

How to achieve the scalability, high availability, and elastic ability of your database infrastructure on Kubernetes

Trista Pan
panjuan@apache.org

Trista Pan

SphereEx Co-Founder & CTO

Apache Member

AWS Data Hero

Tencent Cloud TVP

Apache ShardingSphere PMC

Apache brpc (Incubating) & Apache AGE

& Apache HugeGraph (Incubating) mentor

China Mulan Community Mentor



Bio: <https://tristazero.github.io>

LinkedIn: <https://www.linkedin.com/in/panjuan>

GitHub: <https://github.com/tristaZero>

Twitter: [@tristaZero](#)

Project Twitter: [@ShardingSphere](#)

Content

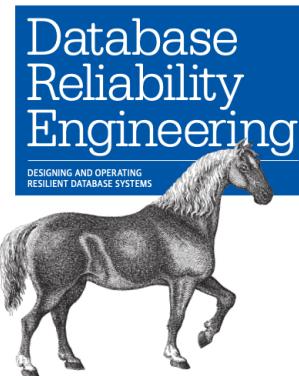
- ✓ SRE & SLA & DBRE
- ✓ The new needs for a database on the cloud
- ✓ Idea & architecture
- ✓ Handling SQL
- ✓ Demo

SRE & SLA & DBRE

- ✓ Database Reliability Engineering (DBRE) is basically a subset of Site Reliability Engineering (SRE)
- ✓ Stateless service VS stateful service (Persistence & status)
- ✓ SLA (Service Level Agreement) & SLO (Service Level Objectives) & SLI (Service Level Indicators)

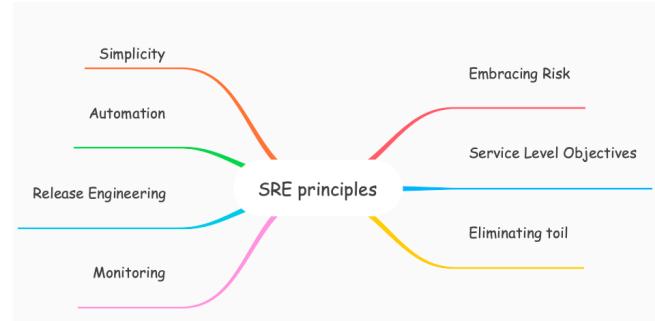


Edited by Betsy Beyer, Chris Jones,
Jennifer Petoff & Niall Richard Murphy



Laine Campbell & Charity Majors

New needs for databases



Eliminate the Barriers Between Software and Operations



Elimination of Toil

Databases Are Not Special Snowflakes(Cattle vs Pet)

