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

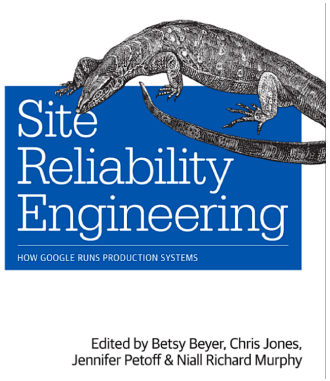
SphereEx

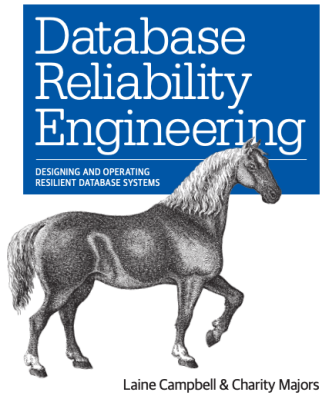
Content

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

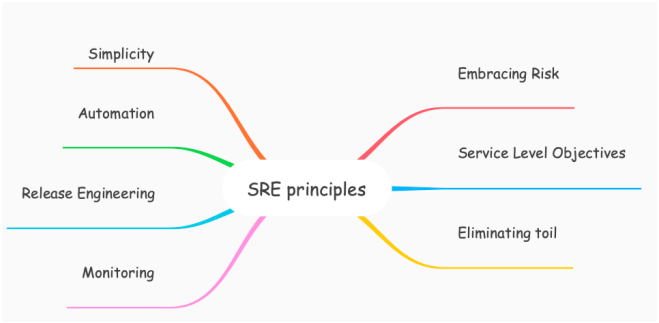
SRE & SLA & DBRE

- ✓ 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)





