	Description
name	Data source name
type	Data source type
host	Data source host
port	Data source port
db	Database name
attribute	Data source attribute

Example

```
mysql> SHOW DATABASE RESOURCES;
+-----+-----+-----+-----+-----+
| name | type  | host      | port | db    | connection_timeout_milliseconds | idle_
timeout_milliseconds | max_lifetime_milliseconds | max_pool_size | min_pool_size |
read_only | other_attributes
+-----+-----+-----+-----+-----+
```


Rule Query

This chapter describes the syntax of rule query.

Sharding

Syntax

Sharding Table Rule

```
SHOW SHARDING TABLE tableRule | RULES [FROM databaseName]

SHOW SHARDING ALGORITHMS [FROM databaseName]

SHOW UNUSED SHARDING ALGORITHMS [FROM databaseName]

SHOW SHARDING AUDITORS [FROM databaseName]

SHOW SHARDING TABLE RULES USED ALGORITHM algorithmName [FROM databaseName]

SHOW SHARDING KEY GENERATORS [FROM databaseName]

SHOW UNUSED SHARDING KEY GENERATORS [FROM databaseName]

SHOW SHARDING TABLE RULES USED KEY GENERATOR keyGeneratorName [FROM databaseName]

SHOW DEFAULT SHARDING STRATEGY

SHOW SHARDING TABLE NODES

tableRule:
    RULE tableName
```

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

Sharding Binding Table Rule

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

Sharding Broadcast Table Rule

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

Sharding Scaling Rule

```
SHOW SHARDING SCALING RULES [FROM databaseName]
```

Return Value Description

Sharding Table Rule

Column	Description
table	Logical table name
actual_data_nodes	Actual data node
actual_data_sources	Actual data source (Displayed when creating rules by RDL)
database_strategy_type	Database sharding strategy type
database_sharding_column	Database sharding column
database_sharding_algorithm_type	Database sharding algorithm type
database_sharding_algorithm_props	Database sharding algorithm properties
table_strategy_type	Table sharding strategy type
table_sharding_column	Table sharding column
table_sharding_algorithm_type	Table sharding algorithm type
table_sharding_algorithm_props	Table sharding algorithm properties
key_generate_column	Sharding key generator column
key_generator_type	Sharding key generator type
key_generator_props	Sharding key generator properties

Sharding Algorithms

Column	Description
name	Sharding algorithm name
type	Sharding algorithm type
props	Sharding algorithm properties

Unused Sharding Algorithms

Column	Description
name	Sharding algorithm name
type	Sharding algorithm type
props	Sharding algorithm properties

Sharding auditors

Column	Description
name	Sharding audit algorithm name
type	Sharding audit algorithm type
props	Sharding audit algorithm properties

Sharding key generators

Column	Description
name	Sharding key generator name
type	Sharding key generator type
props	Sharding key generator properties

Unused Sharding Key Generators

Column	Description
name	Sharding key generator name
type	Sharding key generator type
props	Sharding key generator properties

Default Sharding Strategy

Column	Description
name	Strategy name
type	Sharding strategy type
sharding_column	Sharding column
sharding_algorithm_name	Sharding algorithm name
sharding_algorithm_type	Sharding algorithm type
sharding_algorithm_props	Sharding algorithm properties

Sharding Table Nodes

Column	Description
name	Sharding rule name
nodes	Sharding nodes

Sharding Binding Table Rule

Column	Description
sharding_binding_tables	sharding Binding Table list

Sharding Broadcast Table Rule

Column	Description
sharding_broadcast_tables	sharding Broadcast Table list

Sharding Scaling Rule

Column	Description
name	name of sharding scaling rule
input	data read configuration
output	data write configuration
stream_channel	algorithm of stream channel
completion_detector	algorithm of completion detecting
data_consistency_checker	algorithm of data consistency checking

Example

Sharding Table Rule

SHOW SHARDING TABLE RULES

SHOW SHARDING TABLE RULE tableName

```
mysql> SHOW SHARDING TABLE RULE t_order;
```

```

| table | actual_data_nodes | actual_data_sources | database_strategy_
type | database_sharding_column | database_sharding_algorithm_type | database_
sharding_algorithm_props | table_strategy_type | table_sharding_column |
table_sharding_algorithm_type | table_sharding_algorithm_props | |
key_generate_column | key_generator_type | key_generator_props |
+-----+-----+-----+
+-----+-----+
-----+-----+
-----+-----+
-----+-----+
| t_order | ds_${0..1}.t_order_${0..1} | | INLINE | |
user_id | INLINE | algorithm-expression:ds_$
{user_id % 2} | INLINE | order_id | INLINE |
| algorithm-expression:t_order_${order_id % 2} | order_id | SNOWFLAKE
| |
+-----+-----+
+-----+-----+
-----+-----+
-----+-----+
-----+-----+
-----+-----+
1 row in set (0.01 sec)

```

SHOW SHARDING ALGORITHMS

```

mysql> SHOW SHARDING ALGORITHMS;
+-----+-----+
| name | type | props |
+-----+-----+
| t_order_inline | INLINE | algorithm-expression=t_order_${order_id % 2} |
| t_order_item_inline | INLINE | algorithm-expression=t_order_item_${order_id % 2} |
+-----+-----+
2 row in set (0.01 sec)

```

SHOW UNUSED SHARDING ALGORITHMS

```

mysql> SHOW UNUSED SHARDING ALGORITHMS;
+-----+-----+-----+
| name | type | props |
+-----+-----+
| t1_inline | INLINE | algorithm-expression=t_order_${order_id % 2} |
+-----+-----+
1 row in set (0.01 sec)

```

SHOW SHARDING AUDITORS

```
mysql> SHOW SHARDING AUDITORS;
+-----+-----+-----+
| name | type | props |
+-----+-----+-----+
| dml_audit | DML_SHARDING_CONDITIONS |      |
+-----+-----+-----+
2 row in set (0.01 sec)
```

SHOW SHARDING TABLE RULES USED ALGORITHM algorithmName

```
mysql> SHOW SHARDING TABLE RULES USED ALGORITHM t_order_inline;
+-----+-----+
| type | name   |
+-----+-----+
| table | t_order |
+-----+-----+
1 row in set (0.01 sec)
```

SHOW SHARDING KEY GENERATORS

```
mysql> SHOW SHARDING KEY GENERATORS;
+-----+-----+-----+
| name | type | props |
+-----+-----+-----+
| t_order_snowflake | snowflake |      |
| t_order_item_snowflake | snowflake |      |
| uuid_key_generator | uuid |      |
+-----+-----+-----+
3 row in set (0.01 sec)
```

SHOW UNUSED SHARDING KEY GENERATORS

```
mysql> SHOW UNUSED SHARDING KEY GENERATORS;
+-----+-----+-----+
| name | type | props |
+-----+-----+-----+
| uuid_key_generator | uuid |      |
+-----+-----+-----+
1 row in set (0.01 sec)
```

SHOW SHARDING TABLE RULES USED KEY GENERATOR keyGeneratorName

```
mysql> SHOW SHARDING TABLE RULES USED KEY GENERATOR keyGeneratorName;
+-----+-----+
| type | name   |
+-----+-----+
| table | t_order |
+-----+-----+
```

```
1 row in set (0.01 sec)
```

SHOW DEFAULT SHARDING STRATEGY

```
mysql> SHOW DEFAULT SHARDING STRATEGY ;

+-----+-----+-----+-----+
| name      | type      | sharding_column      | sharding_algorithm_name | sharding_
algorithm_type | sharding_algorithm_props |
+-----+-----+-----+-----+
| TABLE     | NONE      |                   |                   |           |
|          |           |                   |                   |           |
| DATABASE  | STANDARD | order_id          | database_inline      | INLINE
|          |           | {algorithm-expression=ds_${user_id % 2}} |
+-----+-----+-----+-----+
-----+
2 rows in set (0.07 sec)
```

SHOW SHARDING TABLE NODES

```
mysql> SHOW SHARDING TABLE NODES;
+-----+
| name      | nodes
+-----+
| t_order   | ds_0.t_order_0, ds_1.t_order_1, ds_0.t_order_2, ds_1.t_order_3 |
+-----+
1 row in set (0.02 sec)
```

Sharding Binding Table Rule

```
mysql> SHOW SHARDING BINDING TABLE RULES;
+-----+
| sharding_binding_tables |
+-----+
| t_order,t_order_item |
| t1,t2                 |
+-----+
2 rows in set (0.00 sec)
```

Sharding Broadcast Table Rule

```
mysql> SHOW SHARDING BROADCAST TABLE RULES;
+-----+
| sharding_broadcast_tables |
+-----+
| t_1                         |
| t_2                         |
+-----+
2 rows in set (0.00 sec)
```

Sharding Scaling Rule

```
mysql> SHOW SHARDING SCALING RULES;
+-----+
| name          | input           | output          | stream_channel |
| completion_detector | data_consistency_checker | | |
+-----+
| sharding_scaling | {"workerThread":40,"batchSize":1000} | {"workerThread":40, "batchSize":1000} | {"type":"MEMORY","props":{"block-queue-size":"10000"}} | {"type":"IDLE","props":{"incremental-task-idle-seconds-threshold":"1800"}} | {"type":"DATA_MATCH","props":{"chunk-size":"1000"}} |
+-----+
```

1 row in set (0.00 sec)

Single Table

Syntax

```
SHOW SINGLE TABLE (table | RULES) [FROM databaseName]
```

```
SHOW SINGLE TABLES
```

```
COUNT SINGLE_TABLE RULE [FROM databaseName]
```

table:

```
    TABLE tableName
```

Return Value Description

Single Table Rule

Column	Description
name	Rule name
resource_name	Data source name

Single Table

Column	Description
table_name	Single table name
resource_name	The resource name where the single table is located

Single Table Rule Count

列	说明
rule_name	Single table rule name
database	The database name where the single table is located
count	The count of single table rules

Example

SHOW SINGLE TABLES RULES

```
sql> SHOW SINGLE TABLES RULES;
+-----+
| name      | resource_name |
+-----+
| default   | ds_1           |
+-----+
1 row in set (0.01 sec)
```

SHOW SINGLE TABLE tableName

```
sql> SHOW SINGLE TABLE t_single_0;
+-----+
| table_name    | resource_name |
+-----+
| t_single_0    | ds_0           |
+-----+
1 row in set (0.01 sec)
```

SHOW SINGLE TABLES

```
mysql> SHOW SINGLE TABLES;
+-----+
| table_name    | resource_name |
+-----+
| t_single_0    | ds_0           |
| t_single_1    | ds_1           |
+-----+
2 rows in set (0.02 sec)
```

COUNT SINGLE_TABLE RULE

```
mysql> COUNT SINGLE_TABLE RULE;
+-----+
| rule_name    | database | count |
+-----+
| t_single_0   | ds       | 2      |
+-----+
1 row in set (0.02 sec)
```

Readwrite-Splitting

Syntax

```
SHOW READWRITE_SPLITTING RULES [FROM databaseName]
```

Return Value Description

Column	Description
name	Rule name
auto_aware_data_source_name	Auto-aware discovery data source name (Display configuration dynamic readwrite splitting rules)
write_data_source_query_enabled	All read data source are offline, write data source whether the data source is responsible for read traffic
write_data_source_name	Write data source name
read_data_source_names	Read data source name list
load_balancer_type	Load balance algorithm type
load_balancer_props	Load balance algorithm parameter

Example

Static Readwrite Splitting Rules

```
mysql> SHOW READWRITE_SPLITTING RULES;
+-----+-----+-----+-----+
| name      | auto_aware_data_source_name | write_data_source_name | read_data_
source_names | load_balancer_type | load_balancer_props |
+-----+-----+-----+-----+
| ms_group_0 |                         | ds_primary           | ds_slave_0,
ds_slave_1 | random                   |                   |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Dynamic Readwrite Splitting Rules

```
mysql> SHOW READWRITE_SPLITTING RULES FROM readwrite_splitting_db;
+-----+-----+-----+
| name      | auto_aware_data_source_name | write_data_source_query_enabled |
write_data_source_name | read_data_source_names | load_balancer_type | load_
balancer_props |
+-----+-----+-----+
```

```
+-----+-----+-----+
| readwrite_ds | ms_group_0 | random | read_weight=2:1
+-----+-----+-----+
| |
+-----+-----+-----+
1 row in set (0.01 sec)
```

Static Readwrite Splitting Rules And Dynamic Readwrite Splitting Rules

```
mysql> SHOW READWRITE_SPLITTING RULES FROM readwrite_splitting_db;
+-----+-----+-----+
| name | auto_aware_data_source_name | write_data_source_query_enabled |
| write_data_source_name | read_data_source_names | load_balancer_type | load_
balancer_props |
+-----+-----+-----+
| readwrite_ds | ms_group_0 | read_ds_0, read_ds_1 | random | read_
weight=2:1 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

DB Discovery

Syntax

```
SHOW DB_DISCOVERY RULES [FROM databaseName]

SHOW DB_DISCOVERY TYPES [FROM databaseName]

SHOW DB_DISCOVERY HEARTBEATS [FROM databaseName]
```

Return Value Description

DB Discovery Rule

Column	Description
group_name	Rule name
data_source_names	Data source name list
primary_data_source_name	Primary data source name
discovery_type	Database discovery service type
discovery_heartbeat	Database discovery service heartbeat

DB Discovery Type

Column	Description
name	Type name
type	Type category
props	Type properties

DB Discovery Heartbeat

Column	Description
name	Heartbeat name
props	Heartbeat properties

Example

DB Discovery Rule

```
mysql> SHOW DB_DISCOVERY RULES;
+-----+-----+-----+-----+
| group_name      | data_source_names | primary_data_source_name | discovery_type |
|-----+-----+-----+-----|
| db_discovery_group_0 | ds_0,ds_1,ds_2   |           ds_0           | {name=db_discovery_group_0_mgr, type=MySQL.MGR, props={group-name=92504d5b-6dec}} |
| {name=db_discovery_group_0_heartbeat, props={keep-alive-cron=0/5 * * * * ?}} |
+-----+-----+-----+-----+
```

```
1 row in set (0.20 sec)
```

DB Discovery Type

```
mysql> SHOW DB_DISCOVERY TYPES;
+-----+-----+-----+
| name          | type      | props           |
+-----+-----+-----+
| db_discovery_group_0_mgr | MySQL.MGR | {group-name=92504d5b-6dec} |
+-----+-----+-----+
1 row in set (0.01 sec)
```

DB Discovery Heartbeat

```
mysql> SHOW DB_DISCOVERY HEARTBEATS;
+-----+-----+
| name          | props           |
+-----+-----+
| db_discovery_group_0_heartbeat | {keep-alive-cron=0/5 * * * *} |
+-----+-----+
1 row in set (0.01 sec)
```

Encrypt

Syntax

```
SHOW ENCRYPT RULES [FROM databaseName]
```

```
SHOW ENCRYPT TABLE RULE tableName [FROM databaseName]
```

- Support to query all data encryption rules and specify logical table name query

Return Value Description

Column	Description
table	Logical table name
logic_column	Logical column name
logic_data_type	Logical column data type
cipher_column	Ciphertext column name
cipher_data_type	Ciphertext column data type
plain_column	Plaintext column name
plain_data_type	Plaintext column data type
assisted_query_column	Assisted query column name
assisted_query_data_type	Assisted query column data type
encryptor_type	Encryption algorithm type
encryptor_props	Encryption algorithm parameter
query_with_cipher_column	Whether to use encrypted column for query

Example

Show Encrypt Rules

```
mysql> SHOW ENCRYPT RULES FROM encrypt_db;
+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| table      | logic_column | logic_data_type | cipher_column | cipher_data_type |
| plain_column | plain_data_type | assisted_query_column | assisted_query_data_type |
| encryptor_type | encryptor_props |          | query_with_cipher_column |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| t_encrypt    | user_id      |           | user_cipher   |           |
| user_plain   |           |           |           |           |
| AES          | aes-key-value=123456abc | true       |           |           |
| t_encrypt    | order_id     |           | order_cipher |           |
|           |           |           |           |           |
| MD5          |           |           | true        |           |
| t_encrypt_2  | user_id      |           | user_cipher   |           |
| user_plain   |           |           |           |           |
| AES          | aes-key-value=123456abc | false      |           |           |
| t_encrypt_2  | order_id     |           | order_cipher |           |
|           |           |           |           |           |
| MD5          |           |           | false       |           |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```

```
4 rows in set (0.78 sec)
```

Show Encrypt Table Rule Table Name

```
mysql> SHOW ENCRYPT TABLE RULE t_encrypt;
+-----+-----+-----+-----+-----+
| table | logic_column | logic_data_type | cipher_column | cipher_data_type |
| plain_column | plain_data_type | assisted_query_column | assisted_query_data_type |
| encryptor_type | encryptor_props | query_with_cipher_column |
+-----+-----+-----+-----+
| t_encrypt | user_id | | user_cipher | |
| user_plain | | | | |
| AES | aes-key-value=123456abc | true | |
| t_encrypt | order_id | | order_cipher | |
| | | | | |
| MD5 | | | true | |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| 2 rows in set (0.01 sec)

mysql> SHOW ENCRYPT TABLE RULE t_encrypt FROM encrypt_db;
+-----+-----+-----+-----+-----+
| table | logic_column | logic_data_type | cipher_column | cipher_data_type |
| plain_column | plain_data_type | assisted_query_column | assisted_query_data_type |
| encryptor_type | encryptor_props | query_with_cipher_column |
+-----+-----+-----+-----+
| t_encrypt | user_id | | user_cipher | |
| user_plain | | | | |
| AES | aes-key-value=123456abc | true | |
| t_encrypt | order_id | | order_cipher | |
| | | | | |
| MD5 | | | true | |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| 2 rows in set (0.01 sec))
```

Shadow

Syntax

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

```
SHOW SHADOW TABLE RULES [FROM databaseName]
```

```
SHOW SHADOW ALGORITHMS [FROM databaseName]
```

shadowRule:

```
  RULE ruleName
```

- Support querying all shadow rules and specified table query
- Support querying all shadow table rules
- Support querying all shadow algorithms

Return Value Description

Shadow Rule

Column	Description
rule_name	Rule name
source_name	Source database
shadow_name	Shadow database
shadow_table	Shadow table

Shadow Table Rule

Column	Description
shadow_table	Shadow table
shadow_algorithm_name	Shadow algorithm name

Shadow Algorithms

Column	Description
shadow_algorithm_name	Shadow algorithm name
type	Shadow algorithm type
props	Shadow algorithm properties
is_default	Default

Shadow Rule status

Column	Description
status	Enable

Example

SHOW SHADOW RULES

```
mysql> SHOW SHADOW RULES;
+-----+-----+-----+-----+
| rule_name      | source_name | shadow_name | shadow_table |
+-----+-----+-----+-----+
| shadow_rule_1  | ds_1        | ds_shadow_1 | t_order      |
| shadow_rule_2  | ds_2        | ds_shadow_2 | t_order_item |
+-----+-----+-----+-----+
2 rows in set (0.02 sec)
```

SHOW SHADOW RULE ruleName

```
mysql> SHOW SHADOW RULE shadow_rule_1;
+-----+-----+-----+-----+
| rule_name      | source_name | shadow_name | shadow_table |
+-----+-----+-----+-----+
| shadow_rule_1  | ds_1        | ds_shadow_1 | t_order      |
+-----+-----+-----+-----+
1 rows in set (0.01 sec)
```

SHOW SHADOW TABLE RULES

```
mysql> SHOW SHADOW TABLE RULES;
+-----+-----+
| shadow_table | shadow_algorithm_name
|
+-----+-----+
| t_order_1    | user_id_match_algorithm,simple_hint_algorithm_1
|
+-----+-----+
1 rows in set (0.01 sec)
```

SHOW SHADOW ALGORITHMS

```
mysql> SHOW SHADOW ALGORITHMS;
+-----+-----+
```

```

| shadow_algorithm_name | type          | props
|   is_default         |               |
+-----+-----+
| user_id_match_algorithm | REGEX_MATCH | operation=insert,column=user_id,
|   regex=[1] | false      |
| simple_hint_algorithm_1 | SIMPLE_HINT | shadow=true,foo=bar
|   false           |
+-----+-----+
2 rows in set (0.01 sec)

```

RAL Syntax

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

Hint

Statement	Function	Example
SET READ WRITE_SPLITTING HINT SOURCE = [auto / write]	For current connection, set readwrite splitting routing strategy (automatic or forced to write data source)	SET READWRITE_SPLITTINGHINT SOURCE = write
SET SHARDING HINT DATABASE_VALUE = yy	For current connection, set sharding value for database sharding only, yy: sharding value	SET SHARDING HINT D ATABASE_VALUE = 100
ADD SHARDING HINT DATABASE_VALUE xx = yy	For current connection, add sharding value for table, xx: logic table, yy: database sharding value	ADD SHARDING HINT D ATABASE_VALUE t_order = 100
ADD SHARDING HINT TABLE_VALUE xx = yy	For current connection, add sharding value for table, xx: logic table, yy: table sharding value	ADD SHARDING HINT TABLE_VALUE t_order = 100
CLEAR HINT SETTINGS	For current connection, clear all hint settings	CLEAR HINT
CLEAR [SHARDING HINT / READ WRITE_SPLITTING HINT]	For current connection, clear hint settings of sharding or readwrite splitting	CLEAR READWR ITE_SPLITTING HINT
SHOW [SHARDING / READ WRITE_SPLITTING] HINT STATUS	For current connection, query hint settings of sharding or readwrite splitting	SHOW READWR ITE_SPLITTING HINT STATUS

Scaling

Statement	Function	Example
SHOW SCALING LIST	Query running list	SHOW SCALING LIST
SHOW SCALING STATUS jobId	Query scaling status, xx: jobId	SHOW SCALING STATUS 1234
START SCALING jobId	Start scaling, xx: jobId	START SCALING 1234
STOP SCALING jobId	Stop scaling, xx: jobId	STOP SCALING 1234
DROP SCALING jobId	Drop scaling, xx: jobId	DROP SCALING 1234
RESET SCALING jobId	reset progress, xx: jobId	RESET SCALING 1234
CHECK SCALING jobId	Data consistency check with algorithm in server.yaml, xx: jobId	CHECK SCALING 1234
SHOW SCALING CHECK ALGORITHMS	Show available consistency check algorithms	SHOW SCALING CHECK ALGORITHMS
CHECK SCALING {jobId} by type{name={algorithmType}}	Data consistency check with defined algorithm	CHECK SCALING 1234 by type(name=DEFAULT)
STOP SCALING SOURCE WRITING jobId	The source ShardingSphere data source is discontinued, xx: jobId	STOP SCALING SOURCE WRITING 1234
RESTORE SCALING SOURCE WRITING jobId	Restore source data source writing, xx: jobId	RESTORE SCALING SOURCE WRITING 1234
APPLY SCALING jobId	Switch to target ShardingSphere metadata, xx: jobId	APPLY SCALING 1234

Circuit Breaker

Statement	Function	Example
[ENABLE / DISABLE] READWRITE_SPLITTING (READ)? resourceName [FROM databaseName]	Enable or disable read data source	ENABLE READWRITE_SPLITTING READ resource_0
[ENABLE / DISABLE] INSTANCE instanceId	Enable or disable proxy instance	DISABLE INSTANCE instance_1
SHOW INSTANCE LIST	Query proxy instance information	SHOW INSTANCE LIST
SHOW READWRITE_SPLITTING (READ)? resourceName [FROM databaseName]	Query all read resources status	SHOW READWRITE_SPLITTING READ RESOURCES

Global Rule

Statement	Function	Example
SHOW AUTHORITY RULE	Query authority rule configuration	SHOW AUTHORITY RULE
SHOW TRANSACTION RULE	Query transaction rule configuration	SHOW TRANSACTION RULE
SHOW SQL_PARSER RULE	Query SQL parser rule configuration	SHOW SQL_PARSER RULE
ALTER TRANSACTION RULE(DEFAULT:LT=xx,TYPE(NAME=xxx,PROPER TIES("key1" = "value1", "key2" = "value2" ..)))	Alter transaction rule configuration, DEFAULT: default transaction type, support LOCAL, XA, BASE; NAME: name of transaction manager, support Atomikos, Narayana and Bitronix	ALTER TRANSACTION RULE(DEFAULT=XA,TYPE(NAME=Narayana, PROPERTIES("databaseName" = "jbossts" , "host" = "127.0.0.1")))
ALTER SQL_PARSER RULE SQL_COMMENT_PARSE_ENABLE=xx,PARSE_TREE_CACHE(INITIAL_CAPACITY=xx, MAXIMUM_SIZE=xx, CURRENCY_LEVEL=xx), SQL_STATEMENT_CACHE(INITIAL_CAPACITY=xxx, MAXIMUM_SIZE=xxx, CURRENCY_LEVEL=xxx)	Alter SQL parser rule configuration, SQL_COMMENT_PARSE_ENABLE: whether to parse the SQL comment, PARSE_TREE_CACHE: local cache configuration of syntax tree, SQL_STATEMENT_CACHE: local cache of SQL statement	ALTER SQL_PARSER RULE SQL_COMMENT_PARSE_ENABLE=false,PARSE_TREE_CACHE(INITIAL_CAPACITY=10, MAXIMUM_SIZE=11, CURRENCY_LEVEL=1),SQL_STATEMENT_CACHE(INITIAL_CAPACITY=11, MAXIMUM_SIZE=11, CURRENCY_LEVEL=100)

Other

Statement	Function	Example
SHOW INSTANCE MODE	Query the mode configuration of the proxy	SHOW INSTANCE MODE
COUNT DATABASE RULES [FROM database]	Query the number of rules in a database	COUNT DATABASE RULES
SET VARIABLE pro xy_property_name = xx	proxy_property_name is one of properties configuration of proxy, name is split by underscore	SET VARIABLE sql_show = true
SET VARIABLE transaction_type = xx	Modify transaction_type of the current connection, supports LOCAL, XA, BASE	SET VARIABLE transaction_type = XA
SET VARIABLE agent_plugins_enabled = [TRUE / FALSE]	Set whether the agent plugins are enabled, the default value is false	SET VARIABLE agent_plugins_enabled = TRUE
SHOW ALL VARIABLES	Query proxy all properties configuration	SHOW ALL VARIABLES
SHOW VARIABLE variable_name	Query proxy variable, name is split by underscore	SHOW VARIABLE sql_show
PREVIEW SQL	Preview the actual SQLs	PREVIEW SELECT * FROM t_order
PARSE SQL	Parse SQL and output abstract syntax tree	PARSE SELECT * FROM t_order
REFRESH TABLE METADATA	Refresh the metadata of all tables	REFRESH TABLE METADATA
REFRESH TABLE METADATA [tableName / tableName FROM RESOURCE resourceName]	Refresh the metadata of a table	REFRESH TABLE METADATA t_order FROM resource ds_1
SHOW TABLE METADATA tableName [, tableName] ...	Query table metadata	SHOW TABLE METADATA t_order
EXPORT DATABASE CONFIG [FROM database_name] [, file=“file_path”]	Query / export resources and rule configuration in database	EXPORT DATABASE CONFIG FROM re adwrite_sp litting_db
SHOW RULES USED RESOURCE resourceName [from database]	Query the rules for using the specified resource in database	SHOW RULES USED RESOURCE ds_0 FROM da tabaseName

Notice

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

Usage

This chapter will introduce how to use DistSQL to manage resources and rules in a distributed database.

Pre-work

Use MySQL as example, can replace to other databases.

1. Start the MySQL service;
2. Create to be registered MySQL databases;
3. Create role and user in MySQL with creation permission for ShardingSphere-Proxy;
4. Start Zookeeper service;
5. Add mode and authentication configurations to server.yaml;
6. Start ShardingSphere-Proxy;
7. Use SDK or terminal connect to ShardingSphere-Proxy.

Create Logic Database

1. Create logic database

```
CREATE DATABASE foo_db;
```

2. Use newly created logic database

```
USE foo_db;
```

Resource Operation

More details please see concentrate rule examples.

Rule Operation

More details please see concentrate rule examples.

Notice

1. Currently, `DROP DATABASE` will only remove the logical distributed database, not the user's actual database;
2. `DROP TABLE` will delete all logical fragmented tables and actual tables in the database;
3. `CREATE DATABASE` will only create a logical distributed database, so users need to create actual databases in advance.

Sharding

Resource Operation

- Configure data source information

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

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

Rule Operation

- Create sharding rule

```
CREATE SHARDING TABLE RULE t_order(
RESOURCES(ds_0,ds_1),
SHARDING_COLUMN=order_id,
TYPE(NAME=hash_mod,PROPERTIES("sharding-count"=4)),
KEY_GENERATE_STRATEGY(COLUMN=order_id,TYPE(NAME=snowflake))
);
```

- Create sharding table

```
CREATE TABLE `t_order` (
    `order_id` int NOT NULL,
    `user_id` int NOT NULL,
    `status` varchar(45) DEFAULT NULL,
    PRIMARY KEY (`order_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

- Drop sharding table

```
DROP TABLE t_order;
```

- Drop sharding rule

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

- Drop resource

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

- Drop distributed database

```
DROP DATABASE foo_db;
```

Readwrite_splitting

Resource Operation

```
ADD RESOURCE write_ds (
HOST=127.0.0.1,
PORT=3306,
DB=ds_0,
USER=root,
PASSWORD=root
),read_ds (
HOST=127.0.0.1,
PORT=3307,
DB=ds_0,
USER=root,
PASSWORD=root
);
```

Rule Operation

- Create readwrite_splitting rule

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

- Alter readwrite_splitting rule

```
ALTER READWRITE_SPLITTING RULE group_0 (
    WRITE_RESOURCE=write_ds,
    READ_RESOURCES(read_ds),
    TYPE(NAME=random, PROPERTIES(read_weight='2:0'))
);
```

- Drop readwrite_splitting rule

```
DROP READWRITE_SPLITTING RULE group_0;
```

- Drop resource

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

- Drop distributed database

```
DROP DATABASE readwrite_splitting_db;
```

Encrypt

Resource Operation

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

Rule Operation

- Create encrypt rule

```
CREATE ENCRYPT RULE t_encrypt (
    COLUMNS(
        (NAME=user_id,PLAIN=user_plain,CIPHER=user_cipher,TYPE(NAME=AES,PROPERTIES(
        'aes-key-value'='123456abc'))),
        (NAME=order_id,PLAIN=order_plain,CIPHER=order_cipher,TYPE(NAME=RC4,
        PROPERTIES('rc4-key-value'='123456abc'))))
);
```

- Create encrypt table

```
CREATE TABLE `t_encrypt` (
    `id` int(11) NOT NULL,
    `user_id` varchar(45) DEFAULT NULL,
    `order_id` varchar(45) DEFAULT NULL,
    PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

- Alter encrypt rule

```
ALTER ENCRYPT RULE t_encrypt (
    COLUMNS(
        (NAME=user_id,PLAIN=user_plain,CIPHER=user_cipher,TYPE(NAME=AES,PROPERTIES(
        'aes-key-value'='123456abc'))))
);
```

- Drop encrypt rule

```
DROP ENCRYPT RULE t_encrypt;
```

- Drop resource

```
DROP RESOURCE ds_0;
```

- Drop distributed database

```
DROP DATABASE encrypt_db;
```

DB Discovery

Resource Operation

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

Rule Operation

- Create DB discovery rule

```
CREATE DB_DISCOVERY RULE db_discovery_group_0 (
RESOURCES(ds_0, ds_1),
TYPE(NAME=MySQL.MGR,PROPERTIES('group-name'='92504d5b-6dec')),
HEARTBEAT(PROPERTIES('keep-alive-cron'='0/5 * * * * ?'))
);
```

- Alter DB discovery rule

```
ALTER DB_DISCOVERY RULE db_discovery_group_0 (
RESOURCES(ds_0, ds_1, ds_2),
TYPE(NAME=MySQL.MGR,PROPERTIES('group-name'='92504d5b-6dec')),
HEARTBEAT(PROPERTIES('keep-alive-cron'='0/5 * * * * ?'))
);
```

- Drop db_discovery rule

```
DROP DB_DISCOVERY RULE db_discovery_group_0;
```

- Drop db_discovery type

```
DROP DB_DISCOVERY TYPE db_discovery_group_0_mgr;
```

- Drop db_discovery heartbeat

```
DROP DB_DISCOVERY HEARTBEAT db_discovery_group_0_heartbeat;
```

- Drop resource

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

- Drop distributed database

```
DROP DATABASE discovery_db;
```

Shadow

Resource Operation

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

Rule Operation

- Create shadow rule

```
CREATE SHADOW RULE group_0(
SOURCE=ds_0,
SHADOW=ds_1,
t_order((simple_hint_algorithm, TYPE(NAME=SIMPLE_HINT, PROPERTIES("foo"="bar"))),
(TYPE(NAME=REGEX_MATCH, PROPERTIES("operation"="insert","column"="user_id", "regex"
"='[1]' ))),
t_order_item((TYPE(NAME=SIMPLE_HINT, PROPERTIES("foo"="bar")))));
```

- Alter shadow rule

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

- Drop shadow rule

```
DROP SHADOW RULE group_0;
```

- Drop resource

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

- Drop distributed database

```
DROP DATABASE foo_db;
```

7.2.4 Logging

This chapter will introduce the detailed syntax of Logging which is used when users need to distinguish databases or users in the log. To achieve a specific goal, following configurations can be added to logback.xml:

To distinguish databases in the same log

```
<appender name="databaseConsole" class="ch.qos.logback.core.ConsoleAppender">
<encoder>
    <pattern>[%-5level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] [%X{database}]
%logger{36} - %msg%n</pattern>
</encoder>
</appender>