Cost or Efficiency

Durability

Latency

Throughput

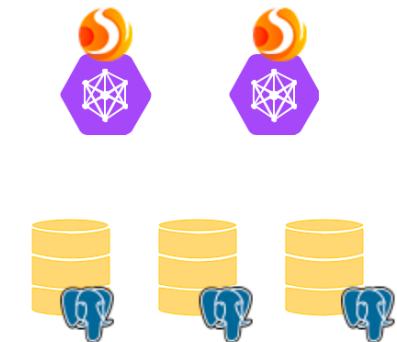
Availability

DBRE SLI

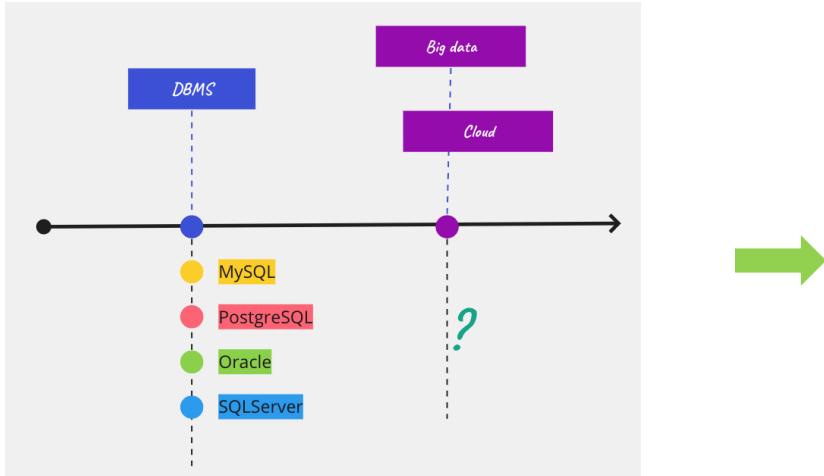
The needs for a database on the cloud

- ✓ Large data to manage
 - ✓ Efficient queries
 - ✓ Data security
 - ✓ Traffic governance
 - ✓ Elastic scaling
 - ✓ Backup & recovery
 - ✓ Metrics
 - ✓ Portability
 - ✓ Out-of-the-box deployment
- 
- Data Sharding
 - HA & read/write splitting & traffic strategy
 - Data Encryption
 - Monitor
 - Reshard for computing nodes and storage nodes
 - Helm & Operator on Kubernetes

Monolithic database on the cloud

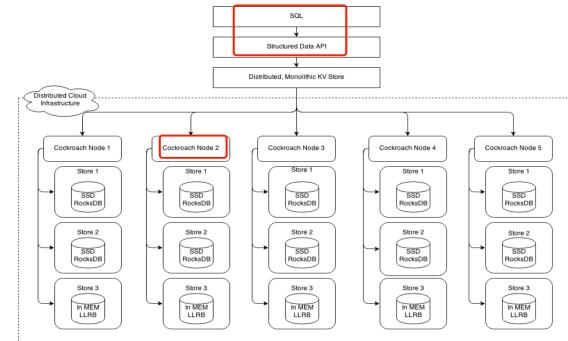
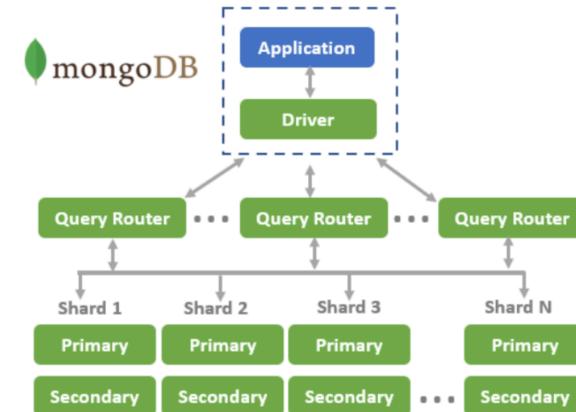
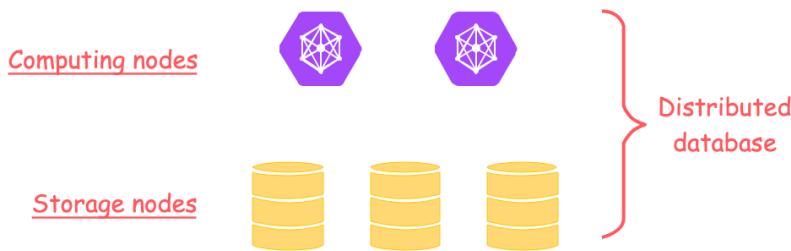


Benefits

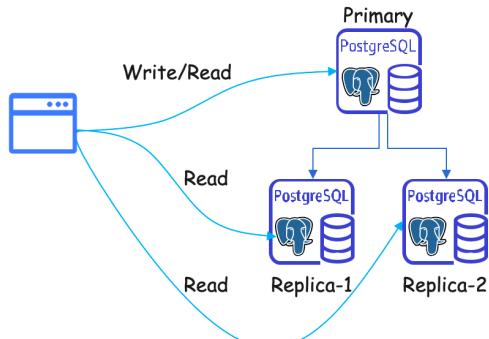


- ✓ Leverage the existing databases
- ✓ Upgrade it into a distributed database at low cost
- ✓ SQL audit & Traffic governance & Elastic scaling
- ✓ Solve the headache of moving database into Kubernetes
- ✓ Out-of-the-box deployment
- ✓ No lock-in