New needs for databases



SphereEx

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

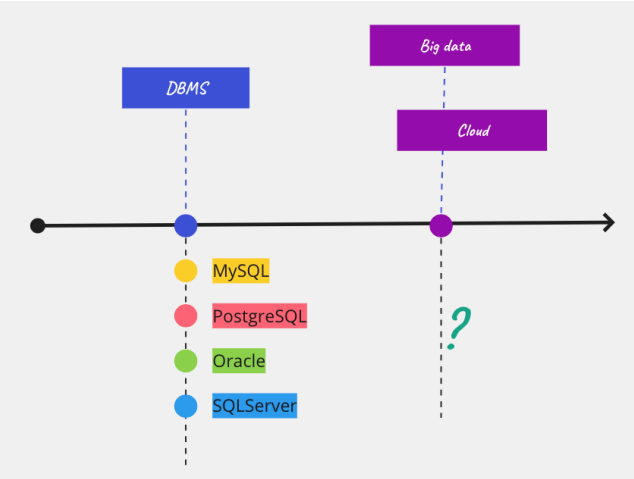
Monitor

Reshard for computing nodes and storage nodes

Helm & Operator on Kubernetes

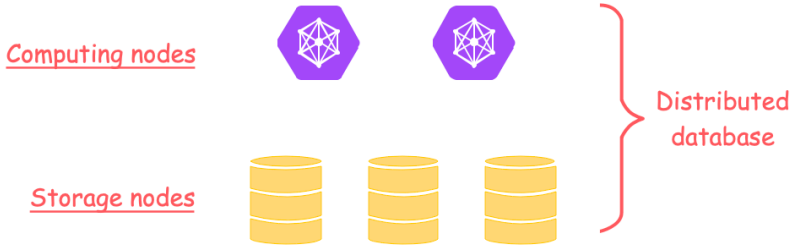
Monolithic database on the cloud

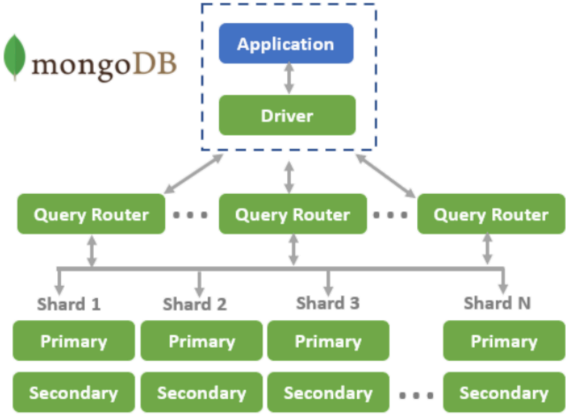
Benefits

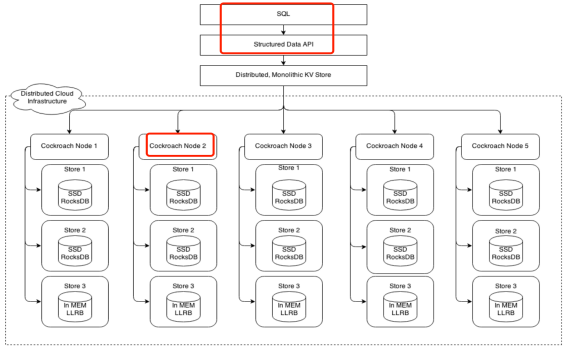


- ✓ Leverage the existing databases
- ✓ Upgrade it into a distributed database at low cost
- ✓ SQL audit & Traffic governance & Elastic scaling
- ✓ Solve the headache of moving database into Kubernetes
- ✓ Out-of-the-box deployment
- ✓ No lock-in