<logger name="ShardingSphere-SQL" level="info" additivity="false">
```

```

<appender-ref ref="databaseConsole" />
</logger>

```

To distinguish databases and users in the same log

```

<appender name="databaseConsole" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
        <pattern>[%-5level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] [%X{database}] [%X{user}] %logger{36} - %msg%n</pattern>
    </encoder>
</appender>

<logger name="ShardingSphere-SQL" level="info" additivity="false">
    <appender-ref ref="databaseConsole" />
</logger>

```

To split into different log files

```

<appender name="SiftingFile" class="ch.qos.logback.classic.sift.SiftingAppender">
    <discriminator>
        <key>database</key>
        <defaultValue>none</defaultValue>
    </discriminator>
    <sift>
        <appender name="File-${database}" class="ch.qos.logback.core.FileAppender">
            <file>logs/${database}.log</file>
            <append>true</append>
            <encoder charset="UTF-8">
                <pattern>[%-5level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] [%X{user}] [%logger{36} - %msg%n</pattern>
            </encoder>
        </appender>
    </sift>
</appender>

<logger name="ShardingSphere-SQL" level="info" additivity="false">
    <appender-ref ref="SiftingFile" />
</logger>

```

7.2.5 Scaling

Introduction

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

Build

Build&Deployment

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

```
git clone --depth 1 https://github.com/apache/shardingsphere.git
cd shardingsphere
mvn clean install -Dmaven.javadoc.skip=true -Dcheckstyle.skip=true -Drat.skip=true
-Djacoco.skip=true -DskipITs -DskipTests -Prelease
```

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

Or get binary package from [download page](#).

Scaling is an experimental feature, if scaling job fail, you could try nightly version, click here to [download nightly build](#).

2. Unzip the proxy distribution package, modify the configuration file `conf/config-sharding.yaml`. Please refer to [proxy startup manual](#) for more details.
3. Modify the configuration file `conf/server.yaml`. Please refer to [Mode Configuration](#) for more details. Type of mode must be `Cluster` for now, please start the registry center before running proxy.

Configuration Example:

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

4. Enable scaling

Way 1. Modify scalingName and scaling configuration in conf/config-sharding.yaml.

Configuration Items Explanation:

```

rules:
- !SHARDING
  # ignored configuration

  scalingName: # Enabled scaling action config name
  scaling:
    <scaling-action-config-name> (+):
      input: # Data read configuration. If it's not configured, then part of its
configuration will take effect.
        workerThread: # Worker thread pool size for inventory data ingestion from
source. If it's not configured, then use system default value.
        batchSize: # Maximum records count of a DML select operation. If it's not
configured, then use system default value.
        rateLimiter: # Rate limit algorithm. If it's not configured, then system
will skip rate limit.
        type: # Algorithm type. Options:
        props: # Algorithm properties
      output: # Data write configuration. If it's not configured, then part of its
configuration will take effect.
        workerThread: # Worker thread pool size for data importing to target. If it
's not configured, then use system default value.
        batchSize: # Maximum records count of a DML insert/delete/update operation.
If it's not configured, then use system default value.
        rateLimiter: # Rate limit algorithm. If it's not configured, then system
will skip rate limit.
        type: # Algorithm type. Options:
        props: # Algorithm properties
      streamChannel: # Algorithm of channel that connect producer and consumer,
used for input and output. If it's not configured, then system will use MEMORY type
        type: # Algorithm type. Options: MEMORY
        props: # Algorithm properties
        block-queue-size: # Property: data channel block queue size. Available
for types: MEMORY
      completionDetector: # Completion detect algorithm. If it's not configured,
then system won't continue to do next steps automatically.
        type: # Algorithm type. Options: IDLE
        props: # Algorithm properties
        incremental-task-idle-seconds-threshold: # If incremental tasks is idle
more than so much seconds, then it could be considered as almost completed.
Available for types: IDLE
      dataConsistencyChecker: # Data consistency check algorithm. If it's not
configured, then system will skip this step.
        type: # Algorithm type. Options: DATA_MATCH, CRC32_MATCH
        props: # Algorithm properties
        chunk-size: # Maximum records count of a query operation for check

```

type of dataConsistencyChecker could be got by executing DistSQL SHOW SCALING CHECK ALGORITHMS. Simple comparison: - DATA_MATCH : Support all types of databases, but it's not the best performant one. - CRC32_MATCH : Support MySQL, performance is better than DATA_MATCH.

Auto Mode Configuration Example:

```
rules:
- !SHARDING
  # ignored configuration

  scalingName: scaling_auto
  scaling:
    scaling_auto:
      input:
        workerThread: 40
        batchSize: 1000
      output:
        workerThread: 40
        batchSize: 1000
      streamChannel:
        type: MEMORY
      props:
        block-queue-size: 10000
    completionDetector:
      type: IDLE
      props:
        incremental-task-idle-seconds-threshold: 1800
  dataConsistencyChecker:
    type: DATA_MATCH
    props:
      chunk-size: 1000
```

Manual Mode Configuration Example:

```
rules:
- !SHARDING
  # ignored configuration

  scalingName: scaling_manual
  scaling:
    scaling_manual:
      input:
        workerThread: 40
        batchSize: 1000
      output:
        workerThread: 40
        batchSize: 1000
      streamChannel:
        type: MEMORY
```

```

props:
  block-queue-size: 10000
dataConsistencyChecker:
  type: DATA_MATCH
  props:
    chunk-size: 1000

```

Way 2: Configure scaling by DistSQL

Auto Mode Configuration Example:

```

CREATE SHARDING SCALING RULE scaling_auto (
INPUT(
  WORKER_THREAD=40,
  BATCH_SIZE=1000
),
OUTPUT(
  WORKER_THREAD=40,
  BATCH_SIZE=1000
),
STREAM_CHANNEL(TYPE(NAME=MEMORY, PROPERTIES("block-queue-size"=10000))),
COMPLETION_DETECTOR(TYPE(NAME=IDLE, PROPERTIES("incremental-task-idle-seconds-threshold"=1800))),
DATA_CONSISTENCY_CHECKER(TYPE(NAME=DATA_MATCH, PROPERTIES("chunk-size"=1000)))
);

```

Manual Mode Configuration Example:

```

CREATE SHARDING SCALING RULE scaling_manual (
INPUT(
  WORKER_THREAD=40,
  BATCH_SIZE=1000
),
OUTPUT(
  WORKER_THREAD=40,
  BATCH_SIZE=1000
),
STREAM_CHANNEL(TYPE(NAME=MEMORY, PROPERTIES("block-queue-size"=10000))),
DATA_CONSISTENCY_CHECKER(TYPE(NAME=DATA_MATCH, PROPERTIES("chunk-size"=1000)))
);

```

Please refer to [RDL#Sharding](#) for more details.

5. Import JDBC driver dependency

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

RDBMS	JDBC driver	Reference
MySQL	mysql-connector-java-5.1. 47.jar	Connector/J Versions

6. Start up ShardingSphere-Proxy:

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

7. Check proxy log logs/stdout.log:

```
[INFO ] [main] o.a.s.p.frontend.ShardingSphereProxy - ShardingSphere-Proxy start success
```

It means proxy start up successfully.

Shutdown

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

Manual

Manual

Environment

JAVA, JDK 1.8+.

The migration scene we support:

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

Supported features:

Feature	MySQL	PostgreSQL	openGauss
Inventory migration	Supported	Supported	Supported
Incremental migration	Supported	Supported	Supported
Create table automatically	Supported	Supported	Supported
DATA_MATCH data consistency check	Supported	Supported	Supported
CRC32_MATCH data consistency check	Supported	Unsupported	Unsupported

Attention:

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

Privileges

MySQL

1. Enable binlog

Configuration Example of MySQL 5.7 my.cnf:

```
[mysqld]
server-id=1
log-bin=mysql-bin
binlog-format=row
binlog-row-image=full
max_connections=600
```

Execute the following SQL to confirm whether binlog is turned on or not:

```
show variables like '%log_bin%';
show variables like '%binlog%';
```

As shown below, it means binlog has been turned on:

Variable_name	Value
log_bin	ON
binlog_format	ROW
binlog_row_image	FULL

2. Privileges of account that scaling use should include Replication privileges.

Execute the following SQL to confirm whether the user has migration permission or not:

```
SHOW GRANTS FOR 'user';
```

Result Example:

Grants for \${username}@\${host}
GRANT REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO \${username}@\${host}
.....

PostgreSQL

1. Enable `test_decoding` feature.
2. Adjust WAL configuration

Configuration Example of `postgresql.conf`:

```
wal_level = logical
max_replication_slots = 10
max_connections = 600
```

Please refer to [Write Ahead Log](#) and [Replication](#) for more details.

DistSQL API for auto mode

Preview current sharding rule

Example:

```
preview SELECT COUNT(1) FROM t_order;
```

Response:

```
mysql> preview SELECT COUNT(1) FROM t_order;
+-----+
| data_source_name | actual_sql
|-----+
| ds_0             | SELECT COUNT(1) FROM t_order_0 UNION ALL SELECT COUNT(1) FROM
t_order_1 |
| ds_1             | SELECT COUNT(1) FROM t_order_0 UNION ALL SELECT COUNT(1) FROM
t_order_1 |
+-----+
2 rows in set (0.65 sec)
```

Start scaling job

1. Add new data source resources

Please refer to [RDL#Data Source](#) for more details.

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

Example:

```

ADD RESOURCE ds_2 (
    URL="jdbc:mysql://127.0.0.1:3306/scaling_ds_2?serverTimezone=UTC&useSSL=false",
    USER=root,
    PASSWORD=root,
    PROPERTIES("maximumPoolSize"=10,"idleTimeout"="30000")
), ds_3 (
    URL="jdbc:mysql://127.0.0.1:3306/scaling_ds_3?serverTimezone=UTC&useSSL=false",
    USER=root,
    PASSWORD=root,
    PROPERTIES("maximumPoolSize"=10,"idleTimeout"="30000")
), ds_4 (
    URL="jdbc:mysql://127.0.0.1:3306/scaling_ds_4?serverTimezone=UTC&useSSL=false",
    USER=root,
    PASSWORD=root,
    PROPERTIES("maximumPoolSize"=10,"idleTimeout"="30000")
);

```

2. Alter sharding table rule for tables to be scaled

We could scale all tables or partial tables. Binding tables must be scaled together.

Currently, scaling job could only be emitted by executing `ALTER SHARDING TABLE RULE` DistSQL.

Please refer to [RDL#Sharding](#) for more details.

`SHARDING TABLE RULE` support two types: `TableRule` and `AutoTableRule`. Following is a comparison of the two sharding rule types:

Type	Au-toTableRule	TableRule
Def-initiation	Auto Sharding Alg orithm	`User-Defined Sharding Algorithm < https://shardingsphere.apache.org/document/current/en/features/sharding/concept/sharding/#user-defined-sharding-algorithm >` __

Meaning of fields in DistSQL is the same as YAML configuration, please refer to [YAML Configuration#Sharding](#) for more details.

Example of alter AutoTableRule:

```

ALTER SHARDING TABLE RULE t_order (
RESOURCES(ds_2, ds_3, ds_4),
SHARDING_COLUMN=order_id,
TYPE(NAME=hash_mod,PROPERTIES("sharding-count"=6)),
KEY_GENERATE_STRATEGY(COLUMN=order_id,TYPE(NAME=snowflake))
);

```

`RESOURCES` is altered from `(ds_0, ds_1)` to `(ds_2, ds_3, ds_4)`, and `sharding-count` is altered from 4 to 6, it will emit scaling job.

Uncompleted example of alter TableRule:

```

ALTER SHARDING ALGORITHM database_inline (
    TYPE(NAME=INLINE, PROPERTIES("algorithm-expression"="ds_${user_id % 3 + 2}"))
);

ALTER SHARDING TABLE RULE t_order (
    DATANODES("ds_${2..4}.t_order_${0..1}"),
    DATABASE_STRATEGY(TYPE=standard, SHARDING_COLUMN=user_id, SHARDING_
    ALGORITHM=database_inline),
    TABLE_STRATEGY(TYPE=standard, SHARDING_COLUMN=order_id, SHARDING_ALGORITHM=t_order_
    inline),
    KEY_GENERATE_STRATEGY(COLUMN=order_id, TYPE(NAME=snowflake))
), t_order_item (
    DATANODES("ds_${2..4}.t_order_item_${0..1}"),
    DATABASE_STRATEGY(TYPE=standard, SHARDING_COLUMN=user_id, SHARDING_
    ALGORITHM=database_inline),
    TABLE_STRATEGY(TYPE=standard, SHARDING_COLUMN=order_id, SHARDING_ALGORITHM=t_order_
    item_inline),
    KEY_GENERATE_STRATEGY(COLUMN=order_item_id, TYPE(NAME=snowflake))
);

```

algorithm-expression of database_inline is alerted from `ds_${user_id % 2}` to `ds_${user_id % 3 + 2}`, and DATANODES of t_order is alerted from `ds_${0..1}`. `t_order_${0..1}` to `ds_${2..4}.t_order_${0..1}`, it will emit scaling job.

Currently, ALTER SHARDING ALGORITHM will take effect immediately, but table rule will not, it might cause inserting data into source side failure, so alter sharding table rule to AutoTableRule is recommended for now.

List scaling jobs

Please refer to [RAL#Scaling](#) for more details.

Example:

```
show scaling list;
```

Response:

```

mysql> show scaling list;
+-----+-----+-----+-----+
| id      | tables          | sharding_total_count | active |
| create_time | stop_time       |                   |
+-----+-----+-----+-----+
| 659853312085983232 | t_order_item, t_order | 2                  | false  |
| 2021-10-26 20:21:31 | 2021-10-26 20:24:01 |
| 660152090995195904 | t_order_item, t_order | 2                  | false  |
| 2021-10-27 16:08:43 | 2021-10-27 16:11:00 |

```

```
+-----+-----+-----+-----+
-----+
2 rows in set (0.04 sec)
```

Get scaling progress

Example:

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

Response:

```
mysql> show scaling status 660152090995195904;
+-----+-----+-----+-----+
-----+
| item | data_source | status      | inventory_finished_percentage | incremental_idle_seconds |
+-----+-----+-----+-----+
-----+
| 0    | ds_1        | FINISHED   | 100                      | 2834
|
| 1    | ds_0        | FINISHED   | 100                      | 2834
|
+-----+-----+-----+-----+
-----+
2 rows in set (0.00 sec)
```

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

status values:

Value	Description
PREPARING	preparing
RUNNING	running
EXECUTE_INVENTORY_TASK	inventory task running
EXE-CUTE_INCREMENTAL_TASK	incremental task running
FINISHED	finished (The whole process is completed, and the new rules have been taken effect)
PREPARING_FAILURE	preparation failed
E-XE-CUTE_INVENTORY_TASK_FAILURE	inventory task failed
EXE-CUTE_INCREMENTAL_TASK_FAILURE	incremental task failed

If status fails, you can check the log of proxy to view the error stack and analyze the problem.

Preview new sharding rule

Example:

```
preview SELECT COUNT(1) FROM t_order;
```

Response:

```
mysql> PREVIEW SELECT COUNT(1) FROM t_order;
+-----+
-----+
| data_source_name | actual_sql
|                 |
+-----+-----+
| ds_2           | SELECT COUNT(1) FROM t_order_0 UNION ALL SELECT COUNT(1) FROM
t_order_1 |
| ds_3           | SELECT COUNT(1) FROM t_order_0 UNION ALL SELECT COUNT(1) FROM
t_order_1 |
| ds_4           | SELECT COUNT(1) FROM t_order_0 UNION ALL SELECT COUNT(1) FROM
t_order_1 |
+-----+-----+
-----+
3 rows in set (0.21 sec)
```

Other DistSQL

Please refer to [RAL#Scaling](#) for more details.

DistSQL manual mode whole process example

On manual mode, data consistency check and switch configuration could be emitted manually. Please refer to [RAL#Scaling](#) for more details.

This example show how to migrate data from MySQL to proxy.

Most SQLs should be executed in proxy, except few ones mentioned for MySQL.

Create source databases

It's not needed in practice. It just simulates databases for testing.

Execute SQLs in MySQL:

```
DROP DATABASE IF EXISTS scaling_ds_0;
CREATE DATABASE scaling_ds_0 DEFAULT CHARSET utf8;
```

```
DROP DATABASE IF EXISTS scaling_ds_1;
CREATE DATABASE scaling_ds_1 DEFAULT CHARSET utf8;
```

Login proxy

```
mysql -h127.0.0.1 -P3307 -uroot -proot
```

Create and configure logical database

Create logical database:

```
CREATE DATABASE scaling_db;

USE scaling_db
```

Add source database resource:

```
ADD RESOURCE ds_0 (
    URL="jdbc:mysql://127.0.0.1:3306/scaling_ds_0?serverTimezone=UTC&useSSL=false",
    USER=root,
    PASSWORD=root,
    PROPERTIES("maximumPoolSize"=50,"idleTimeout"="60000")
), ds_1 (
    URL="jdbc:mysql://127.0.0.1:3306/scaling_ds_1?serverTimezone=UTC&useSSL=false",
    USER=root,
    PASSWORD=root,
    PROPERTIES("maximumPoolSize"=50,"idleTimeout"="60000")
);
```

Configure rules: Configure tables of existing system in sharding rule, sharding table rules and INLINE algorithm will be used to fit existing tables name.

```
CREATE SHARDING ALGORITHM database_inline (
TYPE(NAME=INLINE,PROPERTIES("algorithm-expression"="ds_${user_id % 2}"))
);
CREATE SHARDING ALGORITHM t_order_inline (
TYPE(NAME=INLINE,PROPERTIES("algorithm-expression"="t_order_${order_id % 2}"))
);
CREATE SHARDING ALGORITHM t_order_item_inline (
TYPE(NAME=INLINE,PROPERTIES("algorithm-expression"="t_order_item_${order_id % 2}"))
);

CREATE SHARDING TABLE RULE t_order (
DATANODES("ds_${0..1}.t_order_${0..1}"),
DATABASE_STRATEGY(TYPE=standard,SHARDING_COLUMN=user_id,SHARDING_ALGORITHM=database_inline),
```

```

TABLE_STRATEGY(TYPE=standard,SHARDING_COLUMN=order_id,SHARDING_ALGORITHM=t_order_inline),
KEY_GENERATE_STRATEGY(COLUMN=order_id,TYPE(NAME=snowflake))
), t_order_item (
DATANODES("ds_${0..1}.t_order_item_${0..1}"),
DATABASE_STRATEGY(TYPE=standard,SHARDING_COLUMN=user_id,SHARDING_ALGORITHM=database_inline),
TABLE_STRATEGY(TYPE=standard,SHARDING_COLUMN=order_id,SHARDING_ALGORITHM=t_order_item_inline),
KEY_GENERATE_STRATEGY(COLUMN=order_item_id,TYPE(NAME=snowflake))
);

CREATE SHARDING SCALING RULE scaling_manual2 (
DATA_CONSISTENCY_CHECKER(TYPE(NAME=CRC32_MATCH))
);

```

Create test tables and initialize records

It's not needed in practice.

```

CREATE TABLE t_order (order_id INT NOT NULL, user_id INT NOT NULL, status
VARCHAR(45) CHARSET utf8mb4, PRIMARY KEY (order_id));
CREATE TABLE t_order_item (item_id INT NOT NULL, order_id INT NOT NULL, user_id INT
NOT NULL, status VARCHAR(45) CHARSET utf8mb4, creation_date DATE, PRIMARY KEY
(item_id));

INSERT INTO T_ORDER (order_id, user_id, status) VALUES (1,2,'ok'),(2,4,'ok'),(3,6,
'ok'),(4,1,'ok'),(5,3,'ok'),(6,5,'ok');
INSERT INTO T_ORDER_ITEM (item_id, order_id, user_id, status) VALUES (1,1,2,'ok'),
(2,2,4,'ok'),(3,3,6,'ok'),(4,4,1,'ok'),(5,5,3,'ok'),(6,6,5,'ok');

```

Run migration

Preview sharding:

```

mysql> PREVIEW SELECT COUNT(1) FROM t_order;
+-----+
| data_source_name | actual_sql
|-----|
+-----+
| ds_0           | SELECT COUNT(1) FROM t_order_0 UNION ALL SELECT COUNT(1) FROM
t_order_1 |
| ds_1           | SELECT COUNT(1) FROM t_order_0 UNION ALL SELECT COUNT(1) FROM
t_order_1 |

```

```
+-----+
-----+
2 rows in set (0.65 sec)
```

Create target databases in MySQL:

```
DROP DATABASE IF EXISTS scaling_ds_10;
CREATE DATABASE scaling_ds_10 DEFAULT CHARSET utf8;

DROP DATABASE IF EXISTS scaling_ds_11;
CREATE DATABASE scaling_ds_11 DEFAULT CHARSET utf8;

DROP DATABASE IF EXISTS scaling_ds_12;
CREATE DATABASE scaling_ds_12 DEFAULT CHARSET utf8;
```

Add target database resource:

```
ADD RESOURCE ds_2 (
    URL="jdbc:mysql://127.0.0.1:3306/scaling_ds_10?serverTimezone=UTC&useSSL=false",
    USER=root,
    PASSWORD=root,
    PROPERTIES("maximumPoolSize"=50,"idleTimeout"="60000")
), ds_3 (
    URL="jdbc:mysql://127.0.0.1:3306/scaling_ds_11?serverTimezone=UTC&useSSL=false",
    USER=root,
    PASSWORD=root,
    PROPERTIES("maximumPoolSize"=50,"idleTimeout"="60000")
), ds_4 (
    URL="jdbc:mysql://127.0.0.1:3306/scaling_ds_12?serverTimezone=UTC&useSSL=false",
    USER=root,
    PASSWORD=root,
    PROPERTIES("maximumPoolSize"=50,"idleTimeout"="60000")
);
```

Alter sharding rule to emit scaling job:

```
ALTER SHARDING ALGORITHM database_inline (
TYPE(NAME=INLINE,PROPERTIES("algorithm-expression"="ds_${user_id % 3 + 2}"))
);

ALTER SHARDING TABLE RULE t_order (
DATANODES("ds_${2..4}.t_order_${0..1}"),
DATABASE_STRATEGY(TYPE=standard,SHARDING_COLUMN=user_id,SHARDING_ALGORITHM=database_inline),
TABLE_STRATEGY(TYPE=standard,SHARDING_COLUMN=order_id,SHARDING_ALGORITHM=t_order_inline),
```

```

KEY_GENERATE_STRATEGY(COLUMN=order_id,TYPE(NAME=snowflake))
), t_order_item (
DATANODES("ds_${2..4}.t_order_item_${0..1}"),
DATABASE_STRATEGY(TYPE=standard,SHARDING_COLUMN=user_id,SHARDING_
ALGORITHM=database_inline),
TABLE_STRATEGY(TYPE=standard,SHARDING_COLUMN=order_id,SHARDING_ALGORITHM=t_order_
item_inline),
KEY_GENERATE_STRATEGY(COLUMN=order_item_id,TYPE(NAME=snowflake))
);

```

Query job progress:

```

mysql> SHOW SCALING LIST;
+-----+-----+-----+
| id          | tables      | sharding_ |
| total_count | active     | stop_time |
+-----+-----+-----+
| 0130317c30317c3054317c7363616c696e675f6462 | t_order,t_order_item | 2
| true       | 2022-04-16 17:22:19 | NULL       |
+-----+-----+-----+
1 row in set (0.34 sec)

mysql> SHOW SCALING STATUS 0130317c30317c3054317c7363616c696e675f6462;
+-----+-----+-----+
| item | data_source | status           | active | inventory_finished_
percentage | incremental_idle_seconds |
+-----+-----+-----+
| 0    | ds_0        | EXECUTE_INCREMENTAL_TASK | true   | 100
| 8    |             |                           |         |
| 1    | ds_1        | EXECUTE_INCREMENTAL_TASK | true   | 100
| 7    |             |                           |         |
+-----+-----+-----+
2 rows in set (0.02 sec)

```

When `status` is `EXECUTE_INCREMENTAL_TASK`, it means inventory migration stage is successful, it's running on incremental migration stage.

Choose an idle time of business system, stop source database writing or stop upper database operation.

Stop source writing in proxy:

```

mysql> STOP SCALING SOURCE WRITING 0130317c30317c3054317c7363616c696e675f6462;
Query OK, 0 rows affected (0.07 sec)

```

Data consistency check:

```
mysql> CHECK SCALING 0130317c30317c3054317c7363616c696e675f6462 BY TYPE
(NAME=CRC32_MATCH);
+-----+-----+-----+-----+
| table_name | source_records_count | target_records_count | records_count_
matched | records_content_matched |
+-----+-----+-----+-----+
| t_order     | 6                  | 6                  | true
| true        |                   |                   |
| t_order_item | 6                  | 6                  | true
| true        |                   |                   |
+-----+-----+-----+-----+
2 rows in set (2.16 sec)
```

Apply metadata:

```
mysql> APPLY SCALING 0130317c30317c3054317c7363616c696e675f6462;
Query OK, 0 rows affected (0.22 sec)
```

Preview sharding again:

```
mysql> PREVIEW SELECT COUNT(1) FROM t_order;
+-----+-----+
| data_source_name | actual_sql
|                 |
+-----+-----+
| ds_2            | SELECT COUNT(1) FROM t_order_0 UNION ALL SELECT COUNT(1) FROM
t_order_1 |
| ds_3            | SELECT COUNT(1) FROM t_order_0 UNION ALL SELECT COUNT(1) FROM
t_order_1 |
| ds_4            | SELECT COUNT(1) FROM t_order_0 UNION ALL SELECT COUNT(1) FROM
t_order_1 |
+-----+-----+
3 rows in set (0.21 sec)
```

Sharding already take effect.

Optionally, unused `ds_0` and `ds_1` could be removed.

7.3 ShardingSphere-Sidecar

7.3.1 Introduction

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

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

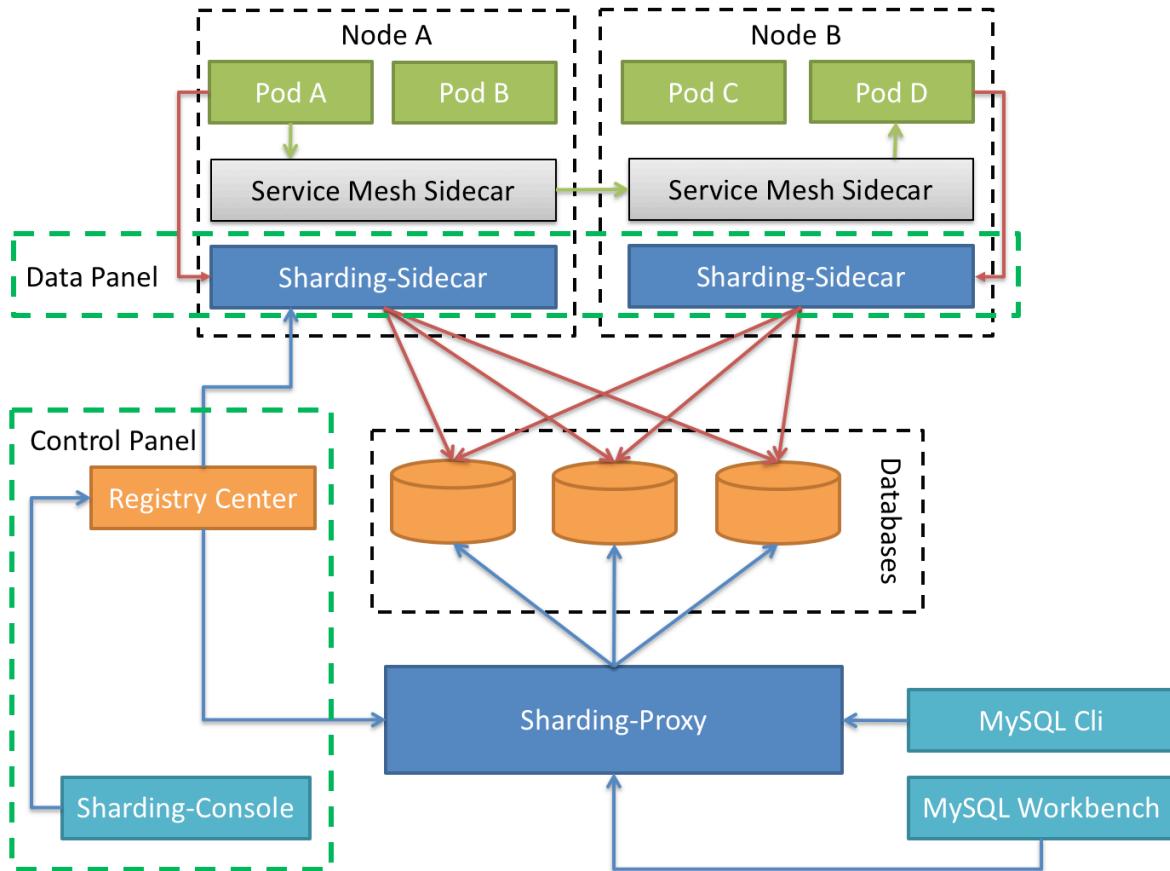


Figure8: ShardingSphere-Sidecar Architecture

7.3.2 Comparison

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

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

Apache ShardingSphere provides dozens of SPI based extensions. It is very convenient to customize the functions for developers.

This chapter lists all SPI extensions of Apache ShardingSphere. If there is no special requirement, users can use the built-in implementation provided by Apache ShardingSphere; advanced users can refer to the interfaces for customized implementation.

Apache ShardingSphere community welcomes developers to feed back their implementations to the open-source community, so that more users can benefit from it.

8.1 Mode

8.1.1 SPI Interface

<i>SPI Name</i>	<i>Description</i>
StandalonePersistRepository	Standalone mode configuration information persistence
ClusterPersistRepository	Cluster mode configuration information persistence
GovernanceWatcher	Governance listener

8.1.2 Sample

StandalonePersistRepository

<i>Implementation Class</i>	<i>Description</i>
FileRepository	File-based persistence
H2Repository	H2-based persistence

ClusterPersistRepository

<i>Implementation Class</i>	<i>Description</i>
CuratorZookeeperRepository	ZooKeeper-based persistence
EtcdRepository	Etcd-based persistence

GovernanceWatcher

<i>Implementation Class</i>	<i>Description</i>
ComputeNodeStateChangedWatcher	Compute node state change listener
DatabaseLockChangedWatcher	Database lock state change listener
DistributedLockChangedWatcher	Distributed lock change listener
GlobalRuleChangedWatcher	The global rule configuration change listener
MetaDataChangedWatcher	Metadata change listener
PropertiesChangedWatcher	Property change listener
StorageNodeStateChangedWatcher	Storage node state change listener

8.2 Configuration

8.2.1 RuleBuilder

Fully-qualified class name

```
`org.apache.shardingsphere.infra.rule.builder.RuleBuilder<https://github.com/apache/shardingsphere/blob/master/shardingsphere-infra/shardingsphere-infra-common/src/main/java/org/apache/shardingsphere/infra/rule/builder/RuleBuilder.java>`__
```

Definition

Used to convert user configurations into rule objects

Implementation classes

<i>Configuration Type</i>	<i>Description</i>	<i>Fully-qualified class name</i>
AuthorityRuleConfiguration	Used to convert authority user configuration into authority rule objects	`org.apache.shardingsphere.authority.rule.builder.AuthorityRuleBuilder`<https://github.com/apache/shardingsphere/blob/master/shardingsphere-authority/shardingsphere-authority-core/src/main/java/org/apache/shardingsphere/authority/rule/builder/AuthorityRuleBuilder.java>`__
SQLParserRuleConfiguration	Used to convert SQL parser user configuration into SQL parser rule objects	`org.apache.shardingsphere.parser.rule.builder.SQLParserRuleBuilder`<https://github.com/apache/shardingsphere/re-blob/master/shardingsphere-kerne/l/shardingsphere-re-parser/shardingsphere-parser-core/src/main/java/org/apache/shardingsphere/parser/rule/builder/SQLParserRuleBuilder.java>`__
TransactionRuleConfiguration	Used to convert transaction user configuration into transaction rule objects	`org.apache.shardingsphere.transaction.rule.builder.TransactionRuleBuilder`<https://github.com/apache/shardingsphere/blob/master/shardingsphere-kernel/shardingsphere-transaction/shardingsphere-transact-ion-core/src/main/java/org/apache/shardingsphere/transac-tion/rule/builder/TransactionRuleBuilder.java>`__
SingleTableRuleConfiguration	Used to convert single-table user configuration into a single-table rule objects	`org.apache.shardingsphere.singletable.rule.builder.SingleTableRuleBuilder`<https://github.com/apache/shardingsphere/blob/master/shardingsphere-single-table/shardingsphere-single-table-cor
8.2. Configuration		299

8.2.2 YamlRuleConfigurationSwapper

Fully-qualified class name