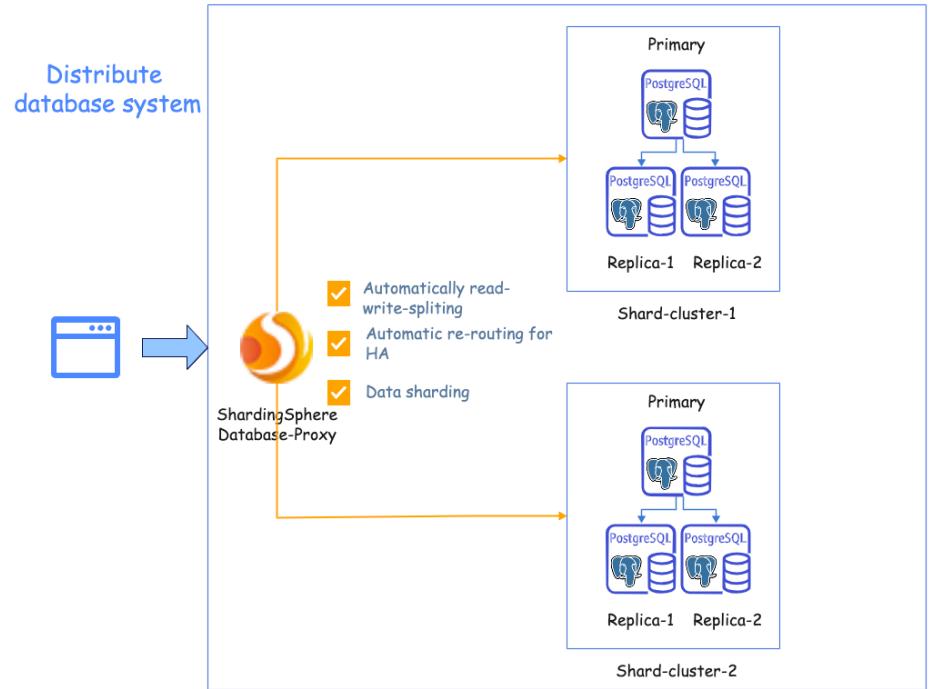
Distributed database



Application -> Database



Before



After

Apache ShardingSphere

The screenshot shows the GitHub repository page for Apache ShardingSphere. Key features highlighted include:

- Contributors:** 437 contributors.
- Environments:** 1 environment, specifically "github-pages" which is active.
- Tags:** mysql, sql, database, bigdata, postgresql, shard, rdbms, distributed-transactions, distributed-database, dba, encrypt, database-cluster, oltp, distributed-sql-database, database-plus.
- Statistics:** Forks: 6k, Stars: 17.3k.
- Links:** Readme, Apache-2.0 license, Code of conduct, 17.3k stars, 1k watching, 6k forks.

The screenshot shows the GitHub releases page for Apache ShardingSphere. Key features highlighted include:

- Releases:** 49 releases, with the latest being 5.2.0 released 6 days ago.
- Packages:** 1 package, specifically "shardingsphere-proxy".
- Contributors:** 437 contributors.
- Environments:** 1 environment, specifically "github-pages" which is active.

What is Apache ShardingSphere?

The ecosystem to transform any database into a distributed database system, and enhance it with sharding, elastic scaling, encryption features & more.

[Download](#)[Learn More](#)[Academic Publications](#)

The screenshot shows the Apache ShardingSphere official website's "Overview" page. Key features highlighted include:

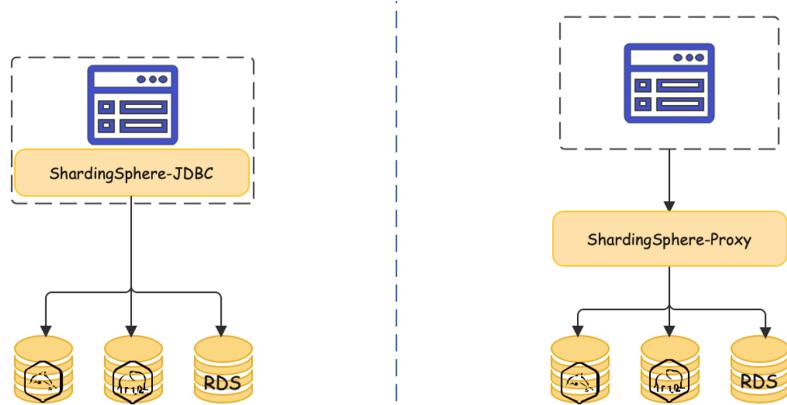
- Table of Contents:** 1. Overview, 2. Quick Start, 3. Features, 4. User Manual, 5. Dev Manual, 6. Test Manual, 7. Reference, 8. FAQ, 9. Downloads.
- Introduction:** This chapter mainly introduces what Apache ShardingSphere is, as well as its design philosophy and deployment architecture.
- What is ShardingSphere:** Frequently asked questions, please refer to FAQ.
- ShardingSphere-JDBC:** A lightweight Java framework that provides additional services at Java's JDBC layer.
- Design Philosophy:** Connect: Create database upper level standard, Enhance: Database computing enhancement engine, Pluggable: Building database function ecology.
- Deployment:** Deployment, Running Modes.

ShardingSphere clients

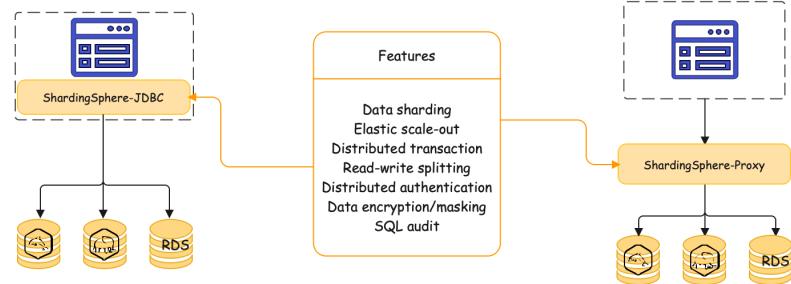
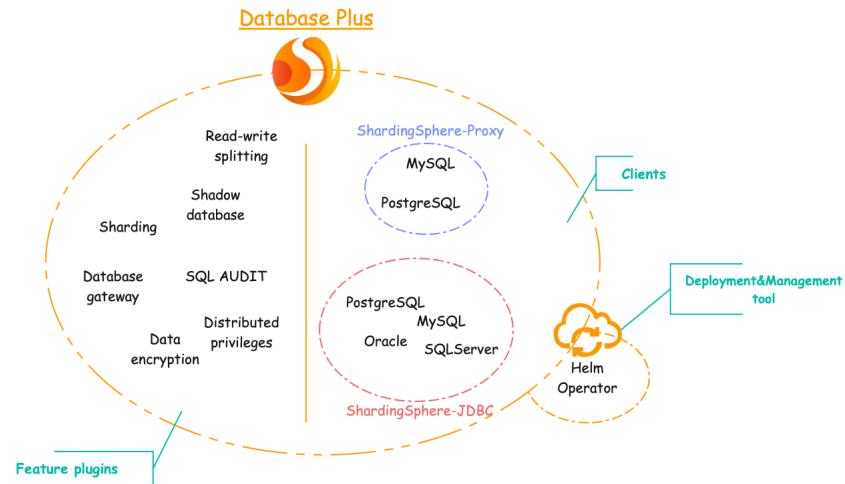
Database Plus

What is Apache ShardingSphere?

The ecosystem to transform any database into a distributed database system, and enhance it with sharding, elastic scaling, encryption features & more.