Distributed database

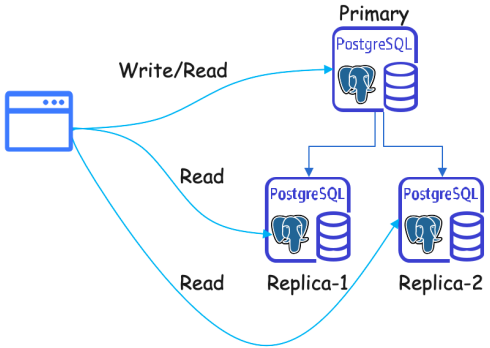




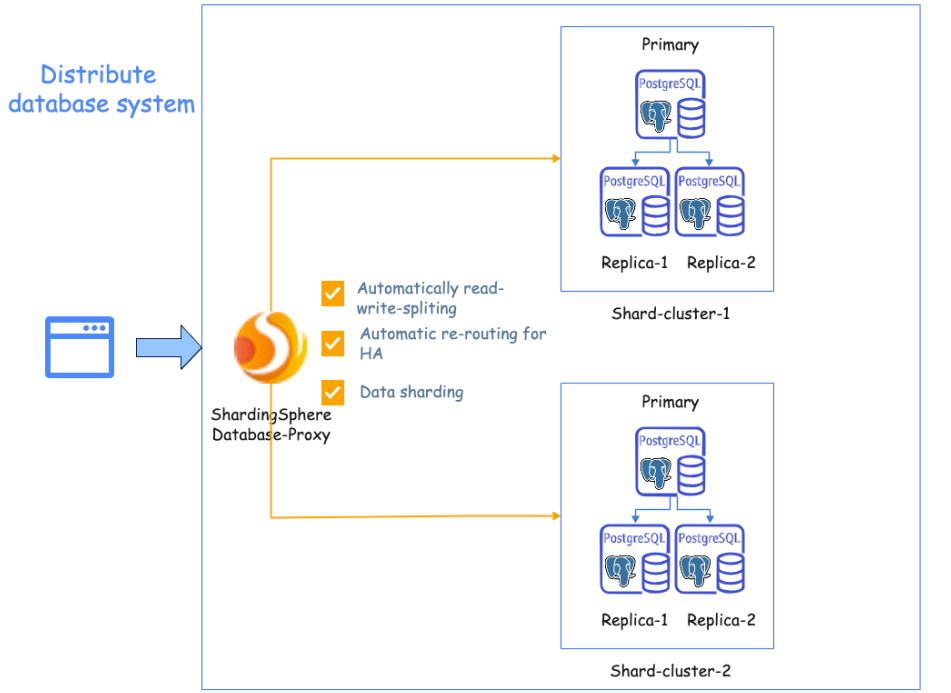


SphereEx

Application -> Database



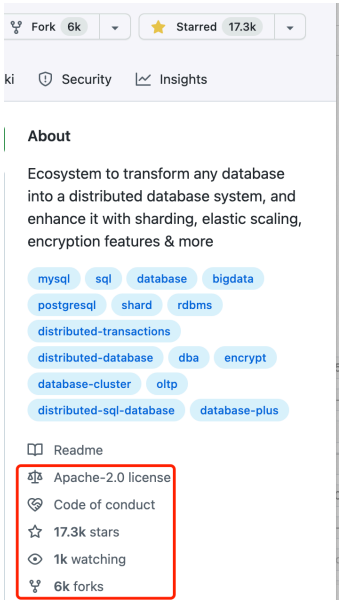
Before

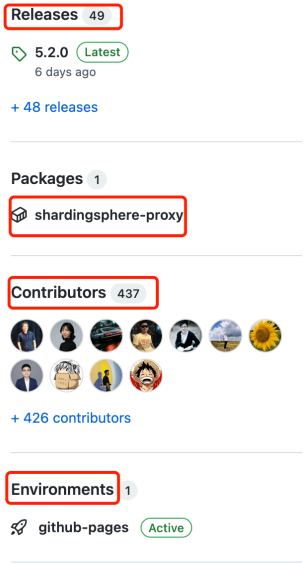


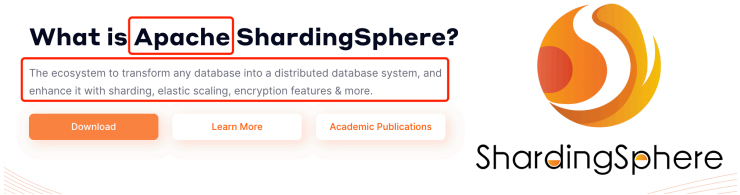
After

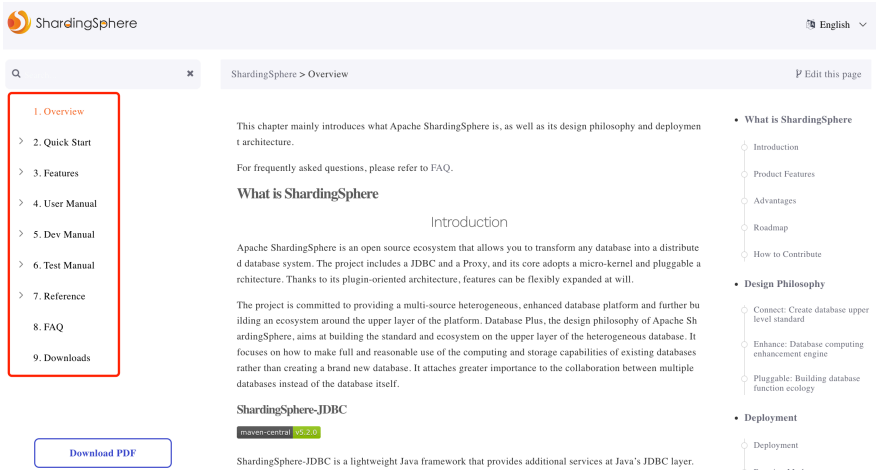
SphereEx

Apache ShardingSphere









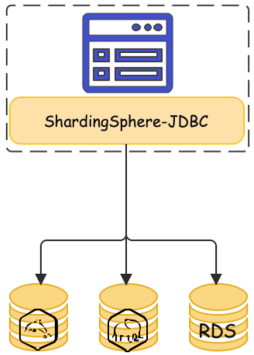
SphereEx

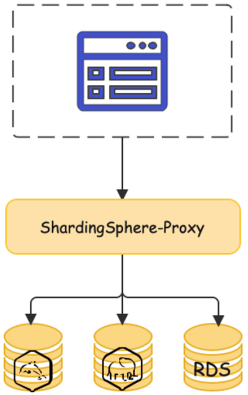
ShardingSphere clients

Database Plus

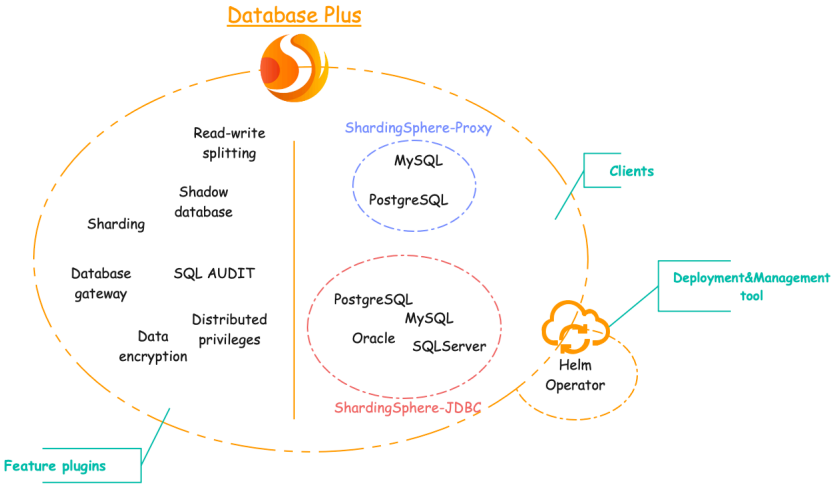
What is Apache ShardingSphere?

