

IT云服务能力平台测试规范

分布式数据库分册

（1.0版）

中国电信企业信息化事业部

文档历史

| **版本号** | **修改版日期** | **修改内容** | **修改人员** |
| --- | --- | --- | --- |
| 1.0 | 2017.01.13 | 佛山集中评审后的修订整合版本 | 刘