```
`org.apache.shardingsphere.infra.yaml.config.swapper.YamlRuleConfigurationSwapper  
<https://github.com/apache/shardingsphere/blob/master/shardingsphere-infra/shardingsphere-infra-common/src/main/java/org/apache/shardingsphere/infra/yaml/config/swapper/YamlRuleConfigurationSwapper.java>`__
```

Definition

Used to convert YAML configuration to standard user configuration

Implementation classes

<i>Con figuration Type</i>	<i>Description</i>	<i>Fully- qualified class name</i>
AUTHORITY	Used to convert the YAML configuration of authority rules into standard configuration of authority rules	`org.apache.shardingsphere.authority.yaml.swapper.AuthorityRuleConfigParser` < https://github.com/apache/shardingsphere/blob/master/shardingsphere-kernel/shardingsphere-auth/shardingsphere-auth-identity-core/src/main/java/org/apache/shardingsphere/auth/yaml/swapper/AuthorityRuleConfigParser.java >`__
SQL_PARSER	Used to convert the YAML configuration of the SQL parser into the standard configuration of the SQL parser	`org.apache.shardingsphere.parser.yaml.swapper.SQLParserRuleConfigParser` < https://github.com/apache/shardingsphere/blob/master/shardingsphere-kernel/shardingsphere-parser/shardingsphere-parser-core/src/main/java/org/apache/shardingsphere/parser/yaml/swapper/SQLParserRuleConfigParser.java >`__
TRANSACTION	Used to convert the YAML configuration of the transaction into the standard configuration of the transaction	`org.apache.shardingsphere.transaction.yaml.swapper.TransactionRuleConfigurationYamlSwapper` < https://github.com/apache/shardingsphere/blob/master/shardingsphere-transaction/shardingsphere-transaction-core/src/main/java/org/apache/shardingsphere/transaction/yaml/swapper/TransactionRuleConfigurationYamlSwapper.java >`__
SINGLE	Used to convert the YAML configuration of the single table into the standard configuration of the single table	`org.apache.shardingsphere.singleTable.yaml.config.wapper.SingleTableRuleConfigurationYamlSwapper` < https://github.com/apache/shardingsphere/blob/master/shardingsphere-single-table/shardingsphere-single-table-core/src/main/java/org/apache/shardingsphere/single-table/yaml/config/wapper/SingleTableRuleConfigurationYamlSwapper.java >`__
8.2. Configuration		301

8.2.3 ShardingSphereYamlConstruct

Fully-qualified class name

```
`org.apache.shardingsphere.infra.yaml.engine.constructor.  
ShardingSphereYamlConstruct <https://github.com/apache/shardingsphere/blob/master/shardingsphere-infra/shardingsphere-infra-common/src/main/java/org/apache/shardingsphere/infra/yaml/engine/constructor/ShardingSphereYamlConstruct.java>`__
```

Definition

Used to convert custom objects and YAML to and from each other

Implementation classes

<i>Configuration Type</i>	<i>Description</i>	<i>Fully-qualified class name</i>
YamlNoneShardingStrategyConfiguration	Used to convert non-sharding policy objects and YAML to and from each other	`org.apache.shardingsphere.sharding.yaml.engine.constructor.NoneShardingStrategyConfiguration< https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-sharding/shardingsphere-re-sharding-core/src/main/java/org/apache/shardingsphere/sharding/yaml/engine/construct/NoneShardingStrategyConfigurationYamlConstruct.java >`__

8.3 Kernel

8.3.1 SPI Interface

<i>SPI Name</i>	<i>Description</i>
SQLRouter	Used to process routing results
SQLRewriteContextDecorator	Used to handle SQL rewrite results
SQLExecutionHook	SQL execution process listener
ResultProcessEngine	Used to process result sets
StoragePrivilegeHandler	Use SQL dialect to process privilege metadata
DynamicDataSourceStrategy	Dynamic data source fetch strategy

8.3.2 Sample

SQLRouter

<i>Implementation Class</i>	<i>Description</i>
ReadWriteSplittingSQLRouter	Used to process read-write splitting routing results
DatabaseDiscoverySQLRouter	Used to process database discovery routing results
SingleTableSQLRouter	Used to process single-table routing results
ShardingSQLRouter	Used to process sharding routing results
ShadowSQLRouter	Used to process shadow database routing results

SQLRewriteContextDecorator

<i>Implementation Class</i>	<i>Description</i>
ShardingSQLRewriteContextDecorator	Used to process sharding SQL rewrite results
EncryptSQLRewriteContextDecorator	Used to process encryption SQL rewrite results

SQLExecutionHook

<i>Implementation Class</i>	<i>Description</i>
TransactionalSQLExecutionHook	Transaction hook of SQL execution

ResultProcessEngine

<i>Implementation Class</i>	<i>Description</i>
ShardingResultMergerEngine	Used to handle sharding result set merge
EncryptResultDecoratorEngine	Used to handle encrypted result set overrides

StoragePrivilegeHandler

<i>Implementation Class</i>	<i>Description</i>
PostgreSQLPrivilegeHandler	Use PostgreSQL dialect to process privilege metadata
SQLServerPrivilegeHandler	Use SQLServer dialect to process privilege metadata
OraclePrivilegeHandler	Use Oracle dialect to process privilege metadata
MySQLPrivilegeHandler	Use MySQL dialect to process privilege metadata

DynamicDataSourceStrategy

<i>Implementation Class</i>	<i>Description</i>
DatabaseDiscoveryDynamicDataSourceStrategy	Use database discovery to dynamic fetch data source

8.4 DataSource

8.4.1 SPI Interface

SPI Name	Description
DatabaseType	Supported database types
DialectTableMetaDataLoader	Use SQL dialect to load meta data rapidly
DataSourcePoolMetaData	Data source connection pool metadata
DataSourcePoolActiveDetector	Data source connection pool active detector

8.4.2 Sample

DatabaseType

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

DialectTableMetaDataLoader

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

DataSourcePoolMetaData

<i>Implementation Class</i>	<i>Description</i>
DBCPDataSourcePoolMetaData	DBCP data source pool meta data
HikariDataSourcePoolMetaData	Hikari data source pool meta data

DataSourcePoolActiveDetector

<i>Implementation Class</i>	<i>Description</i>
DefaultDataSourcePoolActiveDetector	Default data source pool active detector
HikariDataSourcePoolActiveDetector	Hikari data source pool active detector

8.5 SQL Parser

8.5.1 DatabaseTypedSQLParserFacade

Fully-qualified class name

```
`org.apache.shardingsphere.sql.parser.spi.DatabaseTypedSQLParserFacade <https://github.com/apache/shardingsphere/blob/master/shardingsphere-sql-parser/shardingsphere-sql-parser-spi/src/main/java/org/apache/shardingsphere/sql/parser/spi/DatabaseTypedSQLParserFacade.java>`__
```

Definition

Database typed SQL parser facade service definition

Implementation classes

<i>Configuration Type*</i>	<i>Description</i>	<i>Fully-qualified class name</i>
MySQL	SQL parser entry based on MySQL	`org.apache.shardingsphere.sql.parser.mysql.parser.MySQLParserFacade <https://github.com/apache/shardingsphere/blob/master/shardingsphere-sharding-sql-parser/mysql/src/main/java/org/apache/shardingsphere/sql/parser/mysql/parser/MySQLParserFacade.java>`__
PostgreSQL	SQL parser entry based on PostgreSQL	`org.apache.shardingsphere.sql.parser.postgresql.parser.PostgreSQLParserFacade <https://github.com/apache/shardingsphere/blob/master/shardingsphere-sql-parser/shardingsphere-sql-parser-dialect/sharding-sql-parser-postgresql/src/main/java/org/apache/shardingsphere/sql-parser/postgresql/parser/PostgreSQLParserFacade.java>`__
SQLServer	SQL parser entry based on SQLServer	`org.apache.shardingsphere.sql.parser.sqls.SQLServerParserFacade <https://github.com/apache/shardingsphere/blob/master/shardingsphere-sql-parser/sqlserver-sharding-sphere-sql-parser/src/main/java/org/apache/shardingsphere/sql-parser/sqlserver/SQLServerParserFacade.java>`__
8.5. SQL Parser		dialect/shardingsphere-307 sql-parser-shardingsphere-307 /main/java/org/apache/shardingsphere/sql-parser/

8.5.2 SQLVisitorFacade

Fully-qualified class name

```
`org.apache.shardingsphere.sql.parser.spi.SQLVisitorFacade <https://github.com/apache/shardingsphere/blob/master/shardingsphere-sql-parser/shardingsphere-sql-parser-spi/src/main/java/org/apache/shardingsphere/sql/parser/spi/SQLVisitorFacade.java>`__
```

Definition

SQL visitor facade class definition

Implementation classes

.	<i>Description</i>	<i>Fully-qualified class name</i>
Configuration Type*	String.join(".", getDatabaseType(), getVisitorType())	`org.apache.shardingsphere.sql.parser.spi.SQLVisitorFacade <https://github.com/apache/shardingsphere/blob/master/shardingsphere-sql-parser/shardingsphere-sql-parser-spi/src/main/java/org/apache/shardingsphere/sql/parser/spi/SQLVisitorFacade.java>`__

8.6 Proxy

8.6.1 DatabaseProtocolFrontendEngine

Fully-qualified class name

```
`org.apache.shardingsphere.proxy.frontend.spi.DatabaseProtocolFrontendEngine <https://github.com/apache/shardingsphere/blob/master/shardingsphere-proxy/shardingsphere-proxy-frontend/shardingsphere-proxy-frontend-spi/src/main/java/org/apache/shardingsphere/proxy/frontend/spi/DatabaseProtocolFrontendEngine.java>`__
```

Definition

Protocols for ShardingSphere-Proxy to parse and adapt for accessing databases.

Implementation classes

• ConfigurationType*	• Description*	<i>Fully-qualified class name</i>
MySQL	Protocol implementation for MySQL	`org.apache.shardingsphere.proxy.frontend.mysql.MySQLFrontendEngine <https://github.com/apache/shardingsphere/blob/mingSphere-proxy/shardingsphere-proxy-frontend/shardingsphere-proxy-frontend-mysql/src/main/java/org/apache/shardingsphere/proxy/frontend/mysql/MySQLFrontendEngine.java>`
PostgreSQL	Protocol implementation for PostgreSQL	`org.apache.shardingsphere.proxy.frontend.postgresql.PostgreSQLFrontendEngine <https://github.com/apache/shardingsphere/blob/master/shardingproxy/shardingsphere-proxy-frontend/shardingsphere-proxy-front-end-postgresql/src/main/java/org/apache/shardingsphere/proxy/frontend/postgresql/PostgreSQLFrontendEngine.java>`
openGauss	Protocol implementation for openGauss	`org.apache.shardingsphere.proxy.frontend.opengauss.OpenGaussFrontendEngine <https://github.com/apache/shardingsphere/blob/master/shardingproxy/shardingsphere-proxy-frontend/shardingsphere-proxy-front-end-opengauss/src/main/java/org/apache/shardingsphere/proxy/frontend/opengauss/OpenGaussFrontendEngine.java>`

8.6.2 AuthorityProvideAlgorithm

Fully-qualified class name

```
`org.apache.shardingsphere.authority.spi.AuthorityProviderAlgorithm <https://github.com/apache/shardingsphere/blob/master/shardingsphere-kernel/shardingsphere-authority/shardingsphere-authority-api/src/main/java/org/apache/shardingsphere/authority/spi/AuthorityProviderAlgorithm.java>`__
```

Definition

Loading logic for user permission.

Implementation classes

<i>ConfigurationType*</i>	<i>Description</i>	<i>Fully-qualified class name</i>
ALL_PERMITTED	Grant all permissions by default (no restrictions)	`org.apache.shardingsphere.authority.provider.simple.AllPermittedPrivilegesProviderAlgorithm <https://github.com/apache/shardingsphere/blob/master/shardingsphere-kernel/shardingsphere-authority/shardingsphere-authority-core/src/main/java/org/apache/shardingsphere/authority/simple/AllPermittedPrivilegesProviderAlgorithm.java>`__
DATA_BASE_PERMITTED	Permissions configured by user-data-base-map-pings	`org.apache.shardingsphere.authority.provider.database.DatabasePermittedPrivilegesProviderAlgorithm <https://github.com/apache/shardingsphere/blob/master/shardingsphere-kernel/sharding-sphere-authority/shardingsphere-authority-core/src/main/java/org/apache/shardingsphere/authority/provider/DatabasePermittedPrivilegesProviderAlgorithm.java>`__

8.7 Data Sharding

8.7.1 SPI Interface

SPI Name	Description
ShardingAlgorithm	Sharding Algorithm
KeyGenerateAlgorithm	Distributed Key Generating Algorithm
ShardingAuditAlgorithm	Sharding audit algorithm
DatetimeService	Obtain the current date for routing
DatabaseSQLEntry	Obtain database dialects of current date

8.7.2 Sample

ShardingAlgorithm

Implementation Class	Description
BoundaryBased RangeShardingAlgorithm	Boundary based range sharding algorithm
VolumeBased RangeShardingAlgorithm	Volume based range sharding algorithm
ComplexInlineShardingAlgorithm	Complex inline sharding algorithm
AutoIntervalShardingAlgorithm	Mutable interval sharding algorithm
Class BasedShardingAlgorithm	Class based sharding algorithm
HintInlineShardingAlgorithm	Hint inline sharding algorithm
FixedIntervalShardingAlgorithm	Fixed interval sharding algorithm
HashModShardingAlgorithm	Hash modulo sharding algorithm
InlineShardingAlgorithm	Inline sharding algorithm
ModShardingAlgorithm	Modulo sharding algorithm
CosIdModShardingAlgorithm	Modulo sharding algorithm provided by CosId
CosIdIntervalShardingAlgorithm	Fixed interval sharding algorithm provided by CosId
CosIdSnowflakeIntervalShardingAlgorithm	Snowflake key-based fixed interval sharding algorithm provided by CosId

KeyGenerateAlgorithm

<i>Implementation Class</i>	<i>Description</i>
Sno wflakeKeyGenerateAlgorithm	Snowflake key generate algorithm
UUIDKeyGenerateAlgorithm	UUID key generate algorithm
CosIdKeyGenerateAlgorithm	CosId key generate algorithm
CosIdSno wflakeKeyGenerateAlgorithm	Snowflake key generate algorithm provided by CosId
NanoIdKeyGenerateAlgorithm	NanoId key generate algorithm

ShardingAuditAlgorithm

<i>Implementation Class</i>	<i>Description</i>
DMLShardingConditionsShardingAuditAlgorithm	Prohibit DML auditing algorithm without sharding conditions

DatetimeService

<i>Implementation Class</i>	<i>Description</i>
DatabaseDatetimeServiceDelegate	Get the current time from the database for routing
SystemDatetimeService	Get the current time from the application system for routing

DatabaseSQLEntry

<i>Implementation Class</i>	<i>Description</i>
MySQLDatabaseSQLEntry	MySQL dialect for get current time
PostgreSQLDatabaseSQLEntry	PostgreSQL dialect for get current time
OracleDatabaseSQLEntry	Oracle dialect for get current time
SQLServerDatabaseSQLEntry	SQLServer dialect for get current time

8.8 Readwrite-splitting

8.8.1 ReadQueryLoadBalanceAlgorithm

Fully-qualified class name

```
`org.apache.shardingsphere.readwritesplitting.spi.ReadQueryLoadBalanceAlgorithm
<https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-
readwrite-splitting/shardingsphere-readwrite-splitting-api/src/main/java/org/apache/shardingsphe
re/readwritesplitting/spi/ReadQueryLoadBalanceAlgorithm.java>`__
```

Definition

Read query load balance algorithm's definition

Implementation classes

<i>Configuration Type*</i>	<i>Description</i>	<i>Fully-qualified class name</i>
ROUND_ROBIN	the read database load balancer algorithm based on polling	<code>org.apache.shardingsphere.readwritesplitting.algorithm.loadbalance.RoundRobinReadQueryLoadBalanceAlgorithm` <`__</code></td></tr> <tr> <td>RANDOM</td><td>the read database load balancer algorithm based on random</td><td><code>org.apache.shardingsphere.readwritesplitting.algorithm.loadbalance.RandomReadQueryLoadBalanceAlgorithm` <`__<="" readwritesplitting="" shardingsphere="" shardingsphere-features="" shardingsphere-readwrite-splitting="" shardingsphere-readwrite-splitting-core="" src=""></code>
WEIGHT	the read database load balancer algorithm based on weight	<code>org.apache.shardingsphere.readwritesplitting.algorithm.loadbalance.WeightReadQueryLoadBalanceAlgorithm` <<a href="https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-readwrite-splitting/algorithmloadbalance/WeightReadQueryLoadBalanceAlgorithm.java&gt;`__</code></code>
8.8. Readwrite-splitting		314

8.9 HA

8.9.1 DatabaseDiscoveryProviderAlgorithm

Fully-qualified class name

```
`org.apache.shardingsphere.dbdiscovery.spi.DatabaseDiscoveryProviderAlgorithm  
<https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-db-discovery/shardingsphere-db-discovery-api/src/main/java/org/apache/shardingsphere/dbdiscovery/spi/DatabaseDiscoveryProviderAlgorithm.java>`__
```

Definition

Database discovery provider algorithm's definition

Implementation classes

<i>Configuration Type*</i>	<i>Description</i>	<i>Fully-qualified class name</i>
MySQL.MGR	MySQL MGR-based database discovery provider algorithm	`org.apache.shardingsphere.dbdiscovery.mysql.type.MGRMySQLDatabaseDiscoveryProviderAlgorithm<https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/sharedingsphere-db-discovery-provider/shardingsphere-db-discovery-mysql/src/main/java/org/apache/shardingsphere/dbdiscovery/mysql/type/MGRMySQLDatabaseDiscoveryProviderAlgorithm.java>`__
MySQL.NORMAL_REPLICATION	Database discovery provider algorithm of MySQL's replication	`org.apache.shardingsphere.dbdiscovery.mysql.type.MySQLNormalReplicationDatabaseDiscoveryProviderAlgorithm <https://github.com/apache/shardingsphere/blob/master/sharedingsphere-features/sharedingsphere-db-discovery/shardingsphere-db-discovery-provider/shardingsphere-db-discovery-mysql/src/main/java/org/apache/shardingsphere/dbdiscovery/mysql/type/MySQLNormalReplicationDatabaseDiscoveryProviderAlgorithm.java>`__
8.0P HA openGauss.NORMAL_REPLICATION	Database discovery provider algorithm of openGauss' s replication	`org.apache.shardingsphere.dbdiscovery.opengauss.OpenGaussNormalReplicationDatabaseDiscoveryProviderAlgorithm` 316

8.10 Distributed Transaction

8.10.1 ShardingSphereTransactionManager

Fully-qualified class name

```
`org.apache.shardingsphere.transaction.spi.ShardingSphereTransactionManager  
<https://github.com/apache/shardingsphere/blob/master/shardingsphere-kernel/shardingsphere-transaction/shardingsphere-transaction-core/src/main/java/org/apache/shardingsphere/transaction/spi/ShardingSphereTransactionManager.java>`__
```

Definition

ShardingSphere transaction manager service definition

Implementation classes

<i>Configuration Type*</i>	<i>Description</i>	<i>Fully-qualified class name</i>
Trans actionType.XA	XA distributed transaction manager	`org.apache.shardingsphere.transaction.xa.XAShadingSphereTransactionManager < https://github.com/apache/shardingsphere/blob/master/shardingsphere-kernel/shardingsphere-transaction/shardingsphere-transaction-type/shardingsphere-transaction-xa/shardingsphere-transaction-xa-core/src/main/java/org/apache/shardingsphere/transaction/xa/XAShadingSphereTransactionManager.java >`__
Transac tionType.BASE	Seata distributed transaction manager	`org.apache.shardingsphere.transaction.base.seata.at.SeataATShadingSphereTransactionManager < https://github.com/apache/shardingsphere/blob/master/shardingsphere-kernel/shardingsphere-transaction/shardingsphere-transaction-base/seata-at/src/main/java/org/apache/shardingsphere/transaction/base/seata/at/SeataATShadingSphereTransactionManager.java >`__

8.10.2 XATransactionManagerProvider

Fully-qualified class name

```
`org.apache.shardingsphere.transaction.xa.spi.XATransactionManagerProvider  
<https://github.com/apache/shardingsphere/blob/master/shardingsphere-kernel/shardingsphere-transaction/shardingsphere-transaction-type/shardingsphere-transaction-xa/shardingsphere-transaction-xa-spi/src/main/java/org/apache/shardingsphere/transaction/xa/spi/XATransactionManagerProvider.java>`__
```

Definition

XA transaction manager provider definition

Implementation classes

<i>Configuration Type*</i>	<i>Description</i>	<i>Fully-qualified class name</i>
Atomikos	XA distributed transaction manager based on Atomikos	<code>``org.apache.shardingsphere.transaction.xa.atomikos.manager.AtomikosTransactionManagerProvider` <``</code></td></tr> <tr> <td>Narayana</td><td>XA distributed transaction manager based on Narayana</td><td><code>``org.apache.shardingsphere.transaction.xa.narayana.manager.NarayanaXATransactionManagerProvider` <<a href=" https:="" java="" main="" master="" org="" shardingsphere="" shardingsphere-kernel="" shardingsphere-transaction="" shardingsphere-transaction-xa="" shardingsphere-transactions-xa-narayana="" shardingsphere-transactions-xa-provider="" shardingsphere-type="" src="" t<=""></code>
8.10. Distributed Transaction		<code>ransaction/xa/narayana/manager/NarayanaXATransactionManagerProvider.java>`</code>

8.10.3 XADataSourceDefinition

Fully-qualified class name

```
`org.apache.shardingsphere.transaction.xa.jta.datasource.properties.  
XADataSourceDefinition <https://github.com/apache/shardingsphere/blob/master/shardingsphere-kernel/shardingsphere-transaction/shardingsphere-transaction-type/shardingsphere-transaction-xa/shardingsphere-transaction-xa-core/src/main/java/org/apache/shardingsphere/transaction/xa/jta/datasource/properties/XADataSourceDefinition.java>`__
```

Definition

XA Data source definition

Implementation classes

<i>Configuration Type*</i>	<i>Description</i>	<i>Fully-qualified class name</i>
MySQL	Auto convert Non XA MySQL data source to XA MySQL data source	`org.apache.shardingsphere.transaction.xa.jta.datasource.properties.dialect.MySQLXADatasourceDefinition <https://github.com/apache/shardingsphere/blob/master/shardingsphere-kernel/shardingsphere-transaction/shardingsphere-transaction-type/shardingsphere-transaction-xa/shardingsphere-transaction-xa-core/src/main/java/org/apache/shardingsphere/transation/xa/jta/datasource/properties/dialect/MySQLXADatasourceDefinition.java>`__
MariaDBXADatasourceDefinition	Auto convert Non XA MariaDB data source to XA MariaDB data source	`org.apache.shardingsphere.transaction.xa.jta.datasource.properties.dialect.MariaDBXADatasourceDefinition <https://github.com/apache/shardingsphere/blob/master/shardingsphere-kernel/shardingsphere-transaction/shardingsphere-transaction-type/shardingsphere-transaction-xa/shardingsphere-transaction-xa-core/src/main/java/org/apache/shardingsphere/transation/xa/jta/datasource/properties/`__
8.10. Distributed Transaction		ties/dialect/ MariaDBXADatasourceDefinition.java>`__
PostgreSQLXADatasourceDefinition	Auto convert Non XA PostgreSQL data source to XA	`org.apache.shardingsphere.datasource/dialect/postgresql/`__

8.10.4 DataSourcePropertyProvider

Fully-qualified class name

```
`org.apache.shardingsphere.transaction.xa.jta.datasource.swapper.DataSourcePropertyProvider <https://github.com/apache/shardingsphere/blob/master/shardingsphere-kernel/shardingsphere-transaction/shardingsphere-transaction-type/shardingsphere-transaction-xa/shardingsphere-transaction-xa-core/src/main/java/org/apache/shardingsphere/transaction/xa/jta/datasource/swapper/DataSourcePropertyProvider.java>`__
```

Definition

Data source property provider service definition

Implementation classes

<i>.</i> Configuration Type*	<i>Description</i>	<i>Fully-qualified class name</i>
com.zaxxer.hikari.HikariDataSource	Used to get standard properties of HikariCP	`org.apache.shardingsphere.transaction.xa.jta.datasource.swapper.impl.HikariCPProvider < https://github.com/apache/shardingsphere/blob/master/shardingsphere-kernel/shardingsphere-transaction/shardingsphere-transaction-type/shardingsphere-transaction-xa/shardingsphere-transaction-xa-core/src/main/java/org/apache/shardingsphere/transaction/xa/jta/datasource/swapper/impl/HikariCPProvider.java >`__

8.11 SQL Checker

8.11.1 SQLChecker

Fully-qualified class name

```
`org.apache.shardingsphere.infra.executor.check.SQLChecker <https://github.com/apache/shardingsphere/blob/master/shardingsphere-infra/shardingsphere-infra-executor/src/main/java/org/apache/shardingsphere/infra/executor/check/SQLChecker.java>`__
```

Definition

SQL checker class definition

Implementation classes

<i>Configuration Type*</i>	<i>Description</i>	<i>Fully-qualified class name</i>
AuthorityRule.class	Authority checker	`org.apache.shardingsphere.authORITY.checker.AuthorityChecker <https://github.com/apache/shardingsphere/blob/master/shardingkernel/shardingsphere-authority/shardingsphere-autority-core/src/main/java/org/apache/shardingsphere/authORITY/checker/AuthorityChecker.java>`__
ShardingAuditChecker	Sharding audit checker	`org.apache.shardingsphere.sharding.checker.audit.ShardingAuditChecker <https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-re-sharding/shardingsphere-sharding-core/src/main/java/org/apache/shardingsphere/sharding/checker/audit/ShardingAuditChecker.java>`__

8.12 Encryption

8.12.1 EncryptAlgorithm

Fully-qualified class name

```
`org.apache.shardingsphere.encrypt.spi.EncryptAlgorithm<https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-encrypt/shardingsphere-encrypt-api/src/main/java/org/apache/shardingsphere/encrypt/spi/EncryptAlgorithm.java>`__
```

Definition

Data encryption algorithm definition

Implementation classes

<i>.</i> Configuration Type*	<i>Description</i>	<i>Fully-qualified class name</i>
MD5	MD5 data encrypt algorithm	`org.apache.shardingsphere.encryption.algorithm.MD5Encrypt` < https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-encrypt/shardingsphere-encrypt-shardingsphere-encrpt-core/src/main/java/org/apache/shardingsphere/encryption/algorithm/MD5EncryptAlgorithm.java >`__
AES	AES data encrypt algorithm	`org.apache.shardingsphere.encryption.algorithm.AESEncrypt` < https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-encrypt/shardingsphere-encrypt-shardingsphere-encrpt-core/src/main/java/org/apache/shardingsphere/encryption/algorithm/AESEncryptAlgorithm.java >`__
RC4	RC4 data encrypt algorithm	`org.apache.shardingsphere.encryption.algorithm.RC4Encrypt` < https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-encrypt/shardingsphere-encrypt-shardingsphere-encrpt-core/src/main/java/org/apache/shardingsphere/encryption/algorithm/RC4EncryptAlgorithm.java >`__
8.12. Encryption		rithm.java>`__
SM3	SM3 data encrypt algorithm	`org.apache.shardingsphere.encryption.algorithm.SM3Encrypt`

8.13 Shadow DB

8.13.1 ShadowAlgorithm

Fully-qualified class name

```
`org.apache.shardingsphere.shadow.spi.ShadowAlgorithm <https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-shadow/shardingsphere-shadow-api/src/main/java/org/apache/shardingsphere/shadow/spi/ShadowAlgorithm.java>` __
```

Definition

Shadow algorithm's definition

Implementation classes

<i>.</i> Configuration Type*	<i>Description</i>	<i>Fully-qualified class name</i>
VALUE_MATCH	Match shadow algorithms based on field values	`org.apache.shardingsphere.shadow.algorithm.ColumnValueMatchShadowAlg < https://github.com/apache/shardingsphere/blob/master/shardingsphere/features/sharding/shadow/column/ColumnValueMatchShadowAlgorithm.java >`__
REGEX_MATCH	Regular matching shadow algorithm based on field value	`org.apache.shardingsphere.shadow.algorithm.ColumnRegexMatchShadowAlg < https://github.com/apache/shardingsphere/blob/master/shardingsphere/features/sharding/shadow/column/ColumnRegexMatchShadowAlgorithm.java >`__
SIMPLE_HINT	Simple match shadow algorithm based on Hint	`org.apache.shardingsphere.shadow.algorithm.SimpleHintShadowAlgorithm < https://github.com/apache/shardingsphere/blob/master/shardingsphere-features/shardingsphere-shad >`__
8.13. Shadow DB		ow/shardingsphere-shadow-core/src/main/java/org/apache/shardingsphere/shadow

8.14 Observability

8.14.1 PluginBootService

Fully-qualified class name

```
`org.apache.shardingsphere.agent.spi.boot.PluginBootService <https://github.com/apache/shardingsphere/blob/master/shardingsphere-agent/shardingsphere-agent-api/src/main/java/org/apache/shardingsphere/agent/spi/boot/PluginBootService.java>` __
```

Definition

Plugin startup service definition

Implementation classes

<i>Configuration Type*</i>	<i>Description</i>	<i>Fully-qualified class name</i>
Prometheus	Prometheus plugin startup class	` org.apache.shardingsphere.agent.metrics.prometheus.service.PrometheusPluginBootService <https://github.com/apache/shardingsphere/blob/master/shardingsphere-agent/shardingsphere-agent-plugin-metrics-prometheus/src/main/java/org/apache/shardingsphere/agent/metrics/prometheus/service/PrometheusPluginBootstrapService.java>`__
Logging	Logging plugin startup class	`org.apache.shardingsphere.agent.plugin.logging.base.service.BaseLoggingPluginBootService <https://github.com/apache/shardingsphere/blob/master/shardingsphere-agent/plugins/shardingsphere-agent-plugin-logging/logging-base/src/main/java/org/apache/shardingsphere/agent/plugin/logging/base/service/BaseLoggingPluginBootstrapService.java>`__
Jaeger	Jaeger plugin startup class	`org.apache.shardingsphere.agent.plugin.tracing.jaeger.service.JaegerTracingPluginBootstrapService <https://github.com/apache/shardingsphere/blob/master/shardingsphere-agent/plugins/tracing/jaeger/src/main/java/org/apache/shardingsphere/agent/plugin/tracing/jaeger/service/JaegerTracingPluginBootstrapService.java>`__
8.14. Observability		ingPluginBootstrapService <https://github.com/apache/shardingsphere/blob/master/shardingsphere-agent/plugins/observability/src/main/java/org/apache/shardingsphere/agent/plugin/observability/service/ObservabilityPluginBootstrapService.java>`__

8.14.2 PluginDefinitionService

Fully-qualified class name

```
`org.apache.shardingsphere.agent.spi.definition.PluginDefinitionService <http  
s://github.com/apache/shardingsphere/blob/master/shardingsphere-agent/shardingsphere-agent-api  
/src/main/java/org/apache/shardingsphere/agent/spi/definition/PluginDefinitionService.java>`__
```

Definition

Agent plugin definition

Implementation classes

<i>Configuration Type*</i>	<i>Description</i>	<i>Fully-qualified class name</i>
Prometheus	Prometheus plugin definition	<code>org.apache.shardingsphere.agent.metrics.prometheus.definition.PrometheusPluginDefinitionService <<a >`__<="" code="" href="https://github.com/apache/shardingsphere/blob/master/shardingsphere-agent/shardingsphere-agent-plugins/shardingsphere-agent-metrics/shardingsphere-agent-metrics-prometheus/src/main/java/org/apache/shardingsphere/agent/metrics/prometheus/definition/PrometheusPluginDefinitionService.java"></code>
Logging	Logging plugin definition	<code>org.apache.shardingsphere.agent.plugin.logging.base.definition.BaseLoggingPluginDefinitionService <<a >`__<="" code="" href="https://github.com/apache/shardingsphere/blob/master/shardingsphere-agent/plugins/shardingsphere-agent-plugin/logging/shardingsphere-agent-logging-base/src/main/java/org/apache/shardingsphere/agent/plugin/logging/base/definition/BaseLoggingPluginDefinitionService.java"></code>
Jaeger	Jaeger plugin definition	<code>org.apache.shardingsphere.agent.plugin.tracing.jaeger.definition.JaegerPluginDefinitionService <<a >`__<="" code="" href="https://github.com/apache/shardingsphere/blob/master/shardingsphere-agent/plugins/shardingsphere-agent-tracing/jaeger/tracing/jaegegerdefinition/JaegerPluginDefinitionService.java"></code>
8.14. Observability		333

Apache ShardingSphere provides test engines for integration, module and performance.

9.1 Integration Test

Provide point to point test which connect real ShardingSphere and database instances.

They define SQLs in XML files, engine run for each database independently. All test engines designed to modify the configuration files to execute all assertions without any **Java code** modification. It does not depend on any third-party environment, ShardingSphere-Proxy and database used for testing are provided by docker image.

9.2 Module Test

Provide module test engine for complex modules.

They define SQLs in XML files, engine run for each database independently too. It includes SQL parser and SQL rewriter modules.

9.3 Performance Test

Provide multiple performance test methods, includes Sysbench, JMH or TPCC and so on.

9.4 Sysbench Test

9.5 Integration Test

The SQL parsing unit test covers both SQL placeholder and literal dimension. Integration test can be further divided into two dimensions of strategy and JDBC; the former one includes strategies as Sharding, table Sharding, database Sharding, and readwrite-splitting while the latter one includes Statement and PreparedStatement.

Therefore, one SQL can drive 5 kinds of database parsing * 2 kinds of parameter transmission modes + 5 kinds of databases * 5 kinds of Sharding strategies * 2 kinds of JDBC operation modes = 60 test cases, to enable ShardingSphere to achieve the pursuit of high quality.

9.5.1 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,dtbl_with_replica_query,replica_query

# database type, could define multiple databases(H2,MySQL,Oracle,SQLServer,
PostgreSQL)
it.cluster.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.

Running Integration Tests

Run with Docker

```
./mvnw -B clean install -f shardingsphere-test/shardingsphere-integration-test/pom.xml -Pit.env.docker -Dit.cluster.adapters=proxy,jdbc -Dit.scenarios=${scenario_name_1,scenario_name_1,scenario_name_n} -Dit.cluster.databases=MySQL
```

Running the above command will build a Docker image apache/shardingsphere-proxy-test:latest for integration testing. The existing test Docker image can be reused without rebuilding if only the test code is modified. Use the following command to skip the image building and run the integration tests directly:

```
./mvnw -B clean install -f shardingsphere-test/shardingsphere-integration-test/shardingsphere-integration-test-suite/pom.xml -Pit.env.docker -Dit.cluster.adapters=proxy,jdbc -Dit.scenarios=${scenario_name_1,scenario_name_1,scenario_name_n} -Dit.cluster.databases=MySQL
```

9.5.2 Notice

1. If Oracle needs to be tested, please add Oracle driver dependencies to the pom.xml.
2. 10 splitting-databases and 10 splitting-tables are used in the integrated test to ensure the test data is full, so it will take a relatively long time to run the test cases.

9.6 Performance Test

Provides result for each performance test tools.

9.7 Module Test

Provides test engine with each complex modules.

9.7.1 SQL Parser Test

Prepare Data

Not like Integration test, SQL parse test does not need a specific database environment, just define the sql to parse, and the assert data:

SQL Data

As mentioned sql-case-id in Integration test, test-case-id could be shared in different module to test, and the file is at shardingsphere-sql-parser/shardingsphere-sql-parser-test/src/main/resources/sql/supported/\${SQL-TYPE}/*.xml

Assert Data

The assert data is at shardingsphere-sql-parser/shardingsphere-sql-parser-test/src/main/resources/case/\${SQL-TYPE}/*.xml in that xml file, it could assert against the table name, token or sql condition and so on. For example:

```
<parser-result-sets>
    <parser-result sql-case-id="insert_with_multiple_values">
        <tables>
            <table name="t_order" />
        </tables>
        <tokens>
            <table-token start-index="12" table-name="t_order" length="7" />
        </tokens>
        <sharding-conditions>
            <and-condition>
                <condition column-name="order_id" table-name="t_order" operator=
"EQUAL">
                    <value literal="1" type="int" />
                </condition>
                <condition column-name="user_id" table-name="t_order" operator=
"EQUAL">
                    <value literal="1" type="int" />
                </condition>
            </and-condition>
            <and-condition>
                <condition column-name="order_id" table-name="t_order" operator=
"EQUAL">
                    <value literal="2" type="int" />
                </condition>
                <condition column-name="user_id" table-name="t_order" operator=
"EQUAL">
                    <value literal="2" type="int" />
                </condition>
            </and-condition>
        </sharding-conditions>
    </parser-result>
</parser-result-sets>
```

```

        </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.