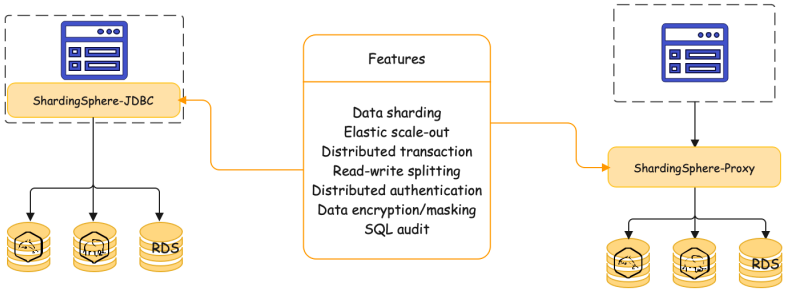
The ecosystem to transform any database into a distributed database system, and enhance it with sharding, elastic scaling, encryption features & more.





ShardingSphere features





SphereEx

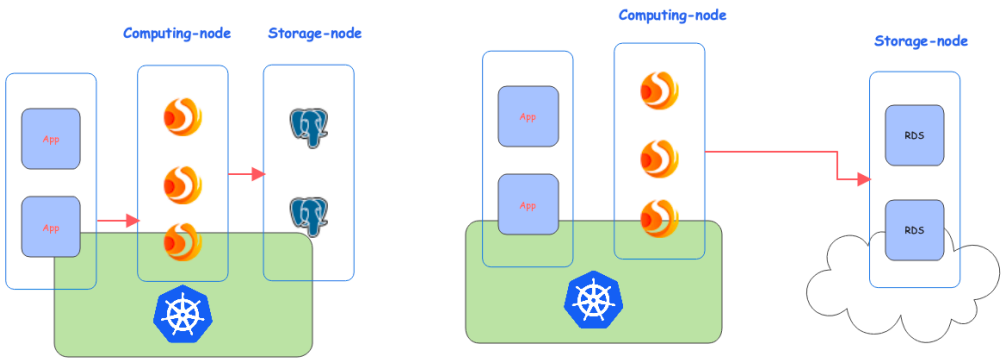
ShardingSphere on Cloud

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.