ShardingSphere features



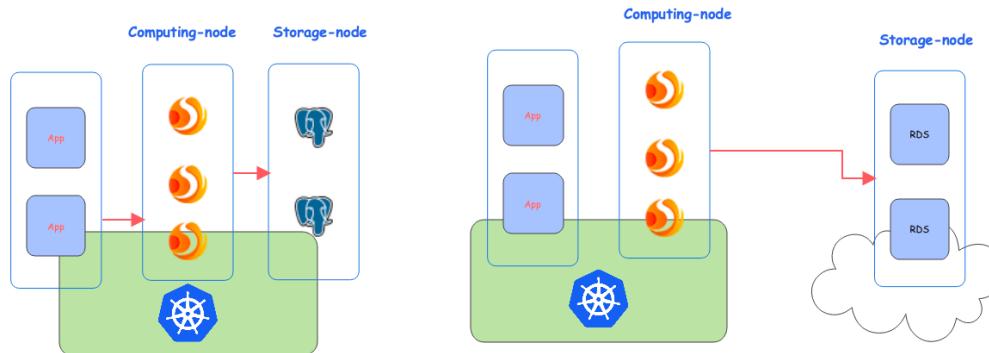
ShardingSphere on Cloud

ShardingSphere-on-Cloud

Take Apache ShardingSphere to the cloud

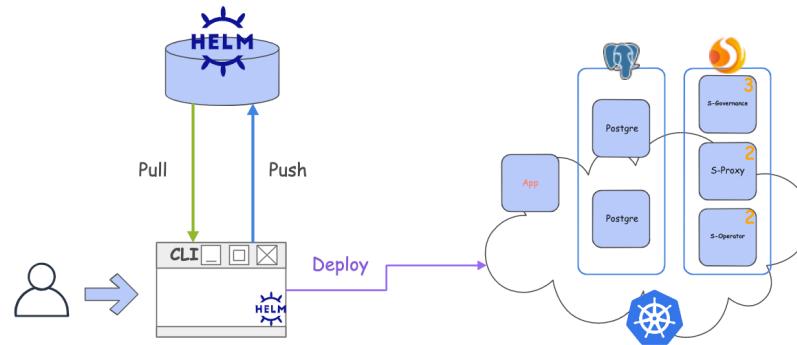
A collection of tools & best practices including automated deployment scripts to virtual machines in AWS, Google Cloud Platform, Alibaba Cloud, CloudFormation Stack templates, and Terraform one-click deployment scripts.

Helm Charts, Operators, automatic horizontal scaling, and other tools for the Kubernetes cloud-native environment are also included.