9.7.2 SQL Rewrite Test

Target

Facing logic databases and tables cannot be executed directly in actual databases. SQL rewrite is used to rewrite logic SQL into rightly executable ones in actual databases, including two parts, correctness rewrite and optimization rewrite. rewrite tests are for these targets.

Test

The rewrite tests are in the test folder under `sharding-core/sharding-core-rewrite`. Followings are the main part for rewrite tests:

- test engine
- environment configuration
- assert data

Test engine is the entrance of rewrite tests, just like other test engines, through Junit [Parameterized](#), read every and each data in the xml file under the target test type in `test\resources`, and then assert by the engine one by one

Environment configuration is the yaml file under test type under `test\resources\yaml`. The configuration file contains `dataSources`, `shardingRule`, `encryptRule` and other info. for example:

```

dataSources:
  db: !!com.zaxxer.hikari.HikariDataSource
    driverClassName: org.h2.Driver
    jdbcUrl: jdbc:h2:mem:db;DB_CLOSE_DELAY=-1;DATABASE_TO_UPPER=false;MODE=MYSQL
    username: sa
    password:

## sharding Rules
rules:
- !SHARDING
  tables:
    t_account:
      actualDataNodes: db.t_account_${0..1}
```

```

tableStrategy:
  standard:
    shardingColumn: account_id
    shardingAlgorithmName: account_table_inline
keyGenerateStrategy:
  column: account_id
  keyGeneratorName: snowflake
t_account_detail:
  actualDataNodes: db.t_account_detail_${0..1}
  tableStrategy:
    standard:
      shardingColumn: order_id
      shardingAlgorithmName: account_detail_table_inline
bindingTables:
  - t_account, t_account_detail
shardingAlgorithms:
  account_table_inline:
    type: INLINE
    props:
      algorithm-expression: t_account_${account_id % 2}
  account_detail_table_inline:
    type: INLINE
    props:
      algorithm-expression: t_account_detail_${account_id % 2}
keyGenerators:
  snowflake:
    type: SNOWFLAKE

```

Assert data are in the xml under test type in `test\resources`. In the xml file, `yaml-rule` means the environment configuration file path, `input` contains the target SQL and parameters, `output` contains the expected SQL and parameters. The `db-type` described the type for SQL parse, default is `SQL92`. For example:

```

<rewrite-assertions yaml-rule="yaml/sharding/sharding-rule.yaml">
  <!-- to change SQL parse type, change db-type -->
  <rewrite-assertion id="create_index_for_mysql" db-type="MySQL">
    <input sql="CREATE INDEX index_name ON t_account ('status')" />
    <output sql="CREATE INDEX index_name ON t_account_0 ('status')" />
    <output sql="CREATE INDEX index_name ON t_account_1 ('status')" />
  </rewrite-assertion>
</rewrite-assertions>

```

After set up the assert data and environment configuration, rewrite test engine will assert the corresponding SQL without any Java code modification.

9.8 Scaling Integration Test

9.8.1 Objective

Verify the correctness of Scaling's own functionality and dependent modules.

9.8.2 Environment

There are two types of environment preparation: Native and Docker

- Native Environment: For local debugging, you can use the IDE's debug mode for debugging
- Docker Environment: Environment run by Maven for cloud compiled environments and testing ShardingSphere-Proxy scenarios, e.g. GitHub Action

The current default Docker environment, involving ShardingSphere-Proxy, Zookeeper, database instances (MySQL, PostgreSQL), are automatically started via Docker.

Database type currently supports MySQL, PostgreSQL, openGauss

9.8.3 Guide

Module path: shardingsphere-test/shardingsphere-integration-test/shardingsphere-integration-test-scaling

The Class distribution of the test is as follows.

Core Case: - MySQLGeneralScalingIT: Covered the most test scenarios, including part of the table migration, most variety of table fields, etc. - PostgreSQLGeneralScalingIT: Similar, except that the database type is PostgreSQL/openGauss and includes custom schema migration scenarios.

Primary Key Case:

- TextPrimaryKeyScalingIT: Support migration of tables with primary key of text type(e.g. UUID).

Configuration File

Catalog: resources/env/ - /common: The Dist SQL used in the Scaling process. - /{SQL-TYPE}: database-level configuration files. - /scenario: The configuration file for the test scenario, mainly SQL, may be written differently for different databases. - it-env.properties: Stores the corresponding configuration information.

Run Test Cases

All property values can be dynamically injected by means of the Maven command line -D.

`${image-name}` Indicates a legal docker image name, e.g., mysql:5.7, separated by commas if multiple.
`-Dscaling.it.docker.postgresql.version=${image-name}` Indicates the version of PostgreSQL that needs to be tested.
`-Dscaling.it.docker.mysql.version=${image-name}` Indicates the version of MySQL that needs to be tested.

Native Environment Startup

Native environments require that ShardingSphere-Proxy (and its own dependent Cluster, such as Zookeeper) and the database be started locally, and that ShardingSphere-Proxy's port be 3307, modify the properties in the it-env.properties file `scaling.it.env.type=native`. The database port can be configured in it-env.properties, or not if it is the default port.

The startup method is as follows: Find the Case you need to test, such as MySQLGeneralScalingIT, and configure the corresponding VM Option before startup, add the following configuration.

```
-Dscaling.it.env.type=native -Dscaling.it.docker.mysql.version=${image-name}
```

Just start it under the IDE using the Junit.

Docker Environment Startup

Step 1: Packaging Image

```
./mvnw -B clean install -am -pl shardingsphere-test/shardingsphere-integration-test/shardingsphere-integration-test-scaling -Pit.env.docker -DskipTests
```

Running the above command will build a Docker image `apache/shardingsphere-proxy-test:latest` for integration testing. If you have only modified the test code, you can reuse the existing test image without rebuilding it.

Docker environment configuration provides remote debugging port for ShardingSphere-Proxy, the default is 3308. You can change it yourself in `ShardingSphereProxyDockerContainer`.

Running Case

As with Native, only one parameter needs to be changed.

```
-Dscaling.it.env.type=docker
```

You can run the use case using the same IDE as Native, or you can run it using maven.

```
./mvnw -nsu -B install -f shardingsphere-test/shardingsphere-integration-test/shardingsphere-integration-test-scaling/pom.xml -Dscaling.it.env.type=DOCKER -Dscaling.it.docker.mysql.version=${image-name}
```

Attentions

The commands in the Scaling integration test are basically executed in the ShardingSphere-Proxy, so if they fail, most of them require a debug of the ShardingSphere-Proxy, and the logs prefixed with :Scaling-Proxy are output from the ShardingSphere-Proxy container.

10

Reference

This chapter contains a section of technical implementation with Apache ShardingSphere, which provide the reference with users and developers.

10.1 DistSQL

This chapter will introduce the detailed syntax of DistSQL.

10.1.1 Syntax

This chapter describes the syntax of DistSQL in detail, and introduces use of DistSQL with practical examples.

RDL Syntax

RDL (Resource & Rule Definition Language) responsible for definition of resources/rules.

Resource Definition

This chapter describes the syntax of resource definition.

ADD RESOURCE

Description

The ADD RESOURCE syntax is used to add resources for the currently selected database.

Syntax

```

AddResource ::=

  'ADD' 'RESOURCE' dataSource (',' dataSource)*

dataSource ::=

  dataSourceName '(' ('HOST' '=' hostName , 'PORT' '=' port , 'DB' '=' dbName
| 'URL' '=' url ) ',' 'USER' '=' user (',' 'PASSWORD' '=' password )? (','
'PROPERTIES' '(' (key '=' value) (',' key '=' value)* ')')? ')'

dataSourceName ::=

  identifier

hostname ::=

  identifier | ip

dbName ::=

  identifier

port ::=

  int

password ::=

  identifier | int | string

user ::=

  identifier

url ::=

  identifier | string

```

Supplement - Before adding resources, please confirm that a database has been created in Proxy, and execute the use command to successfully select a database - Confirm that the added resource can be connected normally, otherwise it will not be added successfully - dataSourceName is case-sensitive - dataSourceName needs to be unique within the current database - dataSourceName name only allows letters, numbers and _, and must start with a letter - poolProperty is used to customize connection pool parameters, key must be the same as the connection pool parameter name, value supports int and String types - When password contains special characters, it is recommended to use the string form; for example, the string form of password@123 is "password@123"

Example - Add resource using standard mode

```

ADD RESOURCE ds_0 (
  HOST=127.0.0.1,
  PORT=3306,
  DB=db_0,
  USER=root,
  PASSWORD=root

```

```
);
```

- Add resource and set connection pool parameters using standard mode

```
ADD RESOURCE ds_1 (
    HOST=127.0.0.1,
    PORT=3306,
    DB=db_1,
    USER=root,
    PASSWORD=root
    PROPERTIES("maximumPoolSize"=10)
);
```

- Add resource and set connection pool parameters using URL patterns

```
ADD RESOURCE ds_2 (
    URL="jdbc:mysql://127.0.0.1:3306/db_2?serverTimezone=UTC&useSSL=false",
    USER=root,
    PASSWORD=root,
    PROPERTIES("maximumPoolSize"=10,"idleTimeout"="30000")
);
```

Reserved word

ADD, RESOURCE, HOST, PORT, DB, USER, PASSWORD, PROPERTIES, URL

Related links - [Reserved word](#)

ALTER RESOURCE

Description

The ALTER RESOURCE syntax is used to alter resources for the currently selected database.

Syntax

```
AlterResource ::=

'ALTER' 'RESOURCE' dataSource (',' dataSource)*

dataSource ::=

dataSourceName '(' ('HOST' '=' hostName ',' 'PORT' '=' port ',' 'DB' '=' dbName
| 'URL' '=' url ) ',' 'USER' '=' user (',' 'PASSWORD' '=' password )? ',' '
'PROPERTIES' '(' (key '=' value) (',' key '=' value)* ')' )? ')'

dataSourceName ::=
```

```

identifier

hostname ::=

    identifier | ip

dbName ::=

    identifier

port ::=

    int

password ::=

    identifier | int | string

user ::=

    identifier

url ::=

    identifier | string

```

Supplement - Before altering the resources, please confirm that a database exists in Proxy, and execute the use command to successfully select a database - dataSourceName is case-sensitive - dataSourceName needs to be unique within the current database - dataSourceName name only allows letters, numbers and _, and must start with a letter - poolProperty is used to customize connection pool parameters, key must be the same as the connection pool parameter name, value supports int and String types - When password contains special characters, it is recommended to use the string form; for example, the string form of password@123 is "password@123"

Example - Alter resource using standard mode

```

ALTER RESOURCE ds_0 (
    HOST=127.0.0.1,
    PORT=3306,
    DB=db_0,
    USER=root,
    PASSWORD=root
);

```

- Alter resource and set connection pool parameters using standard mode

```

ALTER RESOURCE ds_1 (
    HOST=127.0.0.1,
    PORT=3306,
    DB=db_1,
    USER=root,
    PASSWORD=root
    PROPERTIES("maximumPoolSize"=10)
);

```

- Alter resource and set connection pool parameters using URL patterns

```
ALTER RESOURCE ds_2 (
    URL="jdbc:mysql://127.0.0.1:3306/db_2?serverTimezone=UTC&useSSL=false",
    USER=root,
    PASSWORD=root,
    PROPERTIES("maximumPoolSize"=10,"idleTimeout"="30000")
);
```

Reserved word

ALTER, RESOURCE, HOST, PORT, DB, USER, PASSWORD, PROPERTIES, URL

Related links - [Reserved word](#)

DROP RESOURCE

Description

The DROP RESOURCE syntax is used to drop resources from the current database

Syntax

```
DropResource ::=  
    'DROP' 'RESOURCE' ( 'IF' 'EXISTS' )? dataSourceName ( ',' dataSourceName )* (  
    'IGNORE' 'SINGLE' 'TABLES' )?
```

Supplement

- DROP RESOURCE will only drop resources in Proxy, the real data source corresponding to the resource will not be dropped
- Unable to drop resources already used by rules. Resources are still in used. will be prompted when removing resources used by rules
- The resource need to be removed only contains SINGLE TABLE RULE, and when the user confirms that this restriction can be ignored, the IGNORE SINGLE TABLES keyword can be added to remove the resource ### Example
- Drop a resource

```
DROP RESOURCE ds_0;
```

- Drop multiple resources

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

- Ignore single table rule remove resource

```
DROP RESOURCE ds_1 IGNORE SINGLE TABLES;
```

- Drop the resource if it exists

```
DROP RESOURCE IF EXISTS ds_2;
```

Reserved word

```
DROP, RESOURCE, IF, EXISTS, IGNORE, SINGLE, TABLES
```

Related links

- Reserved word

Rule Definition

This chapter describes the syntax of rule definition.

Sharding

This chapter describes the syntax of sharding.

CREATE SHARDING ALGORITHM

Description

The CREATE SHARDING ALGORITHM syntax is used to create a sharding algorithm for the currently selected database.

Syntax

```
CreateShardingAlgorithm ::=  
  'CREATE' 'SHARDING' 'ALGORITHM' shardingAlgorithmName '(' algorithmDefinition ')'  
  
algorithmDefinition ::=  
  'TYPE' '(' 'NAME' '=' algorithmType ( ',' 'PROPERTIES' '(' propertyDefinition ')' )? ')'
```

```

propertyDefinition ::=

  ( key '=' value ) ( ',' key '=' value )*

shardingAlgorithmName ::=

  identifier

algorithmType ::=

  identifier

```

Supplement

- algorithmType is the sharding algorithm type. For detailed sharding algorithm type information, please refer to [Sharding Algorithm](#)

Example

1.Create sharding algorithms

```

-- create a sharding algorithm of type INLINE
CREATE SHARDING ALGORITHM inline_algorithm (
    TYPE(NAME=inline, PROPERTIES("algorithm-expression"="t_order_${user_id % 2}"))
);

-- create a sharding algorithm of type AUTO_INTERVAL
CREATE SHARDING ALGORITHM interval_algorithm (
    TYPE(NAME=auto_interval, PROPERTIES("datetime-lower"="2022-01-01 00:00:00",
    "datetime-upper"="2022-01-03 00:00:00", "sharding-seconds"="86400"))
);

```

Reserved word

CREATE、SHARDING、ALGORITHM、TYPE、NAME、PROPERTIES

Related links

- Reserved word

CREATE SHARDING TABLE RULE

Description

The CREATE SHARDING TABLE RULE syntax is used to add sharding table rule for the currently selected database

Syntax

```

CreateShardingTableRule ::=

    'CREATE' 'SHARDING' 'TABLE' 'RULE' ( tableDefinition | autoTableDefinition ) ( ',' 
    ' ( tableDefinition | autoTableDefinition ) )*

tableDefinition ::=
    tableName '(' 'DATANODES' '(' dataNode ( ',' dataNode )* ')' )? ( ',' 'DATABASE_ 
    STRATEGY' '(' strategyDefinition ')' )? ( ',' 'TABLE_STRATEGY' '(' 
    strategyDefinition ')' )? ( ',' 'KEY_GENERATE_STRATEGY' '(' 
    keyGenerateStrategyDefinition ')' )? ')'

autoTableDefinition ::=
    tableName '(' 'RESOURCES' '(' resourceName ( ',' resourceName )* ')' ',' 
    'SHARDING_COLUMN' '=' columnName ',' algorithmDefinition ( ',' 'KEY_GENERATE_ 
    STRATEGY' '(' keyGenerateStrategyDefinition ')' )? ')'

strategyDefinition ::=
    'TYPE' '=' strategyType ',' ( 'SHARDING_COLUMN' | 'SHARDING_COLUMNS' ) '='
    columnName ',' algorithmDefinition

keyGenerateStrategyDefinition ::=
    'KEY_GENERATE_STRATEGY' '(' 'COLUMN' '=' columnName ',' ( 'KEY_GENERATOR' '='
    algorithmName | algorithmDefinition ) ')'

algorithmDefinition ::=
    ('SHARDING_ALGORITHM' '=' algorithmName | 'TYPE' '(' 'NAME' '=' algorithmType (
    ',' 'PROPERTIES' '(' propertyDefinition ')' ? ')' ) )

propertyDefinition ::=
    ( key '=' value ) ( ',' key '=' value ) *

tableName ::=
    identifier

resourceName ::=
    identifier

columnName ::=
    identifier

```

```
algorithmName ::=  
    identifier
```

Supplement

- `tableDefinition` is defined for standard sharding table rule; `autoTableDefinition` is defined for auto sharding table rule. For standard sharding rules and auto sharding rule, refer to [Data Sharding](#).
- use standard sharding table rule
 - DATANODES can only use resources that have been added to the current database, and can only use INLINE expressions to specify required resources
 - DATABASE_STRATEGY, TABLE_STRATEGY are the database sharding strategy and the table sharding strategy, which are optional, and the default strategy is used when not configured
 - The attribute `TYPE` in `strategyDefinition` is used to specify the type of [Sharding Algorithm](#), currently only supports STANDARD, COMPLEX. Using COMPLEX requires specifying multiple sharding columns with SHARDING_COLUMNS.
- use auto sharding table rule
 - RESOURCES can only use resources that have been added to the current database, and the required resources can be specified by enumeration or INLINE expression
 - Only auto sharding algorithm can be used, please refer to [Auto Sharding Algorithm](#)
- `algorithmType` is the sharding algorithm type, please refer to [Sharding Algorithm](#)
- The auto-generated algorithm naming rule is `tableName_strategyType_shardingAlgorithmType`
- The auto-generated primary key strategy naming rule is `tableName_strategyType`
- `KEY_GENERATE_STRATEGY` is used to specify the primary key generation strategy, which is optional. For the primary key generation strategy, please refer to [Distributed Primary Key](#)

Example

1. Standard sharding table rule

- **Create standard sharding table rule by specifying sharding algorithms**

```
-- create sharding algorithms
CREATE SHARDING ALGORITHM database_inline (
    TYPE(NAME=inline, PROPERTIES("algorithm-expression"="t_order_${user_id % 2}"))
), table_inline (
    TYPE(NAME=inline, PROPERTIES("algorithm-expression"="t_order_${order_id % 2}"))
```

```

);

-- create a sharding rule by specifying sharding algorithms
CREATE SHARDING TABLE RULE t_order (
    DATANODES("ds_${0..1}.t_order_${0..1}"),
    DATABASE_STRATEGY(TYPE=standard, SHARDING_COLUMN=user_id, SHARDING_
ALGORITHM=database_inline),
    TABLE_STRATEGY(TYPE=standard, SHARDING_COLUMN=order_id, SHARDING_
ALGORITHM=table_inline)
);

```

- Use the default sharding database strategy, create standard sharding table rule by specifying a sharding algorithm

```

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

-- create a default sharding database strategy
CREATE DEFAULT SHARDING DATABASE STRATEGY (
    TYPE = standard, SHARDING_COLUMN=order_id, SHARDING_ALGORITHM=database_inline
);

-- create a sharding table rule by specifying a sharding algorithm
CREATE SHARDING TABLE RULE t_order (
    DATANODES("ds_${0..1}.t_order_${0..1}"),
    TABLE_STRATEGY(TYPE=standard, SHARDING_COLUMN=order_id, SHARDING_
ALGORITHM=table_inline)
);

```

- Use both the default sharding and the default sharding strategy, create standard sharding table rule

```

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

-- create a default sharding database strategy
CREATE DEFAULT SHARDING DATABASE STRATEGY (

```

```

    TYPE = standard, SHARDING_COLUMN=order_id, SHARDING_ALGORITHM=database_inline
);

-- create a default sharding table strategy
CREATE DEFAULT SHARDING TABLE STRATEGY (
    TYPE = standard, SHARDING_COLUMN=order_id, SHARDING_ALGORITHM=table_inline
);

-- create a sharding table rule
CREATE SHARDING TABLE RULE t_order (
    DATANODES("ds_${0..1}.t_order_${0..1}")
);

```

- Create standard sharding table rule and sharding algorithms at the same time

```

CREATE SHARDING TABLE RULE t_order (
    DATANODES("ds_${0..1}.t_order_${0..1}"),
    DATABASE_STRATEGY(TYPE=standard, SHARDING_COLUMN=user_id, SHARDING_
ALGORITHM(TYPE(NAME=inline, PROPERTIES("algorithm-expression"="ds_${user_id % 2}
"))),
    TABLE_STRATEGY(TYPE=standard, SHARDING_COLUMN=user_id, SHARDING_
ALGORITHM(TYPE(NAME=inline, PROPERTIES("algorithm-expression"="ds_${order_id % 2}
"))))
);

```

2.Auto sharding table rule

- create auto sharding table rule

```

CREATE SHARDING TABLE RULE t_order (
    RESOURCES(ds_0, ds_1),
    SHARDING_COLUMN=order_id, TYPE(NAME=MOD, PROPERTIES("sharding-count"=4))
);

```

Reserved word

```

CREATE, SHARDING, TABLE, RULE, DATANODES, DATABASE_STRATEGY, TABLE_STRATEGY, KEY_
GENERATE_STRATEGY, RESOURCES, SHARDING_COLUMN, TYPE, SHARDING_COLUMN, KEY_
GENERATOR, SHARDING_ALGORITHM, COLUMN, NAME, PROPERTIES

```

Related links

- Reserved word
- CREATE SHARDING ALGORITHM
- CREATE DEFAULT_SHARDING STRATEGY

CREATE DEFAULT SHARDING STRATEGY

Description

The CREATE DEFAULT SHARDING STRATEGY syntax is used to create a default sharding strategy

Syntax

```
CreateDefaultShardingStrategy ::=  
    'CREATE' 'DEFAULT' 'SHARDING' ('DATABASE' | 'TABLE') 'STRATEGY' '('  
        shardingStrategy ')'  
  
shardingStrategy ::=  
    'TYPE' '=' strategyType ',' ('SHARDING_COLUMN' '=' columnName | 'SHARDING_  
    COLUMNS' '=' columnNames) ',' ('SHARDING_ALGORITHM' '=' algorithmName |  
    algorithmDefinition )  
  
algorithmDefinition ::=  
    'TYPE' '(' 'NAME' '=' algorithmType ( ',' 'PROPERTIES' '(' propertyDefinition  
    ')')? ')'  
  
columnNames ::=  
    columnName ( ',' columnName )+  
  
columnName ::=  
    identifier  
  
algorithmName ::=  
    identifier  
  
algorithmType ::=  
    identifier
```

Supplement

- When using the complex sharding algorithm, multiple sharding columns need to be specified using SHARDING_COLUMNS
- algorithmType is the sharding algorithm type. For detailed sharding algorithm type information, please refer to [Sharding Algorithm](#)

Example

1.Create a default sharding strategy by using an existing sharding algorithm

```
-- create a sharding algorithm
CREATE SHARDING ALGORITHM database_inline (
    TYPE(NAME=inline, PROPERTIES("algorithm-expression"="t_order_${order_id % 2}"))
);

-- create a default sharding database strategy
CREATE DEFAULT SHARDING DATABASE STRATEGY (
    TYPE=standard, SHARDING_COLUMN=user_id, SHARDING_ALGORITHM=database_inline
);
```

2.Create sharding algorithm and default sharding table strategy at the same time

```
-- create a default sharding table strategy
CREATE DEFAULT SHARDING TABLE STRATEGY (
    TYPE=standard, SHARDING_COLUMN=user_id, SHARDING_ALGORITHM(TYPE(NAME=inline,
PROPERTIES("algorithm-expression"="t_order_${user_id % 2}")))
);
```

Related links

- [CREATE SHARDING ALGORITHM](#)

CREATE SHARDING BINDING TABLE RULE

Description

The CREATE SHARDING BINDING TABLE RULE syntax is used to add binding relationships and create binding table rules for tables with sharding table rules

Syntax

```

CreateBindingTableRule ::=

  'CREATE' 'SHARDING' 'BINDING' 'TABLE' 'RULES' bindingTableDefinition (','
bindingTableDefinition )*

bindingTableDefinition ::=

  '(' tableName (',' tableName)* ')'

tableName ::=

  identifier

```

Supplement

- Creating binding relationships rules can only use sharding tables
- A sharding table can only have one binding relationships
- The sharding table for creating binding relationships needs to use the same resources and the same actual tables. For example `ds_{0..1}.t_order_{0..1}` 与 `ds_{0..1}.t_order_item_{0..1}`
- The sharding table for creating binding relationships needs to use the same sharding algorithm for the sharding column. For example `t_order_{order_id % 2}` 和 `t_order_item_{order_item_id % 2}`
- Only one binding rule can exist, but can contain multiple binding relationships, so can not execute `CREATE SHARDING BINDING TABLE RULE` more than one time. When a binding table rule already exists but a binding relationship needs to be added, you need to use `ALTER SHARDING BINDING TABLE RULE` to modify the binding table rule

Example

1.Create a binding table rule

```

-- Before creating a binding table rule, you need to create sharding table rules t_order, t_order_item
CREATE SHARDING BINDING TABLE RULES (t_order,t_order_item);

```

2.Create multiple binding table rules

```
-- Before creating binding table rules, you need to create sharding table rules t_order, t_order_item, t_product, t_product_item
CREATE SHARDING BINDING TABLE RULES (t_order,t_order_item),(t_product,t_product_item);
```

Related links

- [CREATE SHARDING TABLE RULE](#)

CREATE SHARDING BROADCAST TABLE RULE

Description

The CREATE SHARDING BROADCAST TABLE RULE syntax is used to create broadcast table rules for tables that need to be broadcast (broadcast tables)

Syntax

```
CreateBroadcastTableRule ::=

  'CREATE' 'SHARDING' 'BROADCAST' 'TABLE' 'RULES' '(' tableName (',' tableName)* ')'
  '

tableName ::=

  identifier
```

Supplement

- tableName can use an existing table or a table that will be created
- Only one broadcast rule can exist, but can contain multiple broadcast tables, so can not execute CREATE SHARDING BROADCAST TABLE RULE more than one time. When the broadcast table rule already exists but the broadcast table needs to be added, you need to use ALTER BROADCAST TABLE RULE to modify the broadcast table rule

Example

Create sharding broadcast table rule

```
-- Add t_province, t_city to broadcast table rules
CREATE SHARDING BROADCAST TABLE RULES (t_province, t_city);
```

CREATE SHARDING KEY GENERATOR

Description

The CREATE SHARDING KEY GENERATOR syntax is used to add a distributed primary key generator for the currently selected logic database

Syntax

```
CreateShardingAlgorithm ::=

  'CREATE' 'SHARDING' 'KEY' 'GENERATOR' keyGeneratorName '(' algorithmDefinition ')'
  '

algorithmDefinition ::=

  'TYPE' '(' 'NAME' '=' algorithmType ( ',' 'PROPERTIES' '(' propertyDefinition ')')? ')'

propertyDefinition ::=

  ( key '=' value ) ( ',' key '=' value )*

keyGeneratorName ::=

  identifier

algorithmType ::=

  identifier
```

Supplement

- algorithmType is the key generate algorithm type. For detailed key generate algorithm type information, please refer to [KEY GENERATE ALGORITHM](#)

Example

Create a distributed primary key generator

```
CREATE SHARDING KEY GENERATOR snowflake_key_generator (
    TYPE(NAME=SNOWFLAKE, PROPERTIES("max-vibration-offset"=3))
);
```

Reserved word

CREATE、SHARDING、KEY、GENERATOR、TYPE、NAME、PROPERTIES

Related links

- Reserved word

RQL Syntax

RQL (Resource & Rule Query Language) responsible for resources/rules query.

Resource Query

This chapter describes the syntax of resource query.

SHOW RESOURCE

Description

The SHOW RESOURCE syntax is used to query the resources that have been added to the specified database.

Syntax

```
ShowResource ::=  
    'SHOW' 'DATABASE' 'RESOURCES' ('FROM' databaseName)?  
  
databaseName ::=  
    identifier
```

Supplement

- When `databaseName` is not specified, the default is the currently used `DATABASE`; if `DATABASE` is not used, it will prompt `No database selected.`

Return Value Description

Column	Description
<code>name</code>	Data source name
<code>type</code>	Data source type
<code>host</code>	Data source host
<code>port</code>	Data source port
<code>db</code>	Database name
<code>attribute</code>	Data source attribute

Example

- Query resources for the specified database

```
SHOW DATABASE RESOURCES FROM sharding_db;
```

```
+-----+-----+-----+-----+-----+
| name | type  | host      | port | db    | connection_timeout_milliseconds | idle_
timeout_milliseconds | max_lifetime_milliseconds | max_pool_size | min_pool_size |
read_only | other_attributes
+-----+-----+-----+-----+-----+
```

```
| |
+-----+-----+-----+-----+-----+
| |
+-----+-----+-----+-----+
```

- Query resources for the current database

```
SHOW DATABASE RESOURCES;
```

name	type	host	port	db	connection_timeout_milliseconds	idle_timeout_milliseconds	max_lifetime_milliseconds	max_pool_size	min_pool_size	read_only	other_attributes
------	------	------	------	----	---------------------------------	---------------------------	---------------------------	---------------	---------------	-----------	------------------

ds_0	MySQL	127.0.0.1	3306	db_0	30000 1800000	50	1			60000
ds_1	MySQL	127.0.0.1	3306	db_1	30000 1800000	50	1			60000

```
2 rows in set (0.26 sec)
```

SHOW UNUSED RESOURCE

Description

The SHOW UNUSED RESOURCE syntax is used to query resources in the specified database that have not been referenced by rules.

Syntax

```
ShowUnusedResource ::=  
    'SHOW' 'UNUSED' 'DATABASE'? 'RESOURCES' ('FROM' databaseName)?  
  
databaseName ::=  
    identifier
```

Supplement

- When databaseName is not specified, the default is the currently used DATABASE; if DATABASE is not used, it will prompt No database selected.

Return Value Description

Column	Description
name	Data source name
type	Data source type
host	Data source host
port	Data source port
db	Database name
attribute	Data source attribute

Example

- Query resources for the specified database

```
SHOW UNUSED DATABASE RESOURCES FROM sharding_db;
```

```
+-----+-----+-----+-----+-----+
-----+-----+-----+-----+
-----+
-----+
-----+
-----+
-----+
-----+
```

- Query resources for the current database

SHOW UNUSED DATABASE RESOURCES:

```
+-----+-----+-----+-----+-----+
| name | type   | host      | port | db    | connection_timeout_milliseconds | idle_
timeout_milliseconds | max_lifetime_milliseconds | max_pool_size | min_pool_size | 
read_only | other_attributes
+-----+-----+-----+-----+-----+
| ds_0 | MySQL | 127.0.0.1 | 3306 | db_0 | 30000          | 60000
|           | 1800000          | 50      | 1           |
false     | {"dataSourceProperties": {"cacheServerConfiguration": "true",
"elideSetAutoCommits": "true", "useServerPrepStmts": "true", "cachePrepStmts": "true",
"rewriteBatchedStatements": "true", "cacheResultSetMetadata": "false",
"useLocalSessionState": "true", "maintainTimeStats": "false", "prepStmtCacheSize": "2000000",
"tinyInt1isBit": "false", "prepStmtCacheSqlLimit": "2048",
"zeroDateTimeBehavior": "round", "netTimeoutForStreamingResults": "0"}, "initializationFailTimeout": 1, "validationTimeout": 5000,
"leakDetectionThreshold": 0, "registerMbeans": false, "allowPoolSuspension": false,
"autoCommit": true, "isolateInternalQueries": false} |
+-----+-----+-----+-----+-----+
```

```
1 rows in set (0.26 sec)
```

SHOW RULES USED RESOURCE

Description

The SHOW RULES USED RESOURCE syntax is used to query the rules that use the specified resource in the specified database.