https://shardingsphere.apache.org/oncloud/

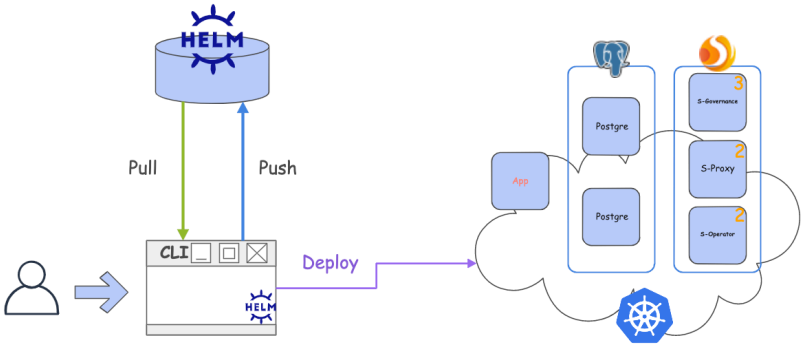
SphereEx

Demo

ShardingSphere-Chart → Deployment & Upgrate of ShardingSphere

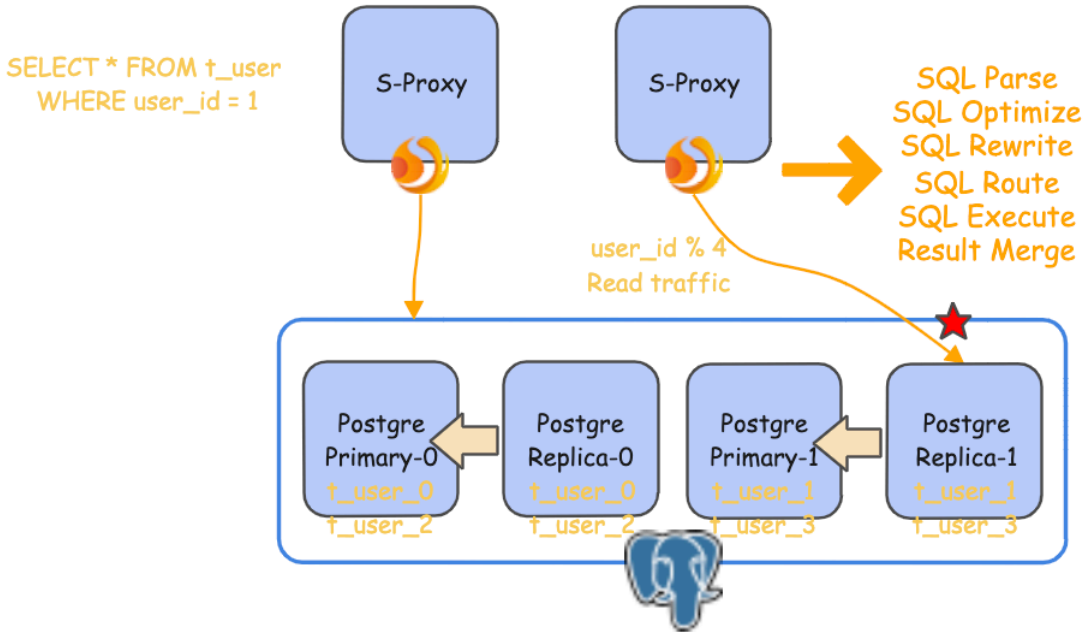
ShardingSphere-Operator-Chart → HA & Elastic scale-out base CPU metric of ShardingSphere

PostgreSQL-Chart → Deployment of PostgreSQL



https://github.com/apache/shardingsphere-on-cloud

The handling process of one SQL

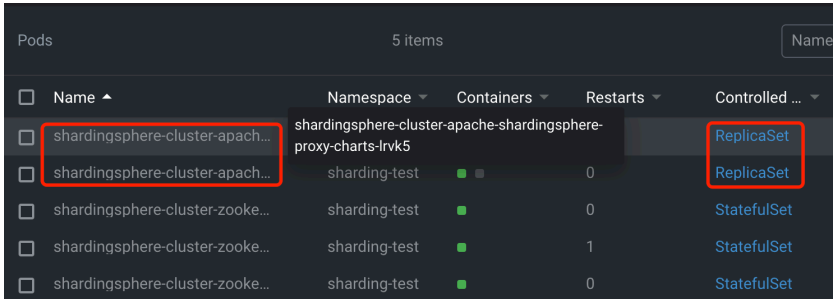


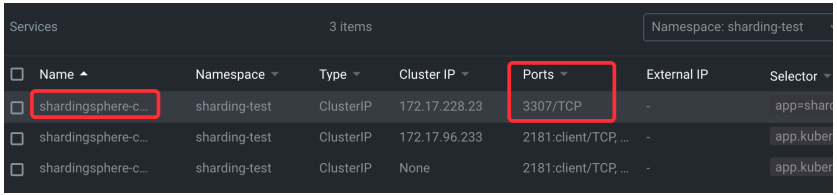
The demo show

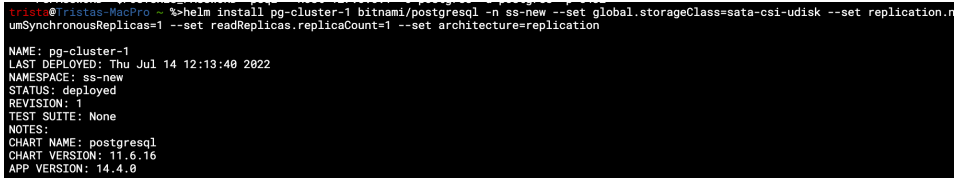
1. Deploy two PostgreSQL (Storage node) clusters made of a primary node and a replica
2. Deploy two ShardingSphere-Proxy (Computing node) and ShardingSphere-governance
3. Add PostgreSQL resources and their relationship into ShardingSphere-Proxy
4. Create sharding table t_user on ShardingSphere-Proxy
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







SphereEx

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=xOxJ1jSIbN
), read_ds_0 (
    HOST=127.0.0.1,
    PORT=5431,
    DB=sharding_rw_splitting_db,
    USER=postgres,
    PASSWORD=xOxJ1jSIbN
),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=write_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)
```

SphereEx

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=>
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=> 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=> 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=>
```

```
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)
```

SphereEx

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

SphereEx