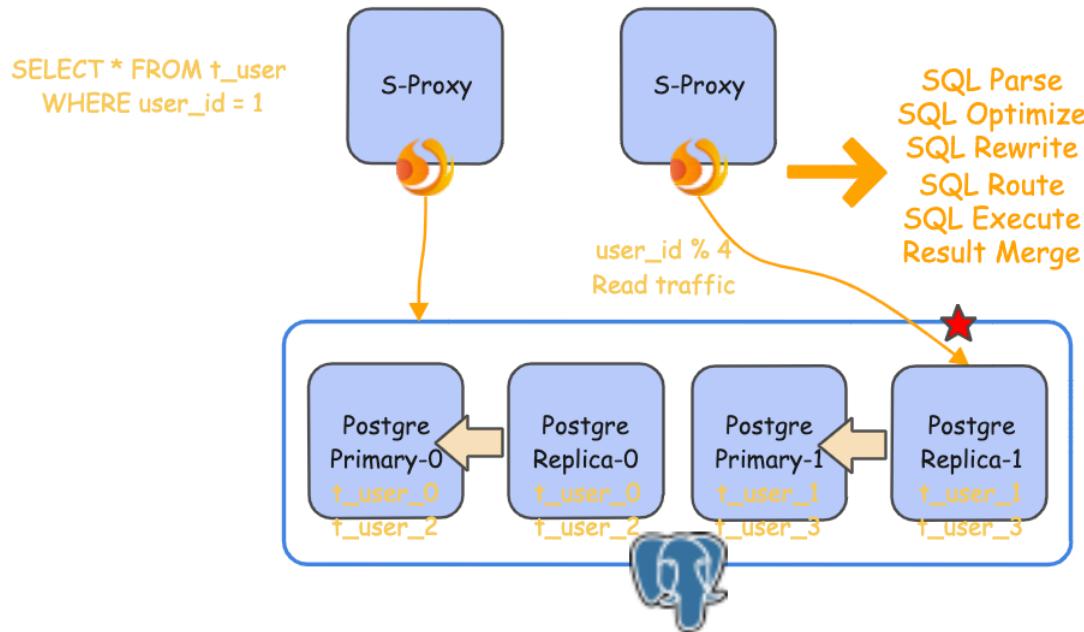


Demo

- ShardingSphere-Chart → Deployment & Upgrade of ShardingSphere
- ShardingSphere-Operator-Chart → HA & Elastic scale-out base CPU metric of ShardingSphere
- PostgreSQL-Chart → Deployment of PostgreSQL



The handling process of one SQL



The demo show

1. Deploy two PostgreSQL (Storage node) clusters made of a primary node and a replica
2. Deploy two ShardingSphere-Proxy (Computing node) and ShardingSphere-governance
3. Add PostgreSQL resources and their relationship into ShardingSphere-Proxy
4. Create sharding table t_user on ShardingSphere-Proxy
5. Show the metadata of this distributed database system
6. INSERT data for test on ShardingSphere-Proxy
7. Preview SELECT routing result
8. Execute SELECT query

Step 1, 2,

```
git clone https://github.com/apache/shardingsphere-on-cloud
```

```
cd charts/shardingsphere-operator-cluster
```

```
helm dependency build
```

```
helm install shardingsphere-cluster shardingsphere-operator-cluster -n sharding-test
```

Pods					
Name	Namespace	Containers	Restarts	Controlled By	
shardingsphere-cluster-apach...	shardingsphere-cluster-apache-shardingsphere-proxy-charts-irv5			ReplicaSet	
shardingsphere-cluster-apach...	sharding-test		0	ReplicaSet	
shardingsphere-cluster-zooke...	sharding-test		0	StatefulSet	
shardingsphere-cluster-zooke...	sharding-test		1	StatefulSet	
shardingsphere-cluster-zooke...	sharding-test		0	StatefulSet	

```
tristan@Tristas-MacPro ~ % helm install pg-cluster-0 bitnami/postgresql -n ss-new --set global.storageClass=sata-csi-udisk --set replication.n  
umSynchronousReplicas=1 --set readReplicas.replicaCount=1 --set architecture=replication
```

```
NAME: pg-cluster-0  
LAST DEPLOYED: Thu Jul 14 12:13:26 2022  
NAMESPACE: ss-new  
STATUS: deployed  
REVISION: 1  
TEST SUITE: None  
NOTES:  
CHART NAME: postgresql  
CHART VERSION: 11.6.16  
APP VERSION: 14.4.0
```

Services						
Name	Namespace	Type	Cluster IP	Ports	External IP	Selector
shardingsphere-c...	sharding-test	ClusterIP	172.17.228.23	3307/TCP	-	app=shard
shardingsphere-c...	sharding-test	ClusterIP	172.17.96.233	2181/client/TCP, ...	-	app.kuber
shardingsphere-c...	sharding-test	ClusterIP	None	2181/client/TCP, ...	-	app.kuber

```
tristan@Tristas-MacPro ~ % helm install pg-cluster-1 bitnami/postgresql -n ss-new --set global.storageClass=sata-csi-udisk --set replication.n  
umSynchronousReplicas=1 --set readReplicas.replicaCount=1 --set architecture=replication
```

```
NAME: pg-cluster-1  
LAST DEPLOYED: Thu Jul 14 12:13:48 2022  
NAMESPACE: ss-new  
STATUS: deployed  
REVISION: 1  
TEST SUITE: None  
NOTES:  
CHART NAME: postgresql  
CHART VERSION: 11.6.16  
APP VERSION: 14.4.0
```