Syntax

```
showRulesUsedResource ::=  
    'SHOW' 'RULES' 'USED' 'RESOURCES' resourceName ('FROM' databaseName)?  
  
resourceName ::=  
    IDENTIFIER | STRING  
  
databaseName ::=  
    IDENTIFIER
```

Supplement

- When databaseName is not specified, the default is the currently used DATABASE; if DATABASE is not used, it will prompt No database selected.

Return Value Description

Column	Description
type	features
name	Data source name

Example

- Query the rules that use the specified resource in the specified database

```
SHOW RULES USED RESOURCE ds_0 FROM sharding_db;
```

```
+-----+-----+  
| type      | name      |  
+-----+-----+  
| sharding  | t_order   |
```

```
| sharding | t_order_item |
+-----+-----+
2 rows in set (0.00 sec)
```

- Query the rules that use the specified resource in the current database

```
SHOW RULES USED RESOURCE ds_0;
```

```
+-----+-----+
| type      | name      |
+-----+-----+
| sharding  | t_order   |
| sharding  | t_order_item |
+-----+-----+
2 rows in set (0.00 sec)
```

Rule Query

This chapter describes the syntax of rule query.

Sharding

This chapter describes the syntax of sharding.

SHOW SHARDING TABLE RULE

Description

The SHOW SHARDING TABLE RULE syntax is used to query the sharding table rule in the specified database.

Syntax

```
ShowShardingTableRule ::=  
    'SHOW' 'SHARDING' 'TABLE' ('RULE' tableName | 'RULES') ('FROM' databaseName)?  
  
tableName ::=  
    identifier  
  
databaseName ::=  
    identifier
```

Supplement

- When `databaseName` is not specified, the default is the currently used DATABASE. If DATABASE is not used, No database selected will be prompted.

Return value description

Column	Description
table	Logical table name
actual_data_nodes	Actual data node
actual_data_sources	Actual data source (Displayed when creating rules by RDL)
database_strategy_type	Database sharding strategy type
database_sharding_column	Database sharding column
database_ sharding_algorithm_type	Database sharding algorithm type
database_sharding_algorithm_props	Database sharding algorithm properties
table_strategy_type	Table sharding strategy type
table_sharding_column	Table sharding column
table_ sharding_algorithm_type	Table sharding algorithm type
table_sharding_algorithm_props	Table sharding algorithm properties
key_generate_column	Sharding key generator column
key_generator_type	Sharding key generator type
key_generator_props	Sharding key generator properties

Example - Query the sharding table rules of the specified logical database

```
SHOW SHARDING TABLE RULES FROM sharding_db;
```

```
| table      | actual_data_nodes | actual_data_sources | database_strategy_type |
database_sharding_column | database_sharding_algorithm_type | database_sharding_
algorithm_props | table_strategy_type | table_sharding_column | table_sharding_
algorithm_type | table_sharding_algorithm_props | key_generate_column | key_
generator_type | key_generator_props |
+-----+-----+-----+
+-----+-----+-----+
-----+-----+-----+
-----+-----+-----+
-----+-----+-----+
-----+-----+-----+
| t_order    |           | ds_0,ds_1          |           |
|           |           |                   |           |
| mod       |           | order_id          |           |
|           |           |                   | mod
| sharding-count=4 |           |           |           |
|           |           |           |           |
```

```

| t_order_item |           | ds_0,ds_1 |           |
|               | mod      | order_id | mod      |
|   sharding-count=4 |           |           |           |
+-----+-----+-----+
+-----+-----+-----+
+-----+-----+-----+
+-----+-----+-----+
2 rows in set (0.12 sec)

```

- Query the sharding table rules of the current logic database

```
SHOW SHARDING TABLE RULES;
```

```

+-----+-----+-----+
+-----+-----+-----+
+-----+-----+-----+
+-----+-----+-----+
+-----+-----+-----+
| table      | actual_data_nodes | actual_data_sources | database_strategy_type |
| database_sharding_column | database_sharding_algorithm_type | database_sharding_
| algorithm_props | table_strategy_type | table_sharding_column | table_sharding_
| algorithm_type | table_sharding_algorithm_props | key_generate_column | key_
| generator_type | key_generator_props |
+-----+-----+-----+
+-----+-----+-----+
+-----+-----+-----+
+-----+-----+-----+
+-----+-----+-----+
| t_order     |           | ds_0,ds_1 |           |
|               | mod      | order_id | mod      |
|   sharding-count=4 |           |           |           |
+-----+-----+-----+
| t_order_item |           | ds_0,ds_1 |           |
|               | mod      | order_id | mod      |
|   sharding-count=4 |           |           |           |
+-----+-----+-----+
+-----+-----+-----+
+-----+-----+-----+
+-----+-----+-----+
2 rows in set (0.12 sec)

```

- Query the specified sharding table rule

```
SHOW SHARDING TABLE RULE t_order;
```

```
+-----+-----+
+-----+-----+
+-----+-----+
+-----+-----+
| table      | actual_data_nodes | actual_data_sources | database_strategy_type |
database_sharding_column | database_sharding_algorithm_type | database_sharding_
algorithm_props | table_strategy_type | table_sharding_column | table_sharding_
algorithm_type | table_sharding_algorithm_props | key_generate_column | key_
generator_type | key_generator_props |
+-----+-----+
+-----+-----+
+-----+-----+
+-----+-----+
| t_order     |                   | ds_0,ds_1           |                   |
|             |                   |                   |                   |
| mod        |                   | order_id          | mod               |
| sharding-count=4 |                   |                   |                   |
|             |                   |                   |                   |
+-----+-----+
+-----+-----+
+-----+-----+
+-----+-----+
1 rows in set (0.12 sec)
```

RAL Syntax

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

Reserved word

Resource definition

```
ADD, RESOURCE, HOST, PORT, DB, USER, PASSWORD, PROPERTIES, URL
```

Supplement

- The above reserved words are not case-sensitive

10.2 Management

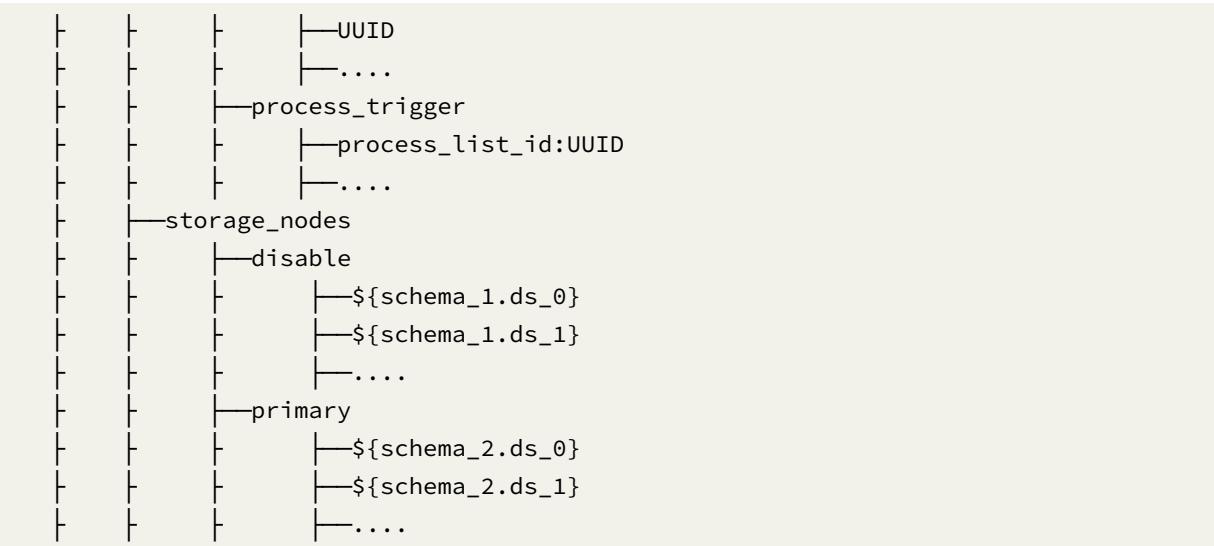
10.2.1 Data Structure in Registry Center

Under defined namespace, rules, props and metadata nodes persist in YAML, modifying nodes can dynamically refresh configurations. nodes node persist the runtime node of database access object, to distinguish different database access instances.

```

namespace
  |-rules                                # Global rule configuration
  |-props                                 # Properties configuration
  |-metadata
    |-${databaseName}
      |-schemas
        |-${schemaName}
          |-tables
            |-${tableName}
            |...
            ...
          |-versions
            |-${versionNumber}
              |-dataSources
              |-rules
              |...
            |-active_version      # Active metadata version
            ...
    ...
  |-nodes
    |-compute_nodes
      |-online
        |-proxy
          |-UUID
          |...
        |-jdbc
          |-UUID
          |...
      |-status
        |-UUID
        |...
      |-xa_recovery_id
        |-recovery_id
          |-UUID
          |...
      |-worker_id

```



/rules

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

```

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

/props

Properties configuration. Please refer to [Configuration Manual](#) for more details.

```

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

/metadata/*databaseName*/versions/{**versionNumber**}/dataSources

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

```

ds_0:
  initializationFailTimeout: 1
  validationTimeout: 5000
  maxLifetime: 1800000
  leakDetectionThreshold: 0
  minimumIdle: 1
  password: root
  
```

```

idleTimeout: 60000
jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds_0?serverTimezone=UTC&useSSL=false
dataSourceClassName: com.zaxxer.hikari.HikariDataSource
maximumPoolSize: 50
connectionTimeout: 30000
username: root
poolName: HikariPool-1

ds_1:
initializationFailTimeout: 1
validationTimeout: 5000
maxLifetime: 1800000
leakDetectionThreshold: 0
minimumIdle: 1
password: root
idleTimeout: 60000
jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds_1?serverTimezone=UTC&useSSL=false
dataSourceClassName: com.zaxxer.hikari.HikariDataSource
maximumPoolSize: 50
connectionTimeout: 30000
username: root
poolName: HikariPool-2

```

/metadata/*databaseName*/versions/{*versionNumber*}/rules

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

- !SHARDING
xxx
- !READWRITE_SPLITTING
xxx
- !ENCRYPT
xxx

/metadata/*databaseName*/schemas/{*schemaName*}/tables

Use separate node storage for each table, dynamic modification of metadata content is not supported currently.

```

name: t_order                      # Table name
columns:
  id:                                # Columns
    caseSensitive: false              # Column name
    dataType: 0

```

```

generated: false
name: id
primaryKey: true
order_id:
  caseSensitive: false
  dataType: 0
  generated: false
  name: order_id
  primaryKey: false
indexes:                                # Index
  t_user_order_id_index:                 # Index name
    name: t_user_order_id_index

```

/nodes/compute_nodes

It includes running instance information of database access object, with sub-nodes as the identifiers of currently running instance, which is automatically generated at each startup using UUID. Those identifiers are temporary nodes, which are registered when instances are on-line and cleared when instances are off-line. The registry center monitors the change of those nodes to govern the database access of running instances and other things.

/nodes/storage_nodes

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

10.3 Sharding

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

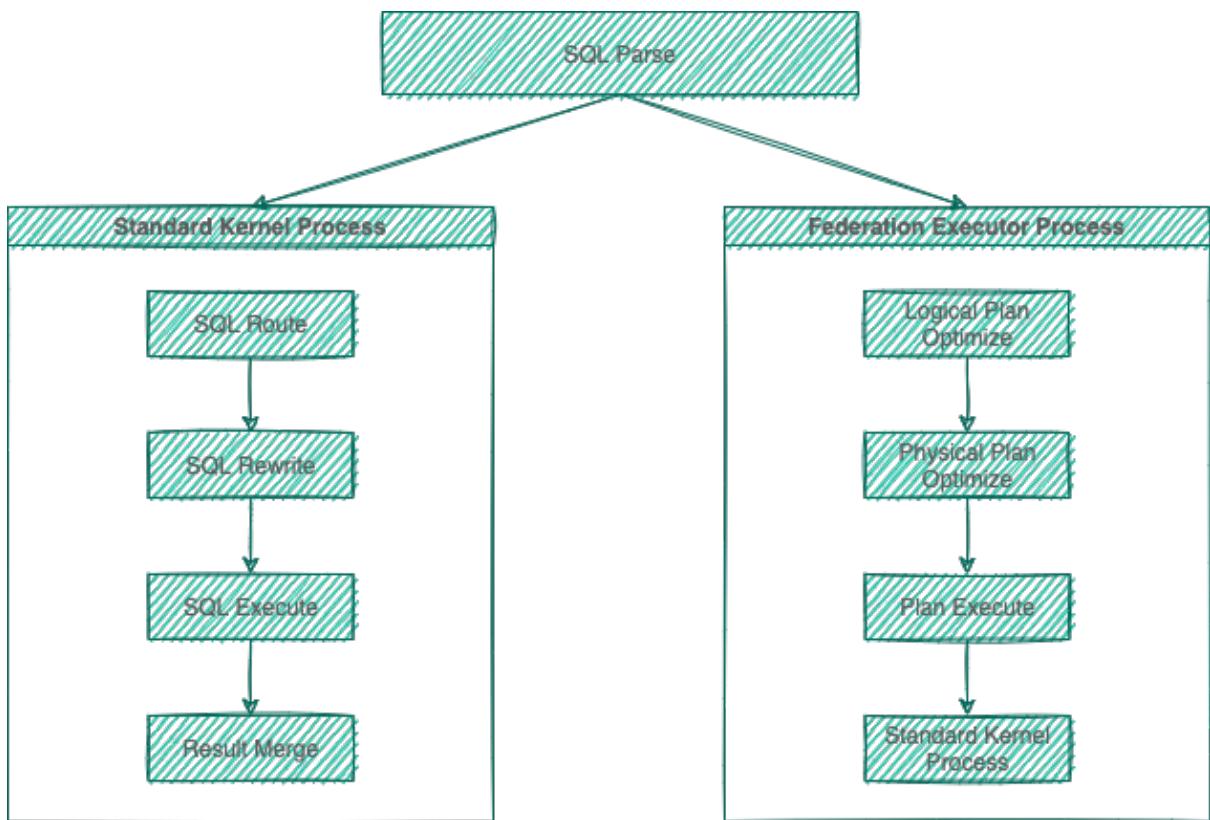


Figure1: Sharding Architecture Diagram

10.3.1 SQL Parsing

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

10.3.2 SQL Route

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

10.3.3 SQL Rewrite

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

10.3.4 SQL Execution

Through multi-thread executor, it executes asynchronously.

10.3.5 Result Merger

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

10.3.6 Query Optimization

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

10.3.7 Parse Engine

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

Abstract Syntax Tree

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

For example, the following SQL:

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

Its parsing AST (Abstract Syntax Tree) is this:

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

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

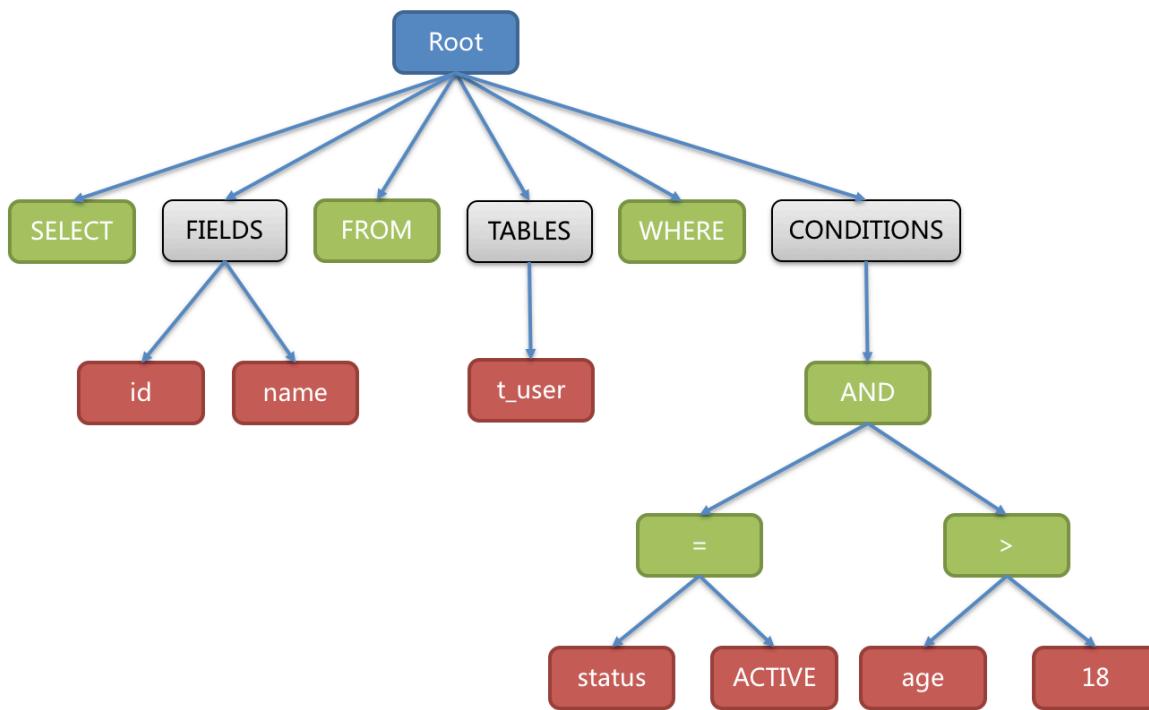


Figure2: SQL AST

SQL Parser

History

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

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

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

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

Features

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

Database	Status
MySQL	perfect supported
PostgreSQL	perfect supported
SQLServer	supported
Oracle	supported
SQL92	supported
openGauss	supported

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

API Usage

- Maven

```
<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-sql-parser-engine</artifactId>
    <version>${project.version}</version>
</dependency>
<!-- According to the needs, introduce the parsing module of the specified dialect
(take MySQL as an example), you can add all the supported dialects, or just what
you need --&gt;
&lt;dependency&gt;
    &lt;groupId&gt;org.apache.shardingsphere&lt;/groupId&gt;
    &lt;artifactId&gt;shardingsphere-sql-parser-mysql&lt;/artifactId&gt;
    &lt;version&gt;${project.version}&lt;/version&gt;
&lt;/dependency&gt;</pre>

```

- Get AST

```
CacheOption cacheOption = new CacheOption(128, 1024L);
SQLParserEngine parserEngine = new SQLParserEngine(sql, cacheOption);
ParseASTNode parseASTNode = parserEngine.parse(sql, useCache);
```

- GET SQLStatement

```
CacheOption cacheOption = new CacheOption(128, 1024L);
SQLParserEngine parserEngine = new SQLParserEngine(sql, cacheOption);
ParseASTNode parseASTNode = parserEngine.parse(sql, useCache);
```

```
SQLVisitorEngine sqlVisitorEngine = new SQLVisitorEngine(sql, "STATEMENT",
useCache, new Properties());
SQLStatement sqlStatement = sqlVisitorEngine.visit(parseASTNode);
```

- SQL Format

```
ParseASTNode parseASTNode = parserEngine.parse(sql, useCache);
SQLVisitorEngine sqlVisitorEngine = new SQLVisitorEngine(sql, "STATEMENT",
useCache, new Properties());
SQLStatement sqlStatement = sqlVisitorEngine.visit(parseASTNode);
```

example:

Original SQL	Formatted SQL
select a+1 as b, name n from table1 join table2 where id=1 and name= ‘lu’ ;	SELECT a + 1 AS b, name nFROM table1 JOIN table2WHERE id = 1 and name = ‘lu’ ;
select id, name, age, sex, ss, yy from table1 where id=1;	SELECT id , name , age , sex , ss , yy FROM 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 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 table1 WHERE 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 table1 WHERE 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_order WHERE order_id = 1;

10.3.8 Route Engine

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

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

Sharding Route

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

Direct Route

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

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

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

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

The route result will be:

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

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

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

Then, the route result will be:

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

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

Cartesian Route

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

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

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

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

Broadcast Route

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

Schema & Table Route

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

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

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

```
SELECT * FROM t_order_0 WHERE good_prority IN (1, 10);
SELECT * FROM t_order_1 WHERE good_prority IN (1, 10);
SELECT * FROM t_order_2 WHERE good_prority IN (1, 10);
SELECT * FROM t_order_3 WHERE good_prority IN (1, 10);
```

Database Schema Route

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

```
SET autocommit=0;
```

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

Database Instance Route

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

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

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

Unicast Route

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

```
DESCRIBE t_order;
```

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

Ignore Route

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

```
USE order_db;
```

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

The overall structure of route engine is as the following:

10.3.9 Rewrite Engine

The SQL written by engineers facing logic databases and tables cannot be executed directly in actual databases. SQL rewrite is used to rewrite logic SQL into rightly executable ones in actual databases, including two parts, correctness rewrite and optimization rewrite.

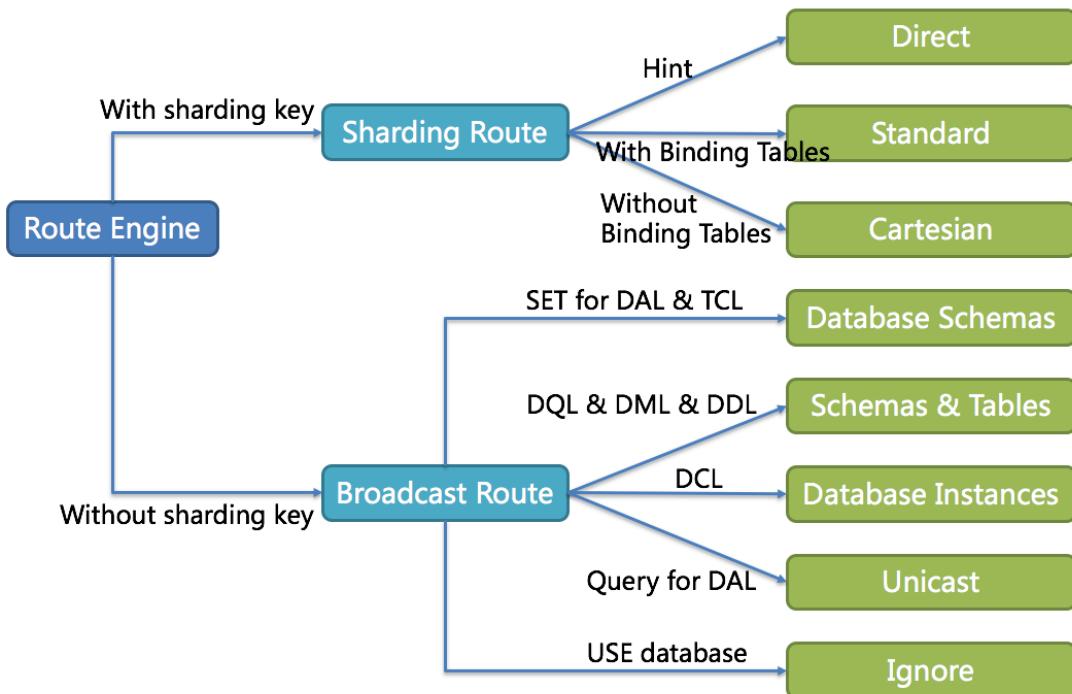


Figure3: Route Engine

Correctness Rewrite

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

Identifier Rewrite

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

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

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

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

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

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

The SQL rightly rewritten is supposed to be:

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

Rather than:

```
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:

```
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:

```
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:

```
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:

```
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:

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

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

Column Derivation

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

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

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

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

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

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

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

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

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

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

```
SELECT o.*, order_item_id AS ORDER_BY_DERIVED_0 FROM t_order o, t_order_item i WHERE o.order_id=i.order_id ORDER BY user_id, order_item_id;
```

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

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

Should be rewritten as:

```
SELECT COUNT(price) AS AVG_DERIVED_COUNT_0, SUM(price) AS AVG_DERIVED_SUM_0 FROM t_order WHERE user_id=1;
```

Then it can calculate the right average value through result merger.

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

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

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

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

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

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

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

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

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

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

Pagination Revision

The scenarios of acquiring pagination data from multiple databases is different from that of one single database. If every 10 pieces of data are taken as one page, the user wants to take the second page of data. It is not right to take, acquire LIMIT 10, 10 under sharding situations, and take out the first 10 pieces of data according to sorting conditions after merging. For example, if the SQL is:

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

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

As shown in the picture, if you want to acquire the second and the third piece of data ordered by score common in both tables, and they are supposed to be 95 and 90. Since the executed SQL can only acquire the second and the third piece of data from each table, i.e., 90 and 80 from t_score_0, 85 and 75 from

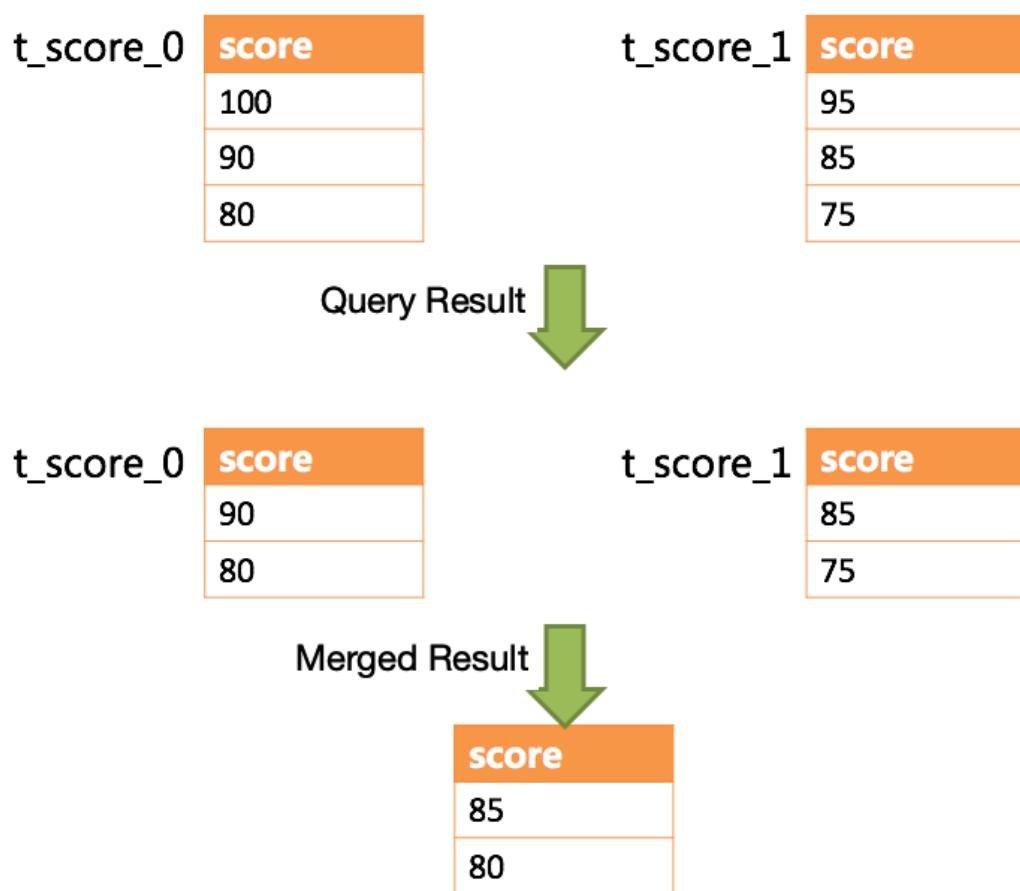


Figure4: Pagination without rewrite

t_score_1. When merging results, it can only merge from 90, 80, 85 and 75 already acquired, so the right result cannot be acquired anyway.

The right way is to rewrite pagination conditions as `LIMIT 0, 3`, take out all the data from the first two pages and combine sorting conditions to calculate the right data. The following picture shows the execution of pagination results after SQL rewrite.

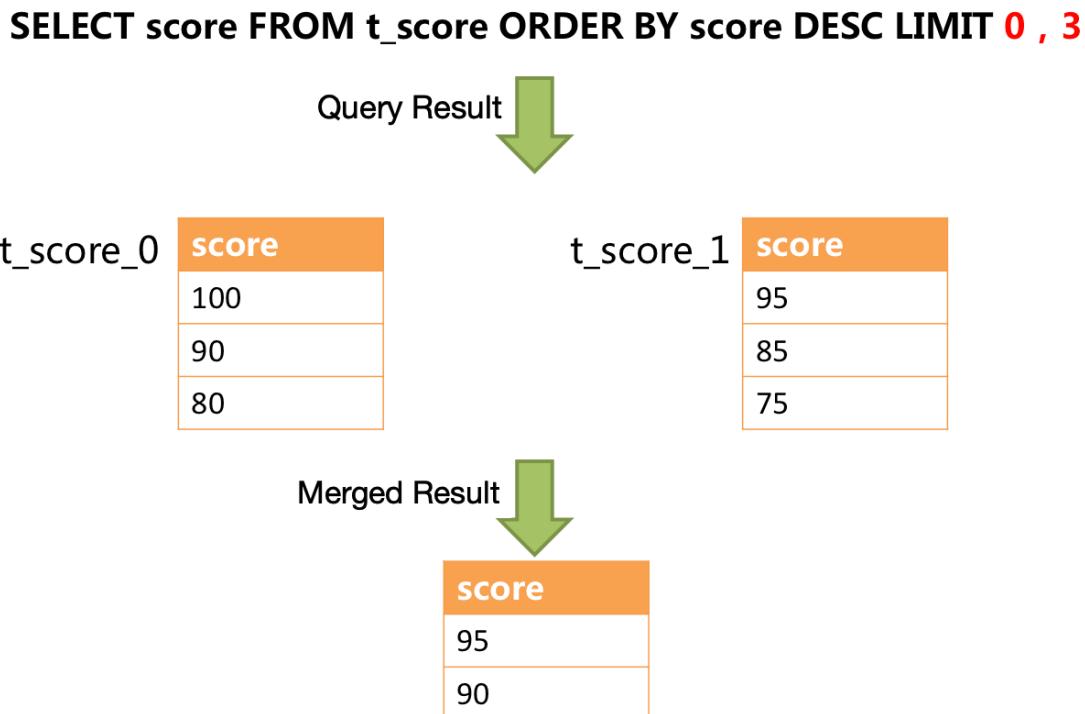


Figure5: Pagination with rewrite

The latter the offset position is, the lower the efficiency of using `LIMIT` pagination will be. There are many ways to avoid using `LIMIT` as pagination method, such as constructing a secondary index to record line record number and line offset amount, or using the tail ID of last pagination data as the pagination method of conditions of the next query.

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

Batch Split

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

```
INSERT INTO t_order (order_id, xxx) VALUES (1, 'xxx'), (2, 'xxx'), (3, 'xxx');
```

If the database is still divided into two parts according to odd and even number of order_id, this SQL will be executed after its table name is revised. Then, both shards will be written with the same record. Though only the data that satisfies sharding conditions can be taken out from query statement, it is not reasonable for the schema to have excessive data. So the SQL should be rewritten as:

```
INSERT INTO t_order_0 (order_id, xxx) VALUES (2, 'xxx');
INSERT INTO t_order_1 (order_id, xxx) VALUES (1, 'xxx'), (3, 'xxx');
```

IN query is similar to batch insertion, but IN operation will not lead to wrong data query result. Through rewriting IN query, the query performance can be further improved. Like the following SQL:

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

Is rewritten as:

```
SELECT * FROM t_order_0 WHERE order_id IN (2);
SELECT * FROM t_order_1 WHERE order_id IN (1, 3);
```

The query performance will be further improved. For now, ShardingSphere has not realized this rewrite strategy, so the current rewrite result is:

```
SELECT * FROM t_order_0 WHERE order_id IN (1, 2, 3);
SELECT * FROM t_order_1 WHERE order_id IN (1, 2, 3);
```

Though the execution result of SQL is right, but it has not achieved the most optimized query efficiency.

Optimization Rewrite

Its purpose is to effectively improve the performance without influencing the correctness of the query. It can be divided into single node optimization and stream merger optimization.

Single Node Optimization

It refers to the optimization that stops the SQL rewrite from the route to the single node. After acquiring one route result, if it is routed to a single data node, result merging is unnecessary to be involved, so there is no need for rewrites as derived column, pagination information and others. In particular, there is no need to read from the first piece of information, which reduces the pressure for the database to a large extent and saves meaningless consumption of the network bandwidth.

Stream Merger Optimization

It only adds sorting items and sorting orders identical with grouping items and ORDER BY to GROUP BY SQL, and they are used to transfer memory merger to stream merger. In the result merger part, stream merger and memory merger will be explained in detail.

The overall structure of rewrite engine is shown in the following picture.

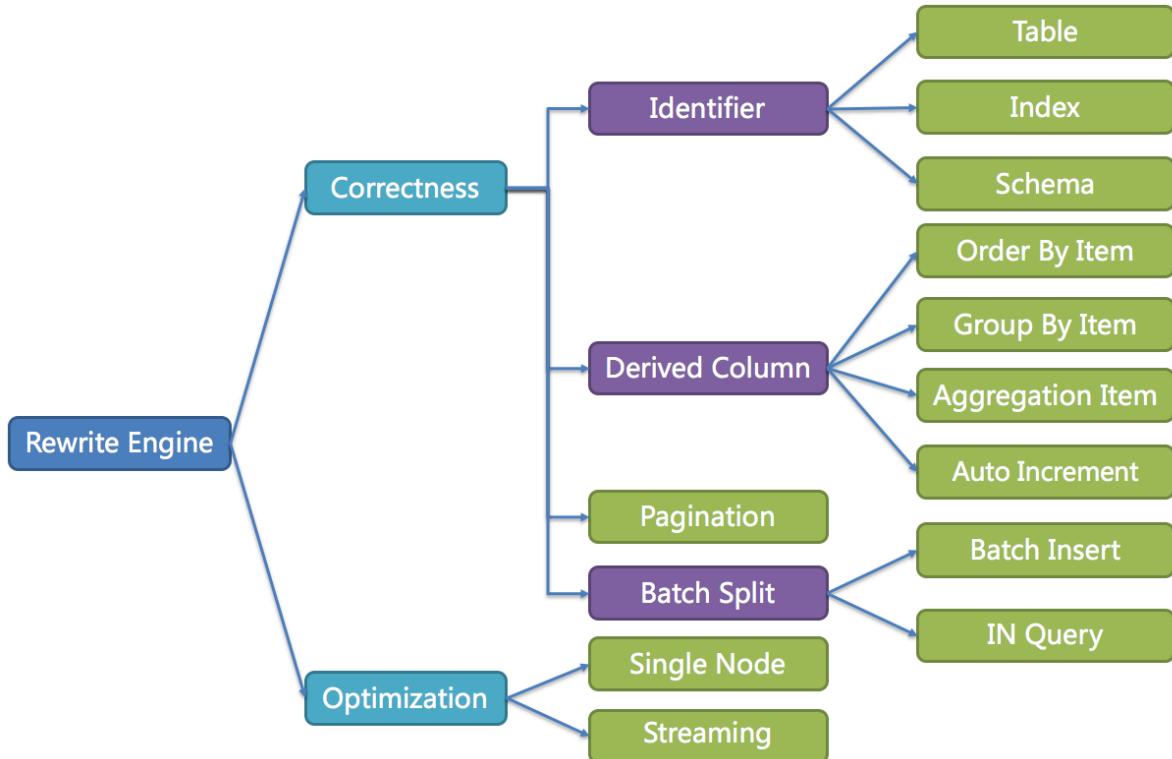


Figure6: Rewrite Engine

10.3.10 Execute Engine

ShardingSphere adopts a set of automatic execution engine, responsible for sending the true SQL, which has been routed and rewritten, to execute in the underlying data source safely and effectively. It does not simply send the SQL through JDBC to directly execute in the underlying data source, or put execution requests directly to the thread pool to concurrently execute, but focuses more on the creation of a balanced data source connection, the consumption generated by the memory usage, the maximum utilization of the concurrency and other problems. The objective of the execution engine is to automatically balance between the resource control and the execution efficiency.

Connection Mode

From the perspective of resource control, the connection number of the business side's visit of the database should be limited. It can effectively prevent some certain business from occupying excessive resource, exhausting database connection resources and influencing the normal use of other businesses. Especially when one database contains many tables, a logic SQL that does not contain any sharding key will produce a large amount of physical SQLs that fall into different tables in one database. If each physical SQL takes an independent connection, a query will undoubtedly take up excessive resources.

From the perspective of execution efficiency, holding an independent database connection for each sharding query can make effective use of multi-thread to improve execution efficiency. Opening an independent thread for each database connection can parallelize IO produced consumption. Holding an independent database connection for each sharding query can also avoid loading the query result to the memory too early. It is enough for independent database connections to maintain result set quotation and cursor position, and move the cursor when acquiring corresponding data.

Merging result set by moving down its cursor is called stream merger. It does not require to load all the query results to the memory. Thus, it is able to save memory resource effectively and reduce trash recycle frequency. When it is not able to make sure each sharding query holds an independent database connection, it requires to load all the current query results to the memory before reusing that database connection to acquire the query result from the next sharding table. Therefore, though the stream merger can be used, under this kind of circumstances, it will also degenerate to the memory merger.

The control and protection of database connection resources is one thing, adopting better merging model to save the memory resources of middleware is another thing. How to deal with the relationship between them is a problem that ShardingSphere execution engine should solve. To be accurate, if a sharding SQL needs to operate 200 tables under some database case, should we choose to create 200 parallel connection executions or a serial connection execution? Or to say, how to choose between efficiency and resource control?

Aiming at the above situation, ShardingSphere has provided a solution. It has put forward a Connection Mode concept divided into two types, MEMORY_STRICTLY mode and CONNECTION_STRICTLY mode.

MEMORY_STRICTLY Mode

The prerequisite to use this mode is that ShardingSphere does not restrict the connection number of one operation. If the actual executed SQL needs to operate 200 tables in some database instance, it will create a new database connection for each table and deal with them concurrently through multi-thread to maximize the execution efficiency. When the SQL is up to standard, it will choose stream merger in priority to avoid memory overflow or frequent garbage recycle.

CONNECTION_STRICTLY Mode

The prerequisite to use this mode is that ShardingSphere strictly restricts the connection consumption number of one operation. If the SQL to be executed needs to operate 200 tables in database instance, it will create one database connection and operate them serially. If shards exist in different databases, it will still be multi-thread operations for different databases, but with only one database connection being created for each operation in each database. It can prevent the problem brought by excessive occupation of database connection from one request. The mode chooses memory merger all the time.

The MEMORY_STRICTLY mode is applicable to OLAP operation and can increase the system capacity by removing database connection restrictions. It is also applicable to OLTP operation, which usually has sharding keys and can be routed to a single shard. So it is a wise choice to control database connection strictly to make sure resources of online system databases can be used by more applications.

Automatic Execution Engine

ShardingSphere uses which mode at first is up to users' setting and they can choose to use MEMORY_STRICTLY mode or CONNECTION_STRICTLY mode according to their actual business scenarios.

The solution gives users the right to choose, requiring them to know the advantages and disadvantages of both modes and make decision according to the actual business situations. No doubt, it is not the best solution due to increasing users' study cost and use cost.

This kind of dichotomy solution lacks flexible coping ability to switch between two modes with static initialization. In practical situations, route results of each time may differ with different SQL and placeholder indexes. It means some operations may need to use memory merger, while others are better to use stream merger. Connection modes should not be set by users before initializing ShardingSphere, but should be decided dynamically by the situation of SQL and placeholder indexes.

To reduce users' use cost and solve the dynamic connection mode problem, ShardingSphere has extracted the thought of automatic execution engine in order to eliminate the connection mode concept inside. Users do not need to know what are so called MEMORY_STRICTLY mode and CONNECTION_STRICTLY mode, but let the execution engine to choose the best solution according to current situations.

Automatic execution engine has narrowed the selection scale of connection mode to each SQL operation. Aiming at each SQL request, automatic execution engine will do real-time calculations and evaluations according to its route result and execute the appropriate connection mode automatically to strike the most optimized balance between resource control and efficiency. For automatic execution engine,

users only need to configure `maxConnectionSizePerQuery`, which represents the maximum connection number allowed by each database for one query.

The execution engine can be divided into two phases: preparation and execution.

Preparation Phrase

As indicated by its name, this phrase is used to prepare the data to be executed. It can be divided into two steps: result set grouping and unit creation.

Result set grouping is the key to realize the internal connection model concept. According to the configuration option of `maxConnectionSizePerQuery`, execution engine will choose an appropriate connection mode combined with current route result.

Detailed steps are as follow:

1. Group SQL route results according to data source names.
2. Through the equation in the following picture, users can acquire the SQL route result group to be executed by each database case within the `maxConnectionSizePerQuery` permission range and calculate the most optimized connection mode of this request.

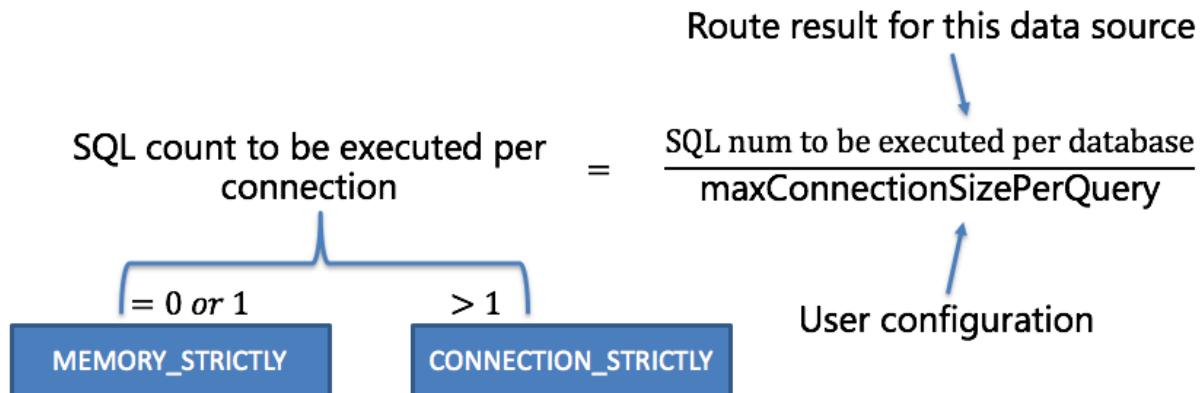


Figure7: Connection mode calculate formula

Within the range that `maxConnectionSizePerQuery` permits, when the request number that one connection needs to execute is more than 1, meaning current database connection cannot hold the corresponding data result set, it must uses memory merger. On the contrary, when it equals to 1, meaning current database connection can hold the according data result set, it can use stream merger.

Each choice of connection mode aims at each physical database; that is to say, if it is routed to more than one databases, the connection mode of each database may mix with each other and not be the same in one query.

Users can use the route group result acquired from the last step to create the execution unit. When the data source uses technologies, such as database connection pool, to control database connection number, there is some chance for deadlock, if it has not dealt with concurrency properly. As multiple

requests waiting for each other to release database connection resources, it will generate hunger wait and cause the crossing deadlock problem.

For example, suppose one query needs to acquire two database connections from a data source and apply them in two table sharding queries routed to one database. It is possible that Query A has already acquired a database connection from that data source and waits to acquire another connection; but in the same time, Query B has also finished it and waits. If the maximum connection number that the connection pool permits is 2, those two query requests will wait forever. The following picture has illustrated the deadlock situation:

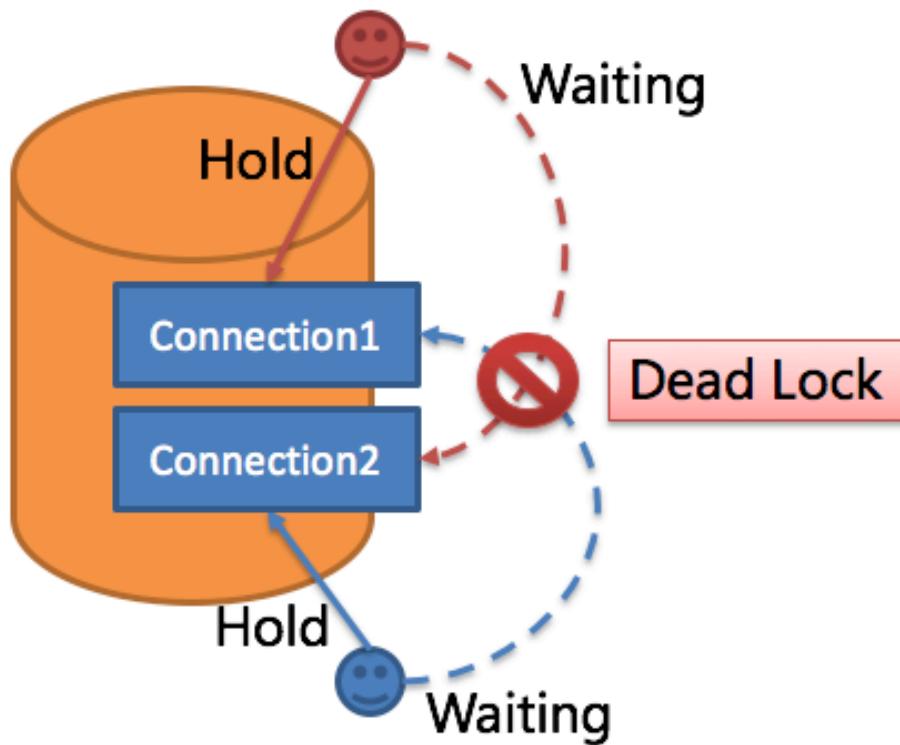


Figure8: Dead lock

To avoid the deadlock, ShardingSphere will go through synchronous processing when acquiring database connection. When creating execution units, it acquires all the database connections that this SQL requires for once with atomic method and reduces the possibility of acquiring only part of the resources. Due to the high operation frequency, locking the connection each time when acquiring it can decrease ShardingSphere's concurrency. Therefore, it has improved two aspects here:

1. Avoid the setting that locking only takes one database connection each time. Because under this kind of circumstance, two requests waiting for each other will not happen, so there is no need for locking. Most OLTP operations use sharding keys to route to the only data node, which will make the system in a totally unlocked state, thereby improve the concurrency efficiency further. In addition to routing to a single shard, readwrite-splitting also belongs to this category.

- Only aim at MEMORY_STRICTLY mode to lock resources. When using CONNECTION_STRICTLY mode, all the query result sets will release database connection resources after loading them to the memory, so deadlock wait will not appear.

Execution Phrase

Applied in actually SQL execution, this phrase can be divided into two steps: group execution and merger result generation.

Group execution can distribute execution unit groups generated in preparation phrase to the underlying concurrency engine and send events according to each key steps during the execution process, such as starting, successful and failed execution events. Execution engine only focuses on message sending rather than subscribers of the event. Other ShardingSphere modules, such as distributed transactions, invoked chain tracing and so on, will subscribe focusing events and do corresponding operations. Through the connection mode acquired in preparation phrase, ShardingSphere will generate memory merger result set or stream merger result set, and transfer it to the result merger engine for the next step.

The overall structure of execution engine is shown as the following picture:

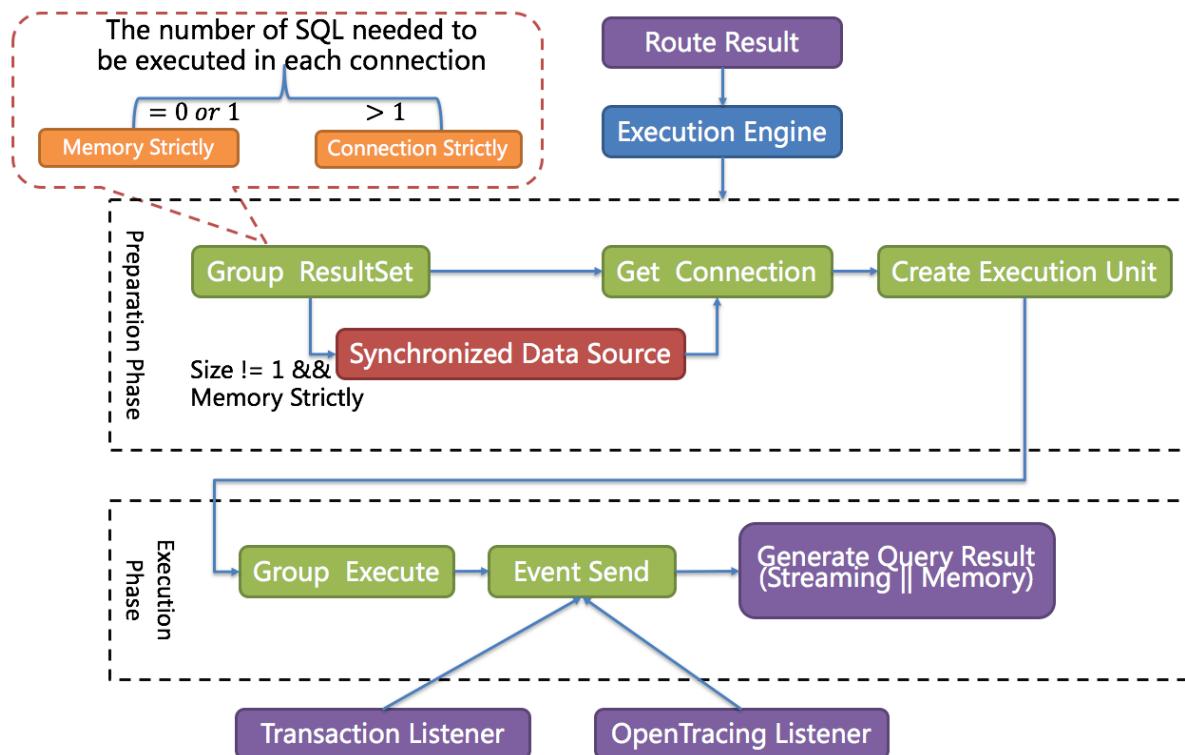


Figure9: Execute engine architecture

10.3.11 Merger Engine

Result merger refers to merging multi-data result set acquired from all the data nodes as one result set and returning it to the request end rightly.

In function, the result merger supported by ShardingSphere can be divided into five kinds, iteration, order-by, group-by, pagination and aggregation, which are in composition relation rather than clash relation. In structure, it can be divided into stream merger, memory merger and decorator merger, among which, stream merger and memory merger clash with each other; decorator merger can be further processed based on stream merger and memory merger.

Since the result set is returned from database line by line instead of being loaded to the memory all at once, the most prior choice of merger method is to follow the database returned result set, for it is able to reduce the memory consumption to a large extend.

Stream merger means, each time, the data acquired from the result set is able to return the single piece of right data line by line.

It is the most suitable one for the method that the database returns original result set. Iteration, order-by, and stream group-by belong to stream merger.

Memory merger needs to iterate all the data in the result set and store it in the memory first. after unified grouping, ordering, aggregation and other computations, it will pack it into data result set, which is visited line by line, and return that result set.

Decorator merger merges and reinforces all the result sets function uniformly. Currently, decorator merger has pagination merger and aggregation merger these two kinds.

Iteration Merger

As the simplest merger method, iteration merger only requires the combination of multiple data result sets into a single-direction chain table. After iterating current data result sets in the chain table, it only needs to move the element of chain table to the next position and iterate the next data result set.

Order-by Merger

Because there is ORDER BY statement in SQL, each data result has its own order. So it is enough only to order data value that the result set cursor currently points to, which is equal to sequencing multiple already ordered arrays, and therefore, order-by merger is the most suitable ordering algorithm in this situation.

When merging order inquiries, ShardingSphere will compare current data values in each result set (which is realized by Java Comparable interface) and put them into the priority queue. Each time when acquiring the next piece of data, it only needs to move down the result set in the top end of the line, reenter the priority order according to the new cursor and relocate its own position.

Here is an instance to explain ShardingSphere's order-by merger. The following picture is an illustration of ordering by the score. Data result sets returned by 3 tables are shown in the example and each one of them has already been ordered according to the score, but there is no order between 3 data

result sets. Order the data value that the result set cursor currently points to in these 3 result sets. Then put them into the priority queue. the data value of t_score_0 is the biggest, followed by that of t_score_2 and t_score_1 in sequence. Thus, the priority queue is ordered by the sequence of t_score_0, t_score_2 and t_score_1.

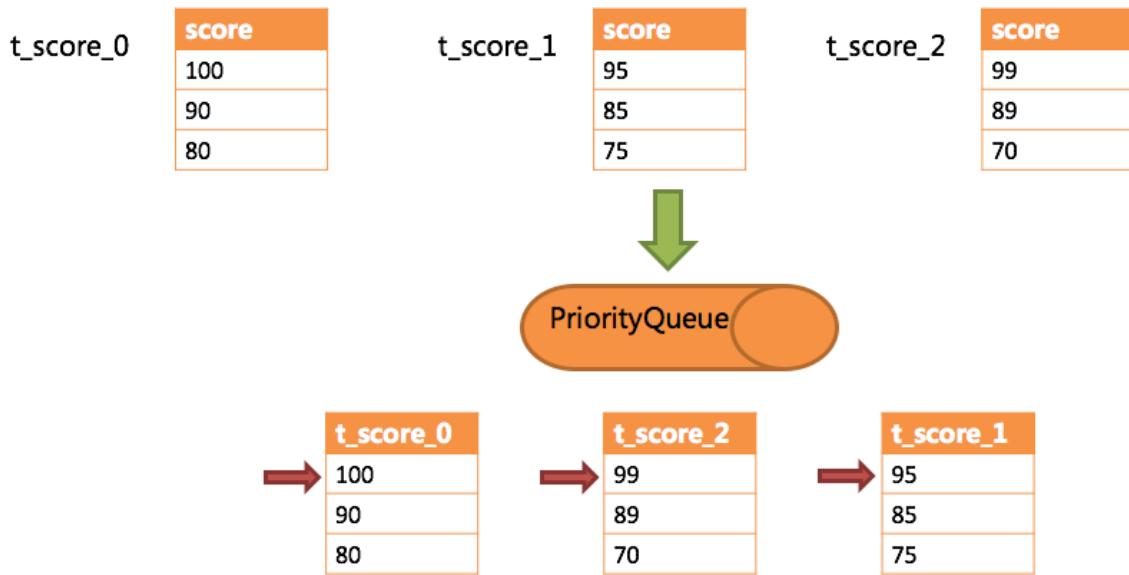


Figure10: Order by merger example 1

This diagram illustrates how the order-by merger works when using next invocation. We can see from the diagram that when using next invocation, t_score_0 at the first of the queue will be popped out. After returning the data value currently pointed by the cursor (i.e., 100) to the client end, the cursor will be moved down and t_score_0 will be put back to the queue.

While the priority queue will also be ordered according to the t_score_0 data value (90 here) pointed by the cursor of current data result set. According to the current value, t_score_0 is at the last of the queue, and in the second place of the queue formerly, the data result set of t_score_2, automatically moves to the first place of the queue.

In the second next operation, t_score_2 in the first position is popped out of the queue. Its value pointed by the cursor of the data result set is returned to the client end, with its cursor moved down to rejoin the queue, and the following will be in the same way. If there is no data in the result set, it will not rejoin the queue.

It can be seen that, under the circumstance that data in each result set is ordered while result sets are disordered, ShardingSphere does not need to upload all the data to the memory to order. In the order-by merger method, each next operation only acquires the right piece of data each time, which saves the memory consumption to a large extent.

On the other hand, the order-by merger has maintained the orderliness on horizontal axis and vertical axis of the data result set. Naturally ordered, vertical axis refers to each data result set itself, which is acquired by SQL with ORDER BY. Horizontal axis refers to the current value pointed by each data result

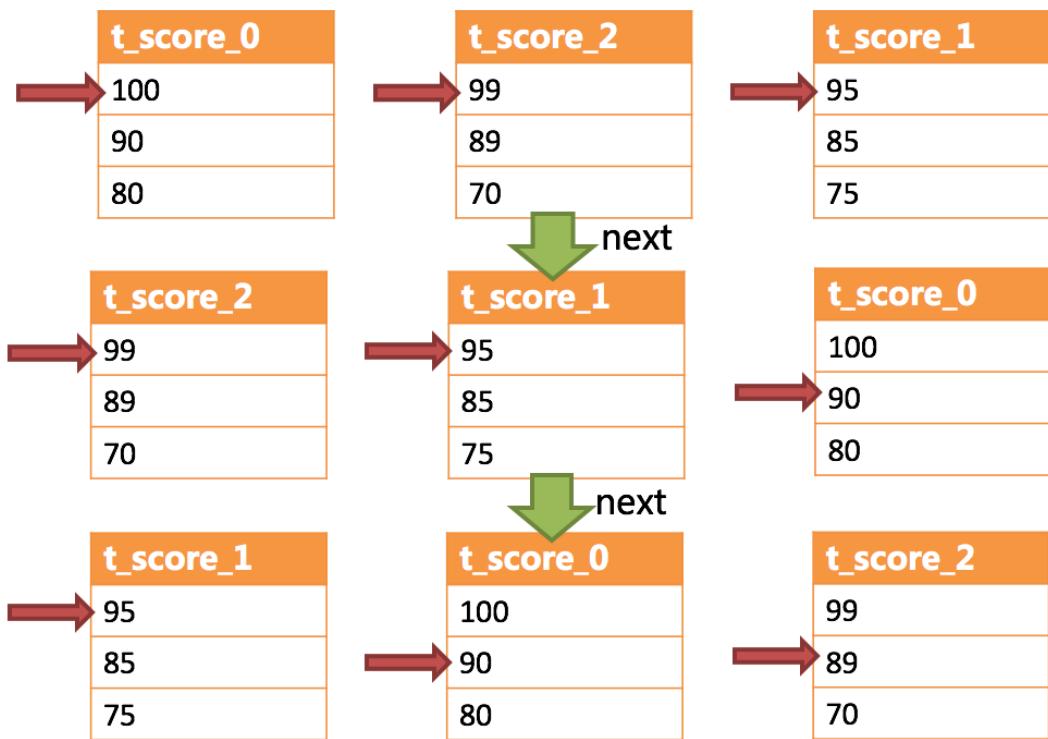


Figure11: Order by merger example 2

set, and its order needs to be maintained by the priority queue. Each time when the current cursor moves down, it requires to put the result set in the priority order again, which means only the cursor of the first data result set can be moved down.

Group-by Merger

With the most complicated situation, group-by merger can be divided into stream group-by merger and memory group-by merger. Stream group-by merger requires SQL field and order item type (ASC or DESC) to be the same with group-by item. Otherwise, its data accuracy can only be maintained by memory merger.

For instance, if it is sharded by subject, table structure contains examinees' name (to simplify, name repetition is not taken into consideration) and score. The SQL used to acquire each examinee's total score is as follow:

```
SELECT name, SUM(score) FROM t_score GROUP BY name ORDER BY name;
```

When order-by item and group-by item are totally consistent, the data obtained is continuous. The data to group are all stored in the data value that data result set cursor currently points to, stream group-by merger can be used, as illustrated by the diagram:

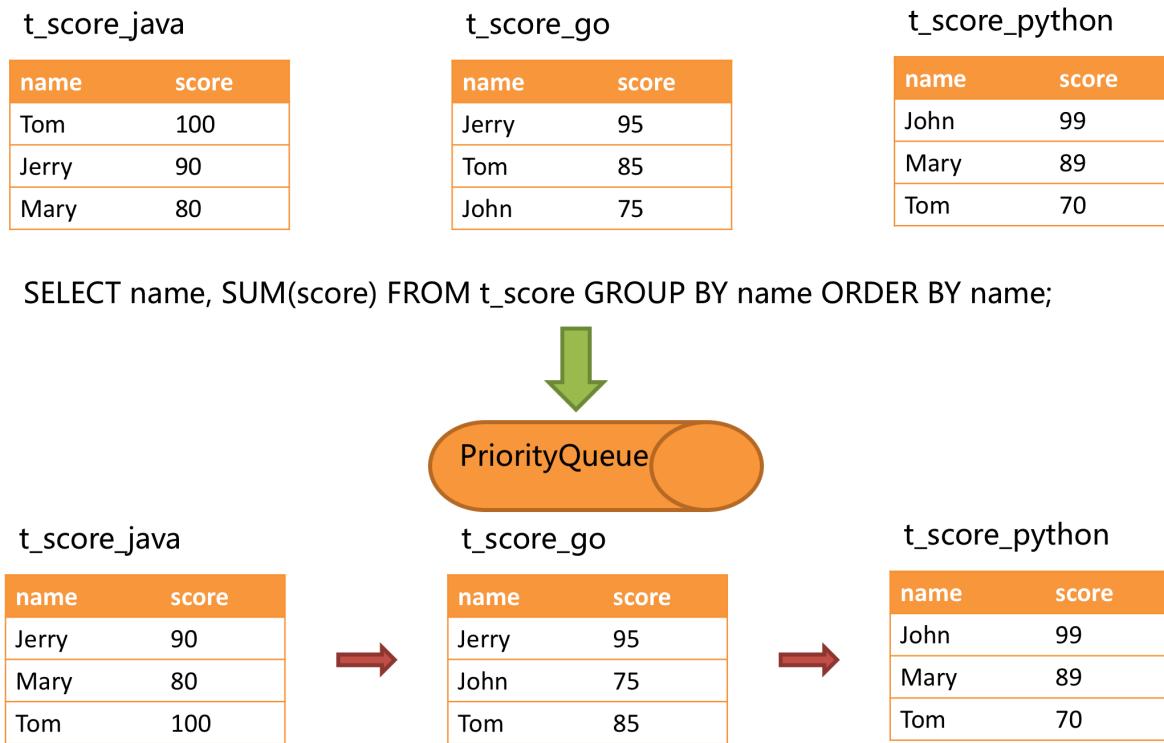


Figure12: Group by merger example 1

The merging logic is similar to that of order-by merger. The following picture shows how stream group-by merger works in next invocation.

We can see from the picture, in the first next invocation, `t_score_java` in the first position, along with other result set data also having the grouping value of "Jerry", will be popped out of the queue. After

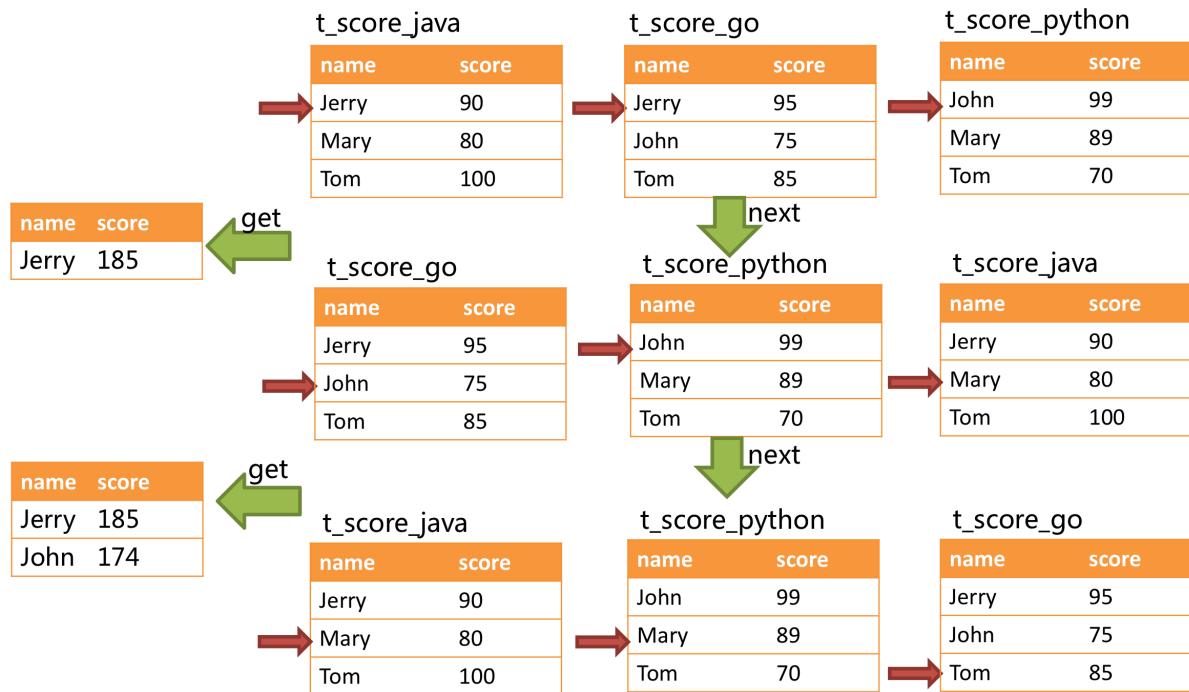


Figure13: Group by merger example 2

acquiring all the students' scores with the name of "Jerry", the accumulation operation will be proceeded. Hence, after the first next invocation is finished, the result set acquired is the sum of Jerry's scores. In the same time, all the cursors in data result sets will be moved down to a different data value next to "Jerry" and rearranged according to current result set value. Thus, the data that contains the second name "John" will be put at the beginning of the queue.

Stream group-by merger is different from order-by merger only in two points:

1. It will take out all the data with the same group item from multiple data result sets for once.
2. It does the aggregation calculation according to aggregation function type.

For the inconsistency between the group item and the order item, it requires to upload all the data to the memory to group and aggregate, since the relevant data value needed to acquire group information is not continuous, and stream merger is not able to use. For example, acquire each examinee's total score through the following SQL and order them from the highest to the lowest:

```
SELECT name, SUM(score) FROM t_score GROUP BY name ORDER BY score DESC;
```

Then, stream merger is not able to use, for the data taken out from each result set is the same as the original data of the diagram ordered by score in the upper half part structure.

When SQL only contains group-by statement, according to different database implementation, its sequencing order may not be the same as the group order. The lack of ordering statement indicates the order is not important in this SQL. Therefore, through SQL optimization re-write, ShardingSphere can automatically add the ordering item same as grouping item, converting it from the memory merger that consumes memory to stream merger.

Aggregation Merger

Whether stream group-by merger or memory group-by merger processes the aggregation function in the same way. Therefore, aggregation merger is an additional merging ability based on what have been introduced above, i.e., the decorator mode. The aggregation function can be categorized into three types, comparison, sum and average.

Comparison aggregation function refers to MAX and MIN. They need to compare all the result set data and return its maximum or minimum value directly.

Sum aggregation function refers to SUM and COUNT. They need to sum up all the result set data.

Average aggregation function refers only to AVG. It must be calculated through SUM and COUNT of SQL re-write, which has been mentioned in SQL re-write, so we will state no more here.

Pagination Merger

All the merger types above can be paginated. Pagination is the decorator added on other kinds of mergers. ShardingSphere augments its ability to paginate the data result set through the decorator mode. Pagination merger is responsible for filtering the data unnecessary to acquire.

ShardingSphere's pagination function can be misleading to users in that they may think it will take a large amount of memory. In distributed scenarios, it can only guarantee the data accuracy by rewriting `LIMIT 10000000, 10` to `LIMIT 0, 10000010`. Users can easily have the misconception that ShardingSphere uploads a large amount of meaningless data to the memory and has the risk of memory overflow. Actually, it can be known from the principle of stream merger, only memory group-by merger will upload all the data to the memory. Generally speaking, however, SQL used for OLAP grouping, is applied more frequently to massive calculation or small result generation rather than vast result data generation. Except for memory group-by merger, other cases use stream merger to acquire data result set. So ShardingSphere would skip unnecessary data through next method in result set, rather than storing them in the memory.

What's to be noticed, pagination with `LIMIT` is not the best practice actually, because a large amount of data still needs to be transmitted to ShardingSphere's memory space for ordering. `LIMIT` cannot search for data by index, so paginating with ID is a better solution on the premise that the ID continuity can be guaranteed. For example:

```
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:

```
SELECT * FROM t_order WHERE id > 10000000 LIMIT 10;
```

The overall structure of merger engine is shown in the following diagram:

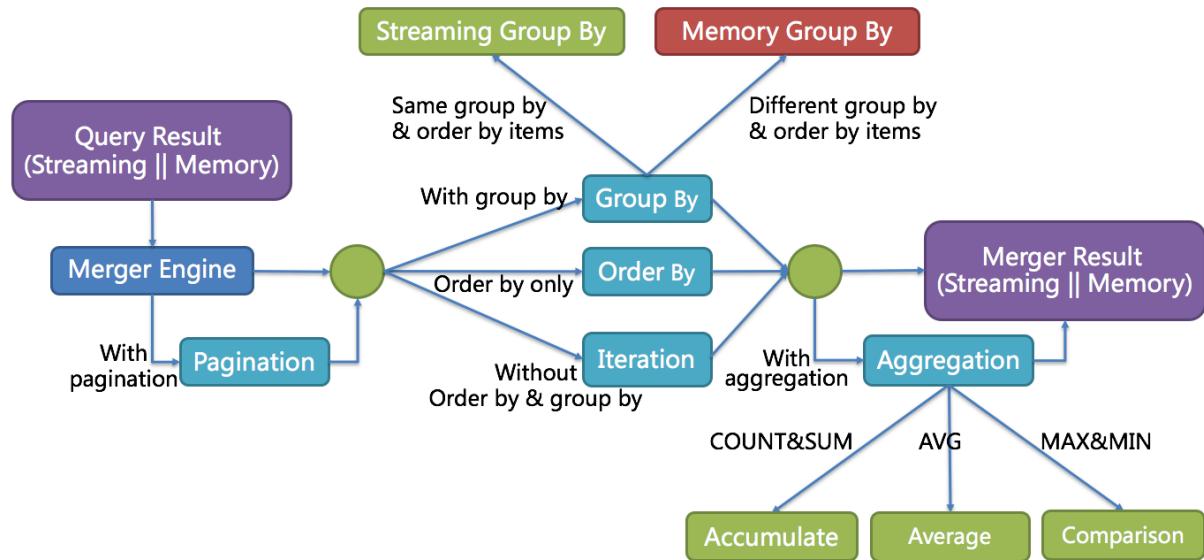


Figure14: Merge Architecture

10.4 Transaction

10.4.1 Navigation

This chapter mainly introduces the principles of the distributed transactions:

- 2PC transaction with XA
- BASE transaction with Seata

10.4.2 XA Transaction

`XAShardingSphereTransactionManager` is XA transaction manager of Apache ShardingSphere. Its main responsibility is manage and adapt multiple data sources, and sent corresponding transactions to concrete XA transaction manager.