Step 3, 4, 5

```
psql (14.2, server 12.3 SphereEx-DBPlusEngine-Proxy 1.1.0)
Type "help" for help.
```

```
postgres=> CREATE DATABASE sharding_rw_splitting_db;
CREATE DATABASE
```

```
postgres=> ADD RESOURCE write_ds_0 (
    HOST=127.0.0.1,
    PORT=5430,
    DB=sharding_rw_splitting_db,
    USER=postgres,
    PASSWORD=x0xJ1jSIbN
), read_ds_0 (
    HOST=127.0.0.1,
    PORT=5431,
    DB=sharding_rw_splitting_db,
    USER=postgres,
    PASSWORD=x0xJ1jSIbN
),write_ds_1 (
    HOST=127.0.0.1,
    PORT=5432,
    DB=sharding_rw_splitting_db,
    USER=postgres,
    PASSWORD=RHVdPNbsyK
), read_ds_1 (
    HOST=127.0.0.1,
    PORT=5433,
    DB=sharding_rw_splitting_db,
    USER=postgres,
    PASSWORD=RHVdPNbsyK
);
SUCCESS
```

```
postgres=>
postgres=> CREATE READWRITE_SPLITTING RULE rw_group_0 (
    WRITE_RESOURCE=write_ds_0,
    READ_RESOURCES(read_ds_0),
    TYPE(NAME=random)
);
SUCCESS
```

```
postgres=> CREATE READWRITE_SPLITTING RULE rw_group_1 (
    WRITE_RESOURCE=read_ds_1,
    READ_RESOURCES(read_ds_1),
    TYPE(NAME=random)
);
SUCCESS
```

```
sharding_rw_splitting_db=> CREATE SHARDING TABLE RULE t_user (
    RESOURCES(rw_group_0,rw_group_1),
    SHARDING_COLUMN=user_id,TYPE(NAME=mod,PROPERTIES("sharding-count"=4)))
);
SUCCESS
```

```
postgres=>
postgres=> CREATE TABLE t_user (
    user_id int4,
    user_name varchar(32),
    tel varchar(32)
);
CREATE TABLE
postgres=>
```

```
sharding_rw_splitting_db=> SHOW SHARDING TABLE NODES;
   name   |          nodes
-----+-----
   t_user | rw_group_0.t_user_0, rw_group_1.t_user_1, rw_group_0.t_user_2, rw_group_1.t_user_3
(1 row)
```

Step 6, 7, 8

```
postgres=>
postgres=> INSERT INTO t_user values (1,'name1','tel11111');
INSERT INTO t_user values (2,'name2','tel22222');
INSERT INTO t_user values (3,'name3','tel33333');
INSERT INTO t_user values (4,'name4','tel44444');
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
```

```
sharding_rw_splitting_db=> PREVIEW SELECT * FROM t_user WHERE user_id=1;
data_source_name | actual_sql
-----+-----
read_ds_1 | SELECT * FROM t_user_1 WHERE user_id=1
(1 row)
```

```
sharding_rw_splitting_db=>
sharding_rw_splitting_db=> SELECT * FROM t_user WHERE user_id=1;
user_id | user_name | tel
-----+-----+-----
1 | name1 | tel11111
(1 row)
```

```
sharding_rw_splitting_db=>
sharding_rw_splitting_db=> PREVIEW SELECT * FROM t_user;
data_source_name | actual_sql
-----+-----
read_ds_0 | SELECT * FROM t_user_0 UNION ALL SELECT * FROM t_user_2
read_ds_1 | SELECT * FROM t_user_1 UNION ALL SELECT * FROM t_user_3
(2 rows)
```

```
sharding_rw_splitting_db=> SELECT * FROM t_user ORDER BY user_id;
user_id | user_name | tel
-----+-----+-----
1 | name1 | tel11111
2 | name2 | tel22222
3 | name3 | tel33333
4 | name4 | tel44444
(4 rows)

sharding_rw_splitting_db=>
```

Thanks! Any questions?

Bio: <https://tristazero.github.io>

LinkedIn: <https://www.linkedin.com/in/panjuan>

GitHub: <https://github.com/tristaZero>

Twitter: @tristaZero

Project Twitter: @ShardingSphere