Transaction Begin

When receiving `set autoCommit=0` from client, `XAShardingSphereTransactionManager` will use XA transaction managers to start overall XA transactions, which is marked by XID.

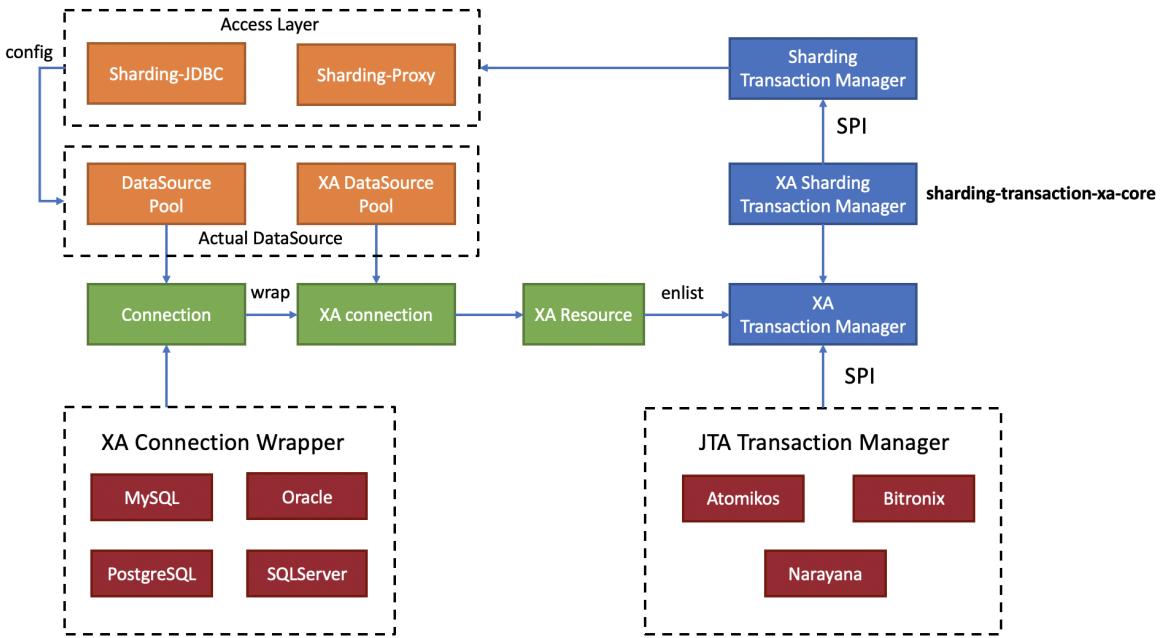


Figure15: Principle of ShardingSphere transaction XA

Execute actual sharding SQL

After XAShadingSphereTransactionManager 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 XAShadingSphereTransactionManager 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

```

10.4.3 Seata BASE transaction

When integrating Seata AT transaction, we need to integrate TM, RM and TC component into ShardingSphere transaction manager. Seata have proxied DataSource interface in order to RPC with TC. Similarly, Apache ShardingSphere faced to DataSource interface to aggregate data sources too. After Seata DataSource encapsulation, it is easy to put Seata AT transaction into Apache ShardingSphere sharding ecosystem.

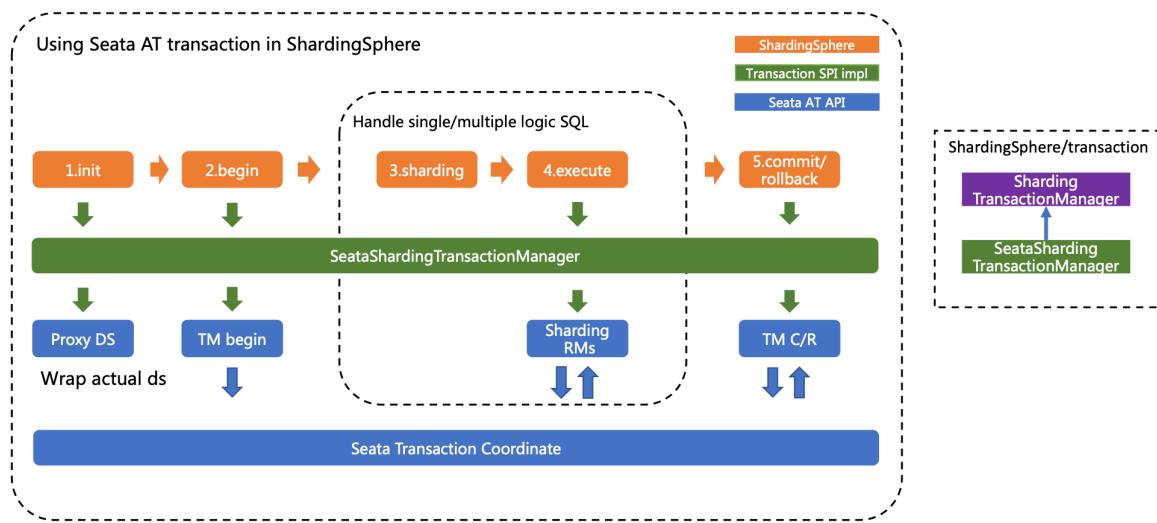


Figure16: Seata BASE transaction

Init Seata Engine

When an application containing `ShardingSphereTransactionBaseSeataAT` startup, the user-configured `DataSource` will be wrapped into `seata DataSourceProxy` through `seata.conf`, then registered into RM.

Transaction Begin

TM controls the boundaries of global transactions. TM obtains the global transaction ID by sending `Begin` instructions to TC. All branch transactions participate in the global transaction through this global transaction ID. The context of the global transaction ID will be stored in the thread local variable.

Execute actual sharding SQL

Actual SQL in Seata global transaction will be intercepted to generate undo snapshots by RM and sends participate instructions to TC to join global transaction. Since actual sharding SQLs executed in multi-threads, global transaction context should transfer from main thread to child thread, which is exactly the same as context transfer between services.

Commit or Rollback

When submitting a seata transaction, TM sends TC the commit and rollback instructions of the global transaction. TC coordinates all branch transactions for commit and rollback according to the global transaction ID.

10.5 Scaling

10.5.1 Principle Description

Consider about these challenges of ShardingSphere-Scaling, the solution is: Use two database clusters temporarily, and switch after the scaling is completed.

Advantages:

1. No effect for origin data during scaling.
2. No risk for scaling failure.
3. No limited by sharding strategies.

Disadvantages:

1. Redundant servers during scaling.
2. All data needs to be moved.

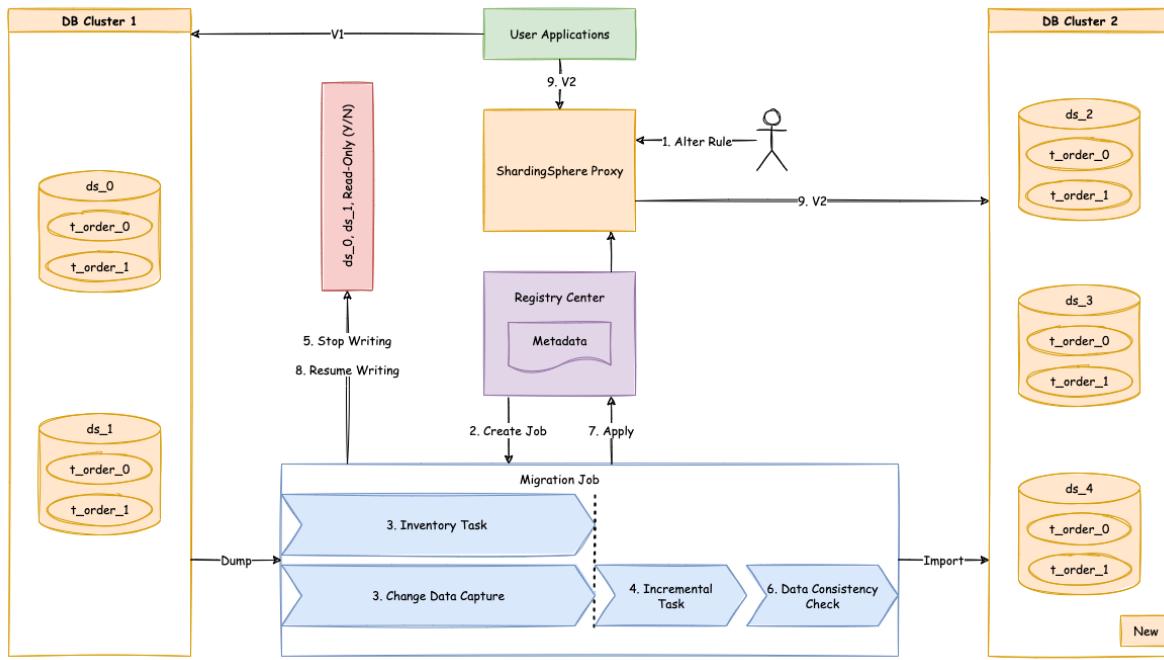


Figure17: Scaling Principle Overview

ShardingSphere-Scaling will analyze the sharding rules and extract information like datasource and data nodes. According the sharding rules, ShardingSphere-Scaling create a scaling job with 4 main phases.

1. Preparing Phase.
2. Inventory Phase.
3. Incremental Phase.
4. Switching Phase.

10.5.2 Phase Description

Preparing Phase

ShardingSphere-Scaling will check the datasource connectivity and permissions, statistic the amount of inventory data, record position of log, shard tasks based on amount of inventory data and the parallelism set by the user.

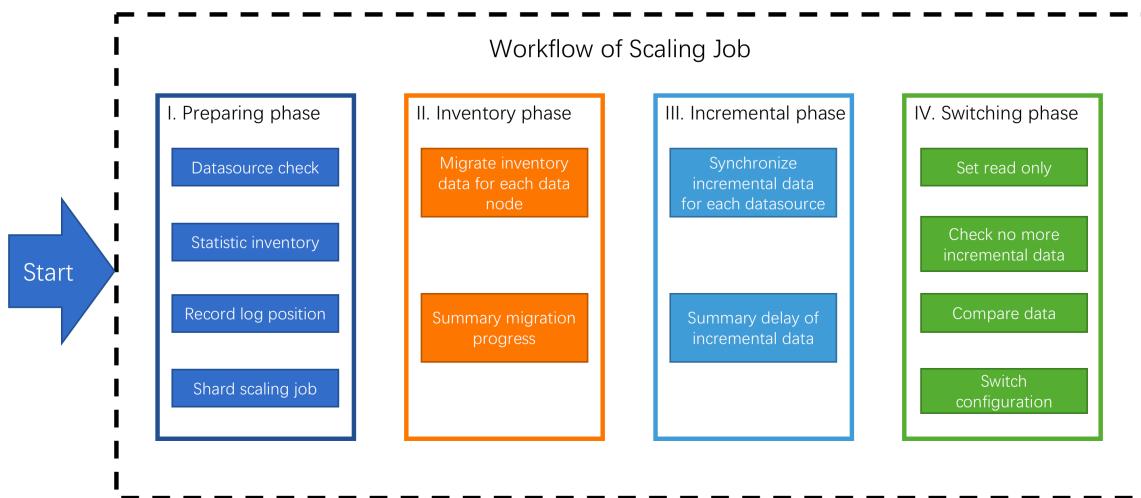


Figure18: Workflow

Inventory Phase

Executing the Inventory data migration tasks sharded in preparing phase. ShardingSphere-Scaling uses JDBC to query inventory data directly from data nodes and write to the new cluster using new rules.

Incremental Phase

The data in data nodes is still changing during the inventory phase, so ShardingSphere-Scaling need to synchronize these incremental data to new data nodes. Different databases have different implementations, but generally implemented by change data capture function based on replication protocols or WAL logs.

- MySQL: subscribe and parse binlog.
- PostgreSQL: official logic replication `test_decoding`.

These captured incremental data, Apache ShardingSphere also write to the new cluster using new rules.

Switching Phase

In this phase, there may be a temporary read only time, make the data in old data nodes static so that the incremental phase complete fully. The read only time is range seconds to minutes, it depends on the amount of data and the checking data. After finished, Apache ShardingSphere can switch the configuration by register-center and config-center, make application use new sharding rule and new data nodes.

10.6 Encryption

10.6.1 Process Details

Apache ShardingSphere can encrypt the plaintext by parsing and rewriting SQL according to the encryption rule, and store the plaintext (optional) and ciphertext data to the database at the same time. Queries data only extracts the ciphertext data from database and decrypts it, and finally returns the plaintext to user. Apache ShardingSphere transparently process of data encryption, so that users do not need to know to the implementation details of it, use encrypted data just like as regular data. In addition, Apache ShardingSphere can provide a relatively complete set of solutions whether the online business system has been encrypted or the new online business system uses the encryption function.

Overall Architecture

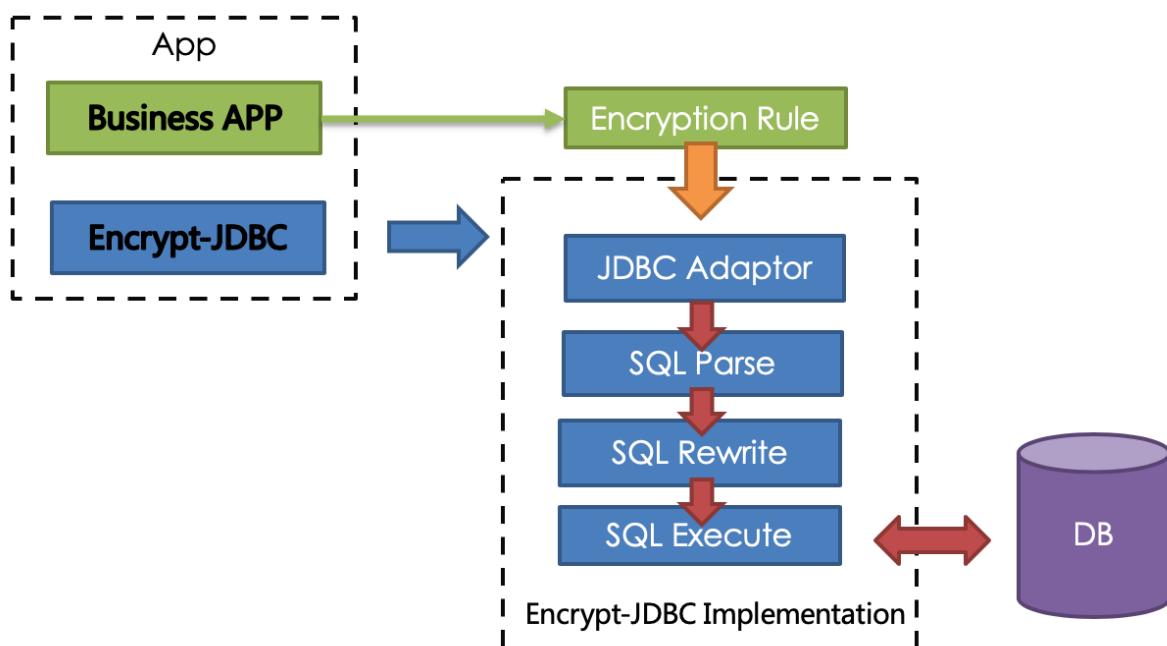


Figure19: 1

Encrypt module intercepts SQL initiated by user, analyzes and understands SQL behavior through the SQL syntax parser. According to the encryption rules passed by the user, find out the fields that need to be encrypted/decrypted and the encryptor/decryptor used to encrypt/decrypt the target fields, and then interact with the underlying database. ShardingSphere will encrypt the plaintext requested by the user and store it in the underlying database; and when the user queries, the ciphertext will be taken out of the database for decryption and returned to the end user. ShardingSphere shields the encryption of data, so that users do not need to perceive the process of parsing SQL, data encryption, and data decryption, just like using ordinary data.

Encryption Rule

Before explaining the whole process in detail, we need to understand the encryption rules and configuration, which is the basis of understanding the whole process. The encryption configuration is mainly divided into four parts: data source configuration, encrypt algorithm configuration, encryption table rule configuration, and query attribute configuration. The details are shown in the following figure:

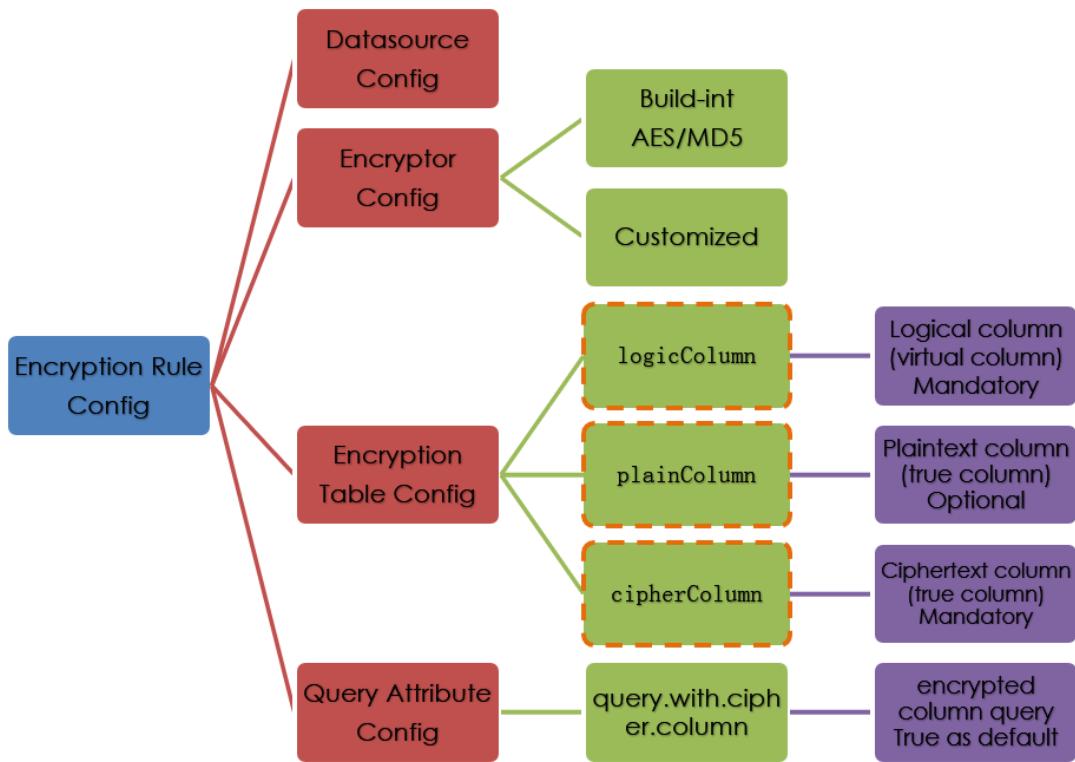


Figure20: 2

Data Source Configuration: The configuration of DataSource.

Encrypt Algorithm Configuration: What kind of encryption strategy to use for encryption and decryption. Currently ShardingSphere has five built-in encryption/decryption strategies: AES, MD5, RC4, SM3, SM4. 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), what algorithm is used to encryption/decryption (encryptorName), which column is used to store assisted query data (assistedQueryColumn), what algorithm is used to encrypt/decrypt assisted query data (assistedQueryEncryptorName), 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, ciphertext data in cipherColumn, and store assisted query data to the assistedQueryColumn. In other words, we do not even want users to know the existence and use of plainColumn, cipherColumn and assistedQueryColumn. 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, assistedQueryColumn. 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, assistedQueryColumn 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. This switch supports table level and whole rule level configuration, and table level has the highest priority.

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, pwd_assisted_query, used to store the auxiliary query 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 decrypts 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.

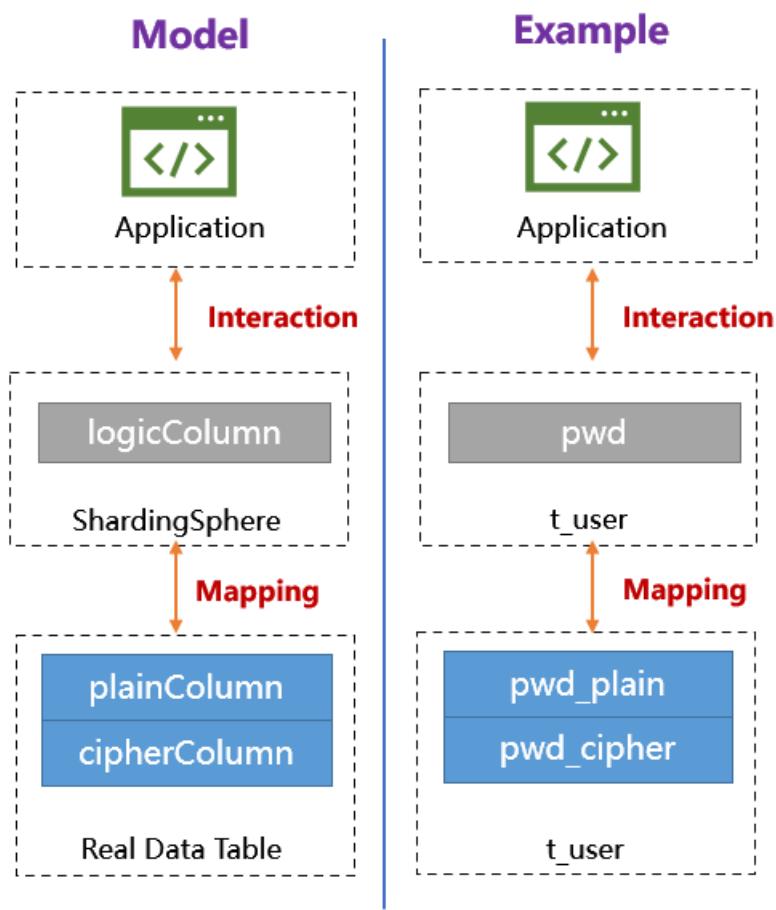


Figure21: 3

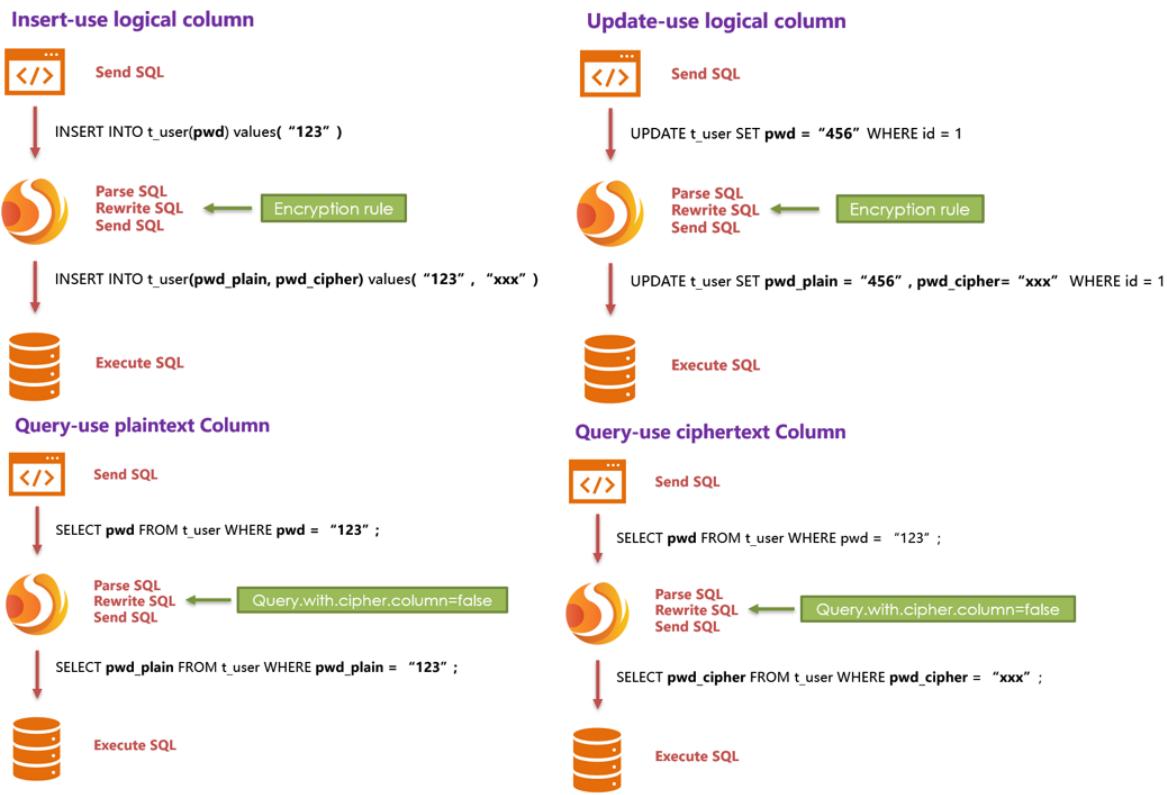


Figure22: 4

10.6.2 Detailed Solution

After understanding the Apache ShardingSphere encryption process, you can combine the encryption configuration and encryption process with the actual scenario. All design and development are to solve the problems encountered in business scenarios. So for the business scenario requirements mentioned earlier, how should ShardingSphere be used to achieve business requirements?

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_cipher
        encryptorName: aes_encryptor
        assistedQueryColumn: pwd_assisted_query
        assistedQueryEncryptorName: pwd_assisted_query_cipher
        queryWithCipherColumn: true

```

With this configuration, Apache ShardingSphere only needs to convert logicColumn and cipherColumn, assistedQueryColumn. The underlying data table does not store plain text, only cipher text. This is also a requirement of the security audit part. If users want to store plain text and cipher text together in the database, they just need to add plainColumn configuration. The overall processing flow is shown below:

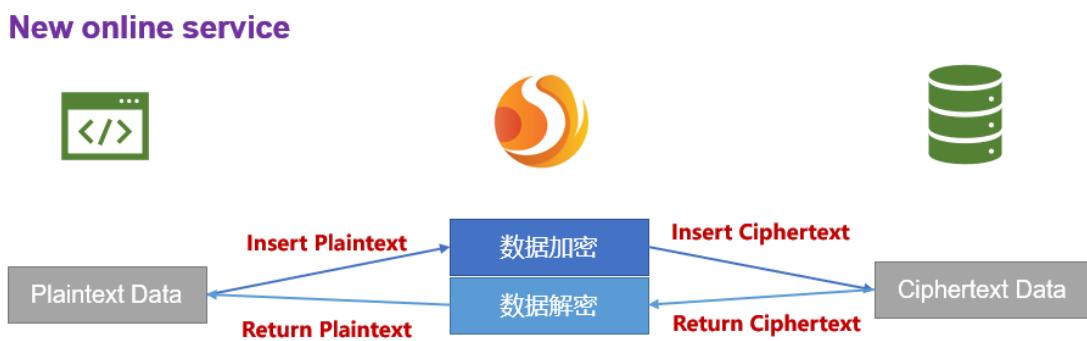


Figure23: 5

Online Business Transformation

Business scenario analysis: As the business is already running online, there must be a large amount of plain text historical data stored in the database. The current challenges are how to enable historical data to be encrypted and cleaned, how to enable incremental data to be encrypted, and how to allow businesses to seamlessly and transparently migrate between the old and new data systems.

Solution description: Before providing a solution, let's brainstorm: First, if the old business needs to be desensitized, it must have stored very important and sensitive information. This information has a high gold content and the business is relatively important. If it is broken, the whole team KPI is over. Therefore, it is impossible to suspend business immediately, prohibit writing of new data, encrypt and clean all historical data with an encrypt algorithm, and then deploy the previously reconstructed code online, so that it can encrypt and decrypt online and incremental data. Such a simple and rough way, based on historical experience, will definitely not work.

Then another relatively safe approach is to rebuild a pre-release environment exactly like the pro-

duction 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
encrytors:
  aes_encryptor:
    type: AES
    props:
      aes-key-value: 123456abc
tables:
  t_user:
    columns:
      pwd:
        plainColumn: pwd
        cipherColumn: pwd_cipher
        encryptorName: aes_encryptor
        assistedQueryColumn: pwd_assisted_query
        assistedQueryEncryptorName: pwd_assisted_query_cipher
        queryWithCipherColumn: false
```

According to the above encryption rules, we need to add a 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:

Online Service Refactor-before migration

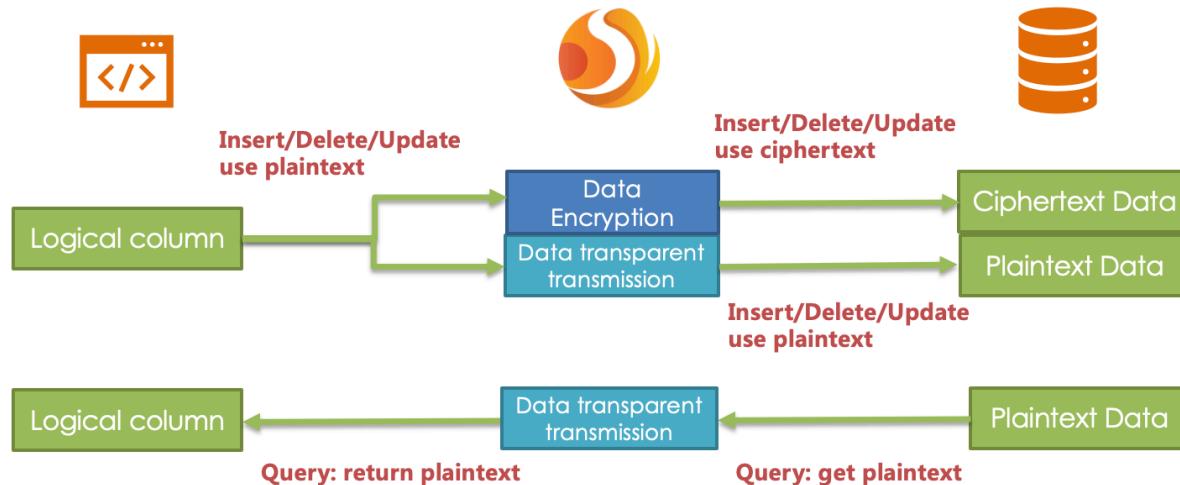


Figure24: 6

When the newly added data is inserted, it is encrypted as ciphertext data through Apache ShardingSphere and stored in the `cipherColumn`. Now it is necessary to process historical plaintext inventory data. **As Apache ShardingSphere currently does not provide the corresponding migration and washing tools, the business party needs to encrypt and store the plain text data in `pwd` to `pwd_cipher`.**

2. During system migration

The incremental data has been stored by Apache ShardingSphere in the ciphertext column and the plaintext is stored in the plaintext column; after the historical data is encrypted and cleaned by the business party itself, the ciphertext is also stored in the ciphertext column. That is to say, the plaintext and the ciphertext are stored in the current database. Since the `queryWithCipherColumn = false` in the configuration item, the ciphertext has never been used. Now we need to set the `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 in 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:

Online Service Refactor-in migration

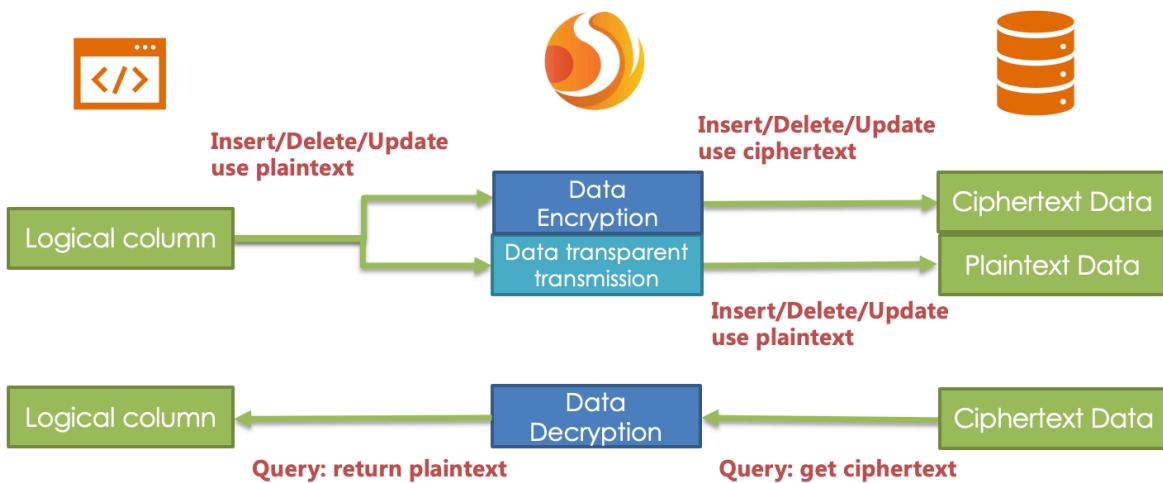


Figure25: 7

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 与 pwd_cipher 的转换映射
```

```

cipherColumn: pwd_cipher
encryptorName: aes_encryptor
assistedQueryColumn: pwd_assisted_query
assistedQueryEncryptorName: pwd_assisted_query_cipher
queryWithCipherColumn: true

```

The processing flow is as follows:

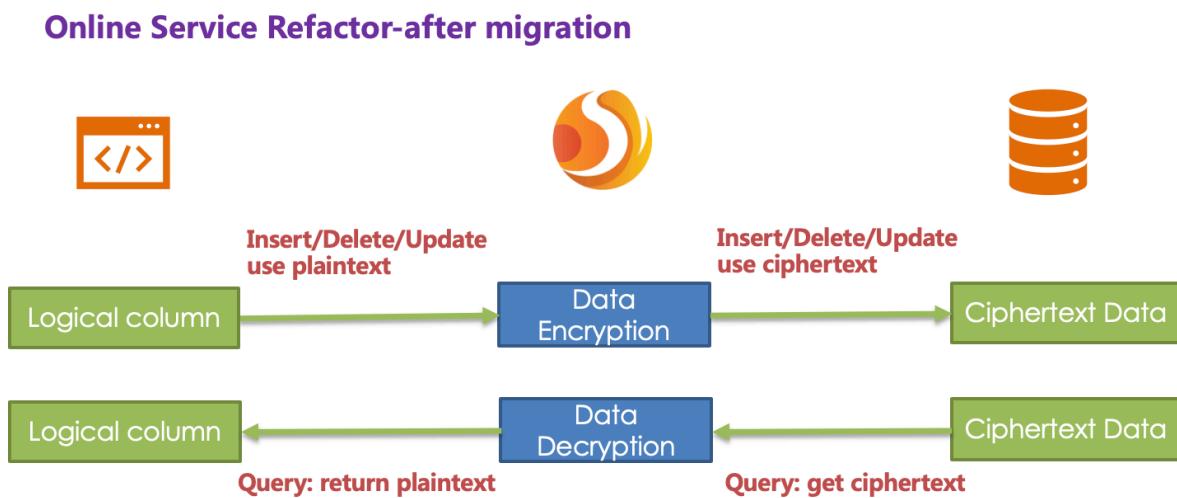


Figure26: 8

4. System migration completed

Security audit department then requested, the business system needs to modify the key periodically or certain emergency security events trigger, we need to migrate the number of wash again, that is, use the old key decryption and then use the new key encryption. Both to and also to the problem came, the plaintext column data has been deleted, the amount of data in the database table tens of millions, the migration shuffle takes a certain amount of time, the migration shuffle process in the cipher column changes, the system also needs to provide services correctly. What to do? The answer is: auxiliary query column. **Because auxiliary query columns generally use algorithms such as irreversible MD5 and SM3, queries based on auxiliary columns are served correctly by the system even during the migration shuffle.**

So far, the online service encryption and rectification solutions have all been demonstrated. We provide Java, YAML, Spring Boot Starter, Spring Namespace multiple ways for users to choose to use, and strive to fulfill business requirements. The solution has been continuously launched on JD Digits, providing internal basic service support.

10.6.3 The advantages of Middleware encryption service

1. Transparent data encryption process, users do not need to pay attention to the implementation details of encryption.
2. Provide a variety of built-in, third-party (AKS) encryption strategies, users only need to modify the configuration to use.
3. Provides a encryption strategy API interface, users can implement the interface to use a custom encryption strategy for data encryption.
4. Support switching different encryption strategies.
5. For online services, it is possible to store plaintext data and ciphertext data synchronously, and decide whether to use plaintext or ciphertext columns for query through configuration. Without changing the business query SQL, the on-line system can safely and transparently migrate data before and after encryption.

10.6.4 Solution

Apache ShardingSphere has provided the encryption solution for data encryption, the `EncryptAlgorithm`.

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.

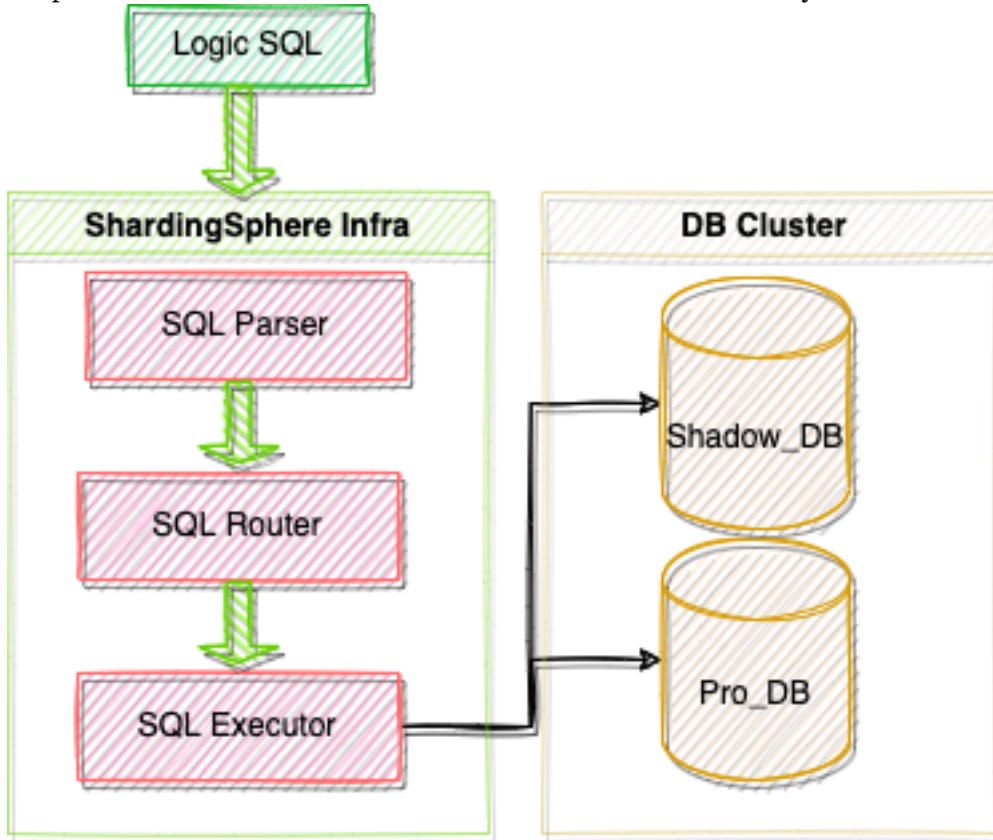
When users `INSERT`, `DELETE` and `UPDATE`, ShardingSphere will parse, rewrite and route SQL according to the configuration. It will also use `encrypt()` to encrypt data and store them in the database. When using `SELECT`, they will decrypt sensitive data from the database with `decrypt()` reversely and return them to users at last.

Currently, Apache ShardingSphere has provided five types of implementations for this kind of encrypt solution, MD5 (irreversible), AES (reversible), RC4 (reversible), SM3 (irreversible) and SM4 (reversible), which can be used after configuration.

10.7 Shadow

10.7.1 How it works

Apache ShardingSphere determines the incoming SQL via shadow by parsing the SQL and routing it to the production or shadow database based on the shadow rules set by the user in the configuration file.



In the example of an INSERT statement, when writing data, Apache ShardingSphere parses the SQL and then constructs a routing chain based on the rules in the configuration file. In the current version, the shadow feature is at the last execution unit in the routing chain, i.e. if other rules exist that require routing, such as sharding, Apache ShardingSphere will first route to a particular database according to the sharding rules, and then run the shadow routing determination process to determine that the execution SQL meets the configuration set by shadow rules. Then data is routed to the corresponding shadow database, while the production data remains unchanged.

DML sentence

Two algorithms are supported. Shadow determination first determines whether the execution SQL-related table intersects with the configured shadow table. If the result is positive, the shadow algorithm within the part of intersection associated with the shadow table will be determined sequentially. If any of the determination is successful, the SQL statement is routed to the shadow library. If there is no intersection or the shadow algorithm determination is unsuccessful, the SQL statement is routed to the production database.

DDL sentence

Only supports shadow algorithm with comments attached. In stress testing scenarios, DDL statements are generally not required for testing, and are used mainly when initializing or modifying shadow tables in the shadow database. The shadow determination will first determine whether the execution SQL contains comments or not. If the result is a yes, the HINT shadow algorithm configured in the shadow rules determines them in order. The SQL statement is routed to the shadow database if any of the determinations are successful. If the execution SQL does not contain comments or the HINT shadow algorithm determination is unsuccessful, the SQL statements are routed to the production database.

##References JAVA API: shadow database configuration

YAMLconfiguration: shadow database

Spring Boot Starter: shadow database configuration

Spring namespace: shadow database configuration

10.8 FAQ

10.8.1 [JDBC] Why there may be an error when configure both shardingsphere-jdbc-spring-boot-starter and a spring-boot-starter of certain datasource pool(such as druid)?

Answer:

1. Because the spring-boot-starter of certain datasource pool (such as druid) will be configured before shardingsphere-jdbc-spring-boot-starter and create a default datasource, then conflict occur when ShardingSphere-JDBC create datasources.
2. A simple way to solve this issue is removing the spring-boot-starter of certain datasource pool, shardingsphere-jdbc create datasources with suitable pools.

10.8.2 [JDBC] Why is xsd unable to be found when Spring Namespace is used?

Answer:

The use norm of Spring Namespace does not require to deploy xsd files to the official website. But considering some users' needs, we will deploy them to ShardingSphere's official website.

Actually, META-INF:raw-latex:spring.schemas in the jar package of shardingsphere-jdbc-spring-namespace has been configured with the position of xsd files: META-INF:raw-latex:namespace:**raw-latex:\sharding**.xsd and META-INF:raw-latex:namespace:**raw-latex:\replica**-query.xsd, so you only need to make sure that the file is in the jar package.

10.8.3 [JDBC] Found a JtaTransactionManager in spring boot project when integrating with transaction of XA

Answer:

1. shardingsphere-transaction-xa-core include atomikos, it will trigger auto-configuration mechanism in spring-boot, add @SpringBootApplication(exclude = JtaAutoConfiguration.class) will solve it.

10.8.4 [Proxy] In Windows environment, could not find or load main class org.apache.shardingsphere.proxy.Bootstrap, how to solve it?

Answer:

Some decompression tools may truncate the file name when decompressing the ShardingSphere-Proxy binary package, resulting in some classes not being found.

The solutions:

Open cmd.exe and execute the following command:

```
tar zxvf apache-shardingsphere-${RELEASE.VERSION}-shardingsphere-proxy-bin.tar.gz
```

10.8.5 [Proxy] How to add a new logic database dynamically when use ShardingSphere-Proxy?

Answer:

When using ShardingSphere-Proxy, users can dynamically create or drop logic database through Dist-SQL, the syntax is as follows:

```
CREATE DATABASE [IF NOT EXISTS] databaseName;
```

```
DROP DATABASE [IF EXISTS] databaseName;
```

Example:

```
CREATE DATABASE sharding_db;
```

```
DROP DATABASE sharding_db;
```

10.8.6 [Proxy] How to use a suitable database tools connecting ShardingSphere-Proxy?

Answer:

1. ShardingSphere-Proxy could be considered as a mysql sever, so we recommend using mysql command line tool to connect to and operate it.
2. If users would like use a third-party database tool, there may be some errors cause of the certain implementation/options.
3. The currently tested third-party database tools are as follows:
 - Navicat: 11.1.13, 15.0.20.
 - DataGrip: 2020.1, 2021.1 (turn on “introspect using jdbc metadata” in idea or datagrip).
 - WorkBench: 8.0.25.

10.8.7 [Proxy] When using a client such as Navicat to connect to ShardingSphere-Proxy, if ShardingSphere-Proxy does not create a database or does not add a resource, the client connection will fail?

Answer:

1. Third-party database tools will send some SQL query metadata when connecting to ShardingSphere-Proxy. When ShardingSphere-Proxy does not create a database or does not add a resource, ShardingSphere-Proxy cannot execute SQL.
2. It is recommended to create database and resource first, and then use third-party database tools to connect.
3. Please refer to [Related introduction](#) the details about resource.

10.8.8 [Sharding] How to solve Cloud not resolve placeholder …in string value … error?

Answer:

`${...}` or `$->{...}` can be used in inline expression identifiers, but the former one clashes with place holders in Spring property files, so `$->{...}` is recommended to be used in Spring as inline expression identifiers.

10.8.9 [Sharding] Why does float number appear in the return result of inline expression?

Answer:

The division result of Java integers is also integer, but in Groovy syntax of inline expression, the division result of integers is float number. To obtain integer division result, A/B needs to be modified as A.intdiv(B).

10.8.10 [Sharding] If sharding database is partial, should tables without sharding database & table configured in sharding rules?

Answer:

No, ShardingSphere will recognize it automatically.

10.8.11 [Sharding] When generic Long type `SingleKeyTableShardingAlgorithm` is used, why `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 generic type is Integer, if you want to configure Long type, please make sure the field type in the database is bigint.

10.8.12 [Sharding:raw-latex:PROXY] When implementing the `StandardShardingAlgorithm` custom algorithm, the specific type of `Comparable` is specified as `Long`, and the field type in the database table is `bigint`, a `ClassCastException: Integer can not cast to Long` exception occurs.

Answer:

When implementing the doSharding method, it is not recommended to specify the specific type of Comparable in the method declaration, but to convert the type in the implementation of the doSharding method. You can refer to the ModShardingAlgorithm#doSharding method.

10.8.13 [Sharding] Why are the default distributed auto-augment key strategy provided by ShardingSphere not continuous and most of them end with even numbers?

Answer:

ShardingSphere uses snowflake algorithms as the default distributed auto-augment key strategy to make sure unrepeatable 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>

10.8.14 [Sharding] How to allow range query with using inline sharding strategy(BETWEEN AND, >, <, >=, <=)?

Answer:

1. Update to 4.1.0 above.
2. Configure(A tip here: then each range query will be broadcast to every sharding table):
 - Version 4.x: `allow.range.query.with.inline.sharding` to `true` (Default value is `false`).
 - Version 5.x: `allow-range-query-with-inline-sharding` to `true` in `InlineShardingStrategy` (Default value is `false`).

10.8.15 [Sharding] Why does my custom distributed primary key do not work after implementing KeyGenerateAlgorithm interface and configuring type property?

Answer:

Service Provider Interface (SPI) is a kind of API for the third party to implement or expand. Except implementing interface, you also need to create a corresponding file in META-INF/services to make the JVM load these SPI implementations.

More detail for SPI usage, please search by yourself.

Other ShardingSphere functionality implementation will take effect in the same way.

10.8.16 [Sharding] In addition to internal distributed primary key, does ShardingSphere support other native auto-increment keys?

Answer:

Yes. But there is restriction to the use of native auto-increment keys, which means they cannot be used as sharding keys at the same time.

Since ShardingSphere does not have the database table structure and native auto-increment key is not included in original SQL, it cannot parse that field to the sharding field. If the auto-increment key is not sharding key, it can be returned normally and is needless to be cared. But if the auto-increment key is also used as sharding key, ShardingSphere cannot parse its sharding value, which will make SQL routed to multiple tables and influence the rightness of the application.

The premise for returning native auto-increment key is that INSERT SQL is eventually routed to one table. Therefore, auto-increment key will return zero when INSERT SQL returns multiple tables.

10.8.17 [Encryption] How to solve that data encryption can't work with JPA?

Answer:

Because DDL for data encryption has not yet finished, JPA Entity cannot meet the DDL and DML at the same time, when JPA that automatically generates DDL is used with data encryption.

The solutions are as follows:

1. Create JPA Entity with logicColumn which needs to encrypt.
2. Disable JPA auto-ddl, For example setting auto-ddl=none.
3. Create table manually. Table structure should use cipherColumn,plainColumn and assistedQueryColumn to replace the logicColumn.

10.8.18 [DistSQL] How to set custom JDBC connection properties or connection pool properties when adding a data source using DistSQL?

Answer:

1. If you need to customize JDBC connection properties, please take the urlSource way to define dataSource.
2. ShardingSphere presets necessary connection pool properties, such as maxPoolSize, idleTimeout, etc. If you need to add or overwrite the properties, please specify it with PROPERTIES in the dataSource.
3. Please refer to [Related introduction](#) for above rules.

10.8.19 [DistSQL] How to solve Resource [xxx] is still used by [SingleTableRule]. exception when dropping a data source using DistSQL?

Answer:

1. Resources referenced by rules cannot be deleted
2. If the resource is only referenced by single table rule, and the user confirms that the restriction can be ignored, the optional parameter ignore single tables can be added to perform forced deletion

```
DROP RESOURCE dataSourceName [, dataSourceName] ... [ignore single tables]
```

10.8.20 [DistSQL] How to solve Failed to get driver instance for jd-bcURL=xxx. exception when adding a data source using DistSQL?

Answer:

ShardingSphere Proxy do not have jdbc driver during deployment. Some example of this include mysql-connector. To use it otherwise following syntax can be used:

```
ADD RESOURCE dataSourceName [..., dataSourceName]
```

10.8.21 [Other] How to debug when SQL can not be executed rightly in ShardingSphere?

Answer:

sql.show configuration is provided in ShardingSphere-Proxy and post-1.5.0 version of ShardingSphere-JDBC, enabling the context parsing, rewritten SQL and the routed data source printed to info log. sql.show configuration is off in default, and users can turn it on in configurations.

A Tip: Property sql.show has changed to sql-show in version 5.x.

10.8.22 [Other] Why do some compiling errors appear? Why did not the IDEA index the generated codes?

Answer:

ShardingSphere uses lombok to enable minimal coding. For more details about using and installment, please refer to the official website of [lombok](#).

The codes under the package org.apache.shardingsphere.sql.parser.autogen are generated by ANTLR. You may execute the following command to generate codes:

```
./mvnw -Dcheckstyle.skip=true -Drat.skip=true -Dmaven.javadoc.skip=true -Djacoco.skip=true -DskipITs -DskipTests install -T1C
```

The generated codes such as org.apache.shardingsphere.sql.parser.autogen.PostgreSQLStatementParser may be too large to be indexed by the IDEA. You may configure the IDEA's property idea.max.intellisense.filesize=10000.

10.8.23 [Other] In SQLServer 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;
```

10.8.24 [Other] Why does Oracle database throw “Order by value must implements Comparable” exception when using Timestamp Order By?

Answer:

There are two solutions for the above problem: 1. Configure JVM parameter “-oracle.jdbc.J2EE13Compliant=true” 2. Set System.getProperties().setProperty(“oracle.jdbc.J2EE13Compliant”, “true”) codes in the initialization of the project.

Reasons:

```
org.apache.shardingsphere.sharding.merge.dql.orderby.OrderByValue#getValue():
```

```
private List<Comparable<?>> getValues() throws SQLException {
    List<Comparable<?>> result = new ArrayList<>(orderByItems.size());
    for (OrderByItem each : orderByItems) {
        Object value = queryResult.getValue(each.getIndex(), Object.class);
        Preconditions.checkNotNull(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 ora-

cle.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;
}

```

10.8.25 [Other] In Windows environment,when cloning ShardingSphere source code through Git, why prompt filename too long and how to solve it?

Answer:

To ensure the readability of source code, the ShardingSphere Coding Specification requires that the naming of classes, methods and variables be literal and avoid abbreviations, which may result in Some source files have long names.

Since the Git version of Windows is compiled using msys, it uses the old version of Windows Api, limiting the file name to no more than 260 characters.

The solutions are as follows:

Open cmd.exe (you need to add git to environment variables) and execute the following command to allow git supporting log paths:

```
git config --global core.longpaths true
```

If we use windows 10, also need enable win32 log paths in registry editor or group strategy(need reboot):
> Create the registry key HKLM\SYSTEM\CurrentControlSet\Control\FileSystem LongPath-
sEnabled (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>

10.8.26 [Other] How to solve Type is required error?

Answer:

In Apache ShardingSphere, many functionality implementation are uploaded through SPI, such as Distributed Primary Key. These functions load SPI implementation by configuring the type, so the type must be specified in the configuration file.

10.8.27 [Other] How to speed up the metadata loading when service starts up?

Answer:

1. Update to 4.0.1 above, which helps speed up the process of loading table metadata.
2. Configure:
 - max.connections.size.per.query(Default value is 1) higher referring to connection pool you adopt(Version \geq 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 \geq 5.0.0).

10.8.28 [Other] The ANTLR plugin generates codes in the same level directory as src, which is easy to commit by mistake. How to avoid it?

Answer:

Goto **Settings -> Languages & Frameworks -> ANTLR v4** default project settings and configure the output directory of the generated code as target/gen as shown:

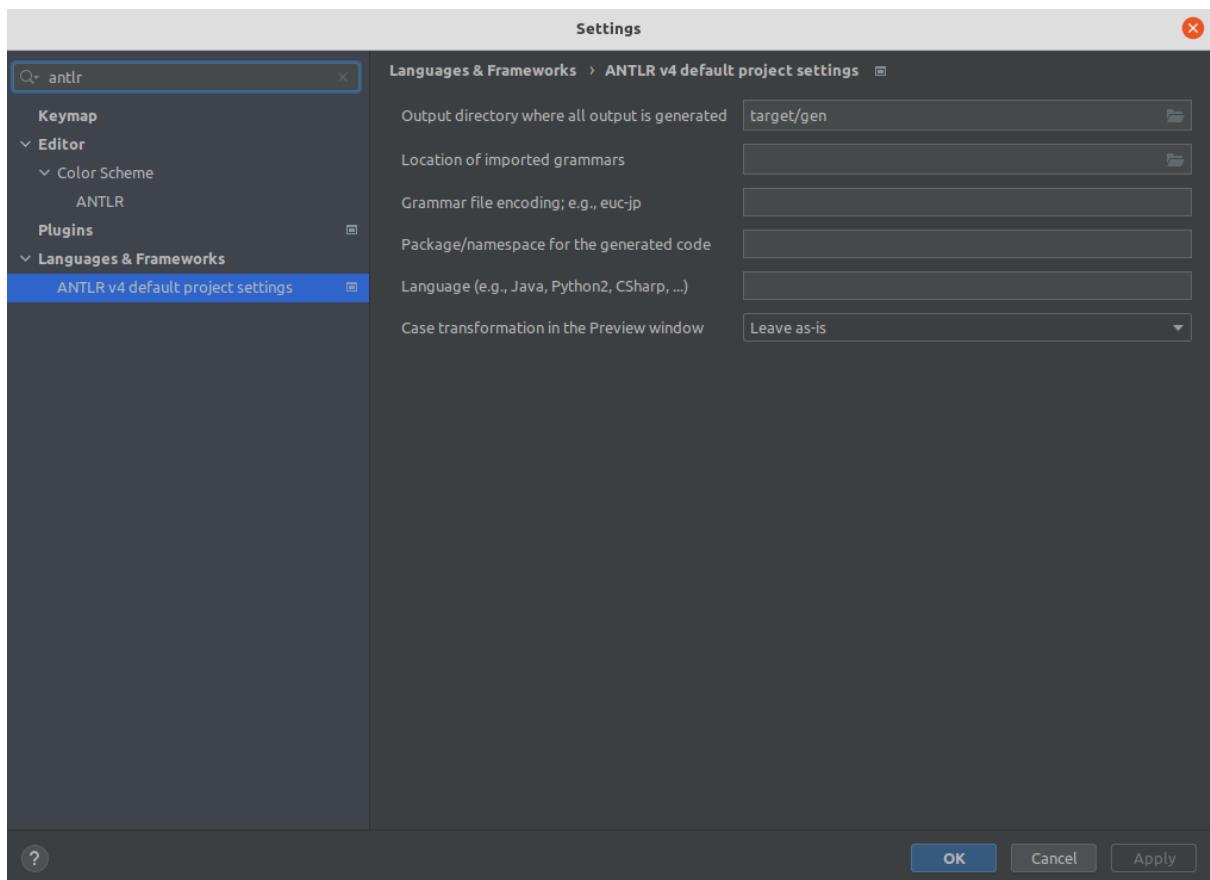


Figure27: Configure ANTLR plugin

10.8.29 [Other] Why is the database sharding result not correct when using Proxool?

Answer:

When using Proxool to configure multiple data sources, each one of them should be configured with alias. It is because Proxool would check whether existing alias is included in the connection pool or not when acquiring connections, so without alias, each connection will be acquired from the same data source.

The followings are core codes from ProxoolDataSource getConnection method in Proxool:

```
if(!ConnectionPoolManager.getInstance().isPoolExists(this.alias)) {
    this.registerPool();
}
```

For more alias usages, please refer to [Proxool](#) official website.

10.8.30 [Other] The property settings in the configuration file do not take effect when integrating ShardingSphere with Spring Boot 2.x ?

Answer:

Note that the property name in the Spring Boot 2.x environment is constrained to allow only lowercase letters, numbers and short transverse lines, [a-z] [0-9] and -.

Reasons:

In the Spring Boot 2.x environment, ShardingSphere binds the properties through Binder, and the unsatisfied property name (such as camel case or underscore.) can throw a `NullPointerException` exception when the property setting does not work to check the property value. Refer to the following error examples:

Underscore case: `database_inline`

```
spring.shardingsphere.rules.sharding.sharding-algorithms.database_inline.
type=INLINE
spring.shardingsphere.rules.sharding.sharding-algorithms.database_inline.props.
algorithm-expression=ds-$->{user_id % 2}
```

```
Caused by: org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'database_inline': Initialization of bean failed; nested exception is java.lang.NullPointerException: Inline sharding algorithm expression cannot be null.
```

```
Caused by: java.lang.NullPointerException: Inline sharding algorithm expression cannot be null.
```

```
    at com.google.common.base.Preconditions.checkNotNull(Preconditions.java:897)
    at org.apache.shardingsphere.sharding.algorithm.sharding.inline.
InlineShardingAlgorithm.getAlgorithmExpression(InlineShardingAlgorithm.java:58)
    at org.apache.shardingsphere.sharding.algorithm.sharding.inline.
InlineShardingAlgorithm.init(InlineShardingAlgorithm.java:52)
```

```

    at org.apache.shardingsphere.spring.boot.registry.
AbstractAlgorithmProvidedBeanRegistry.
postProcessAfterInitialization(AbstractAlgorithmProvidedBeanRegistry.java:98)
    at org.springframework.beans.factory.support.
AbstractAutowireCapableBeanFactory.
applyBeanPostProcessorsAfterInitialization(AbstractAutowireCapableBeanFactory.
java:431)
    ...

```

Camel case: databaseInline

```

spring.shardingsphere.rules.sharding.sharding-algorithms.databaseInline.type=INLINE
spring.shardingsphere.rules.sharding.sharding-algorithms.databaseInline.props.
algorithm-expression=ds-$->{user_id % 2}

```

Caused by: org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'databaseInline': Initialization of bean failed; nested exception is java.lang.NullPointerException: Inline sharding algorithm expression cannot be null.

```

    ...
Caused by: java.lang.NullPointerException: Inline sharding algorithm expression
cannot be null.
    at com.google.common.base.Preconditions.checkNotNull(Preconditions.java:897)
    at org.apache.shardingsphere.sharding.algorithm.sharding.inline.
InlineShardingAlgorithm.getAlgorithmExpression(InlineShardingAlgorithm.java:58)
    at org.apache.shardingsphere.sharding.algorithm.sharding.inline.
InlineShardingAlgorithm.init(InlineShardingAlgorithm.java:52)
    at org.apache.shardingsphere.spring.boot.registry.
AbstractAlgorithmProvidedBeanRegistry.
postProcessAfterInitialization(AbstractAlgorithmProvidedBeanRegistry.java:98)
    at org.springframework.beans.factory.support.
AbstractAutowireCapableBeanFactory.
applyBeanPostProcessorsAfterInitialization(AbstractAutowireCapableBeanFactory.
java:431)
    ...

```

From the exception stack, the `AbstractAlgorithmProvidedBeanRegistry.registerBean` method calls `PropertyUtil.containsPropertyPrefix (environment, prefix)`, and `PropertyUtil.containsPropertyPrefix (environment, prefix)` determines that the configuration of the specified prefix does not exist, while the method uses Binder in an unsatisfied property name (such as camelcase or underscore) causing property settings does not to take effect.

10.8.31 [ShardingSphere-JDBC] The tableName and columnName configured in yaml or properties leading incorrect result when loading Oracle metadata?

Answer:

Note that, in Oracle's metadata, the tableName and columnName is default UPPERCASE, while double-quoted such as CREATE TABLE "TableName"("Id" number) the tableName and columnName is the actual content double-quoted, refer to the following SQL for the reality in metadata:

```
SELECT OWNER, TABLE_NAME, COLUMN_NAME, DATA_TYPE FROM ALL_TAB_COLUMNS WHERE TABLE_NAME IN ('TableName')
```

The ShardingSphere uses the OracleTableMetaDataLoader to load the metadata, keep the tableName and columnName in the yaml or properties consistent with the metadata.

The ShardingSphere assembled the SQL using the following code:

```
private String getTableMetaDataSQL(final Collection<String> tables, final DatabaseMetaData metaData) throws SQLException {
    StringBuilder stringBuilder = new StringBuilder(28);
    if (versionContainsIdentityColumn(metaData)) {
        stringBuilder.append(", IDENTITY_COLUMN");
    }
    if (versionContainsCollation(metaData)) {
        stringBuilder.append(", COLLATION");
    }
    String collation = stringBuilder.toString();
    return tables.isEmpty() ? String.format(TABLE_META_DATA_SQL, collation)
                           : String.format(TABLE_META_DATA_SQL_IN_TABLES, collation, tables.
stream().map(each -> String.format("'%s'", each)).collect(Collectors.joining(", ")));
}
```

11

Downloads

11.1 Latest Releases

Apache ShardingSphere is released as source code tarballs with corresponding binary tarballs for convenience. The downloads are distributed via mirror sites and should be checked for tampering using GPG or SHA-512.

11.1.1 Apache ShardingSphere - Version: 5.1.2 (Release Date: June 17th, 2022)

- Source Codes: [SRC \(ASC, SHA512 \)](#)
- ShardingSphere-JDBC Binary Distribution: [TAR \(ASC, SHA512 \)](#)
- ShardingSphere-Proxy Binary Distribution: [TAR \(ASC, SHA512 \)](#)
- ShardingSphere-Agent Binary Distribution: [TAR \(ASC, SHA512 \)](#)

11.2 All Releases

Find all releases in the [Archive repository](#). Find all incubator releases in the [Archive incubator repository](#).

11.3 Verify the Releases

PGP signatures KEYS

It is essential that you verify the integrity of the downloaded files using the PGP or SHA signatures. The PGP signatures can be verified using GPG or PGP. Please download the KEYS as well as the asc signature files for relevant distribution. It is recommended to get these files from the main distribution directory and not from the mirrors.

```
gpg -i KEYS
```

or

```
pgpk -a KEYS
```

or

```
pgp -ka KEYS
```

To verify the binaries/sources you can download the relevant asc files for it from main distribution directory and follow the below guide.

```
gpg --verify apache-shardingsphere-*****.asc apache-shardingsphere-*****
```

or

```
pgpv apache-shardingsphere-*****.asc
```

or

```
pgp apache-shardingsphere-*****.asc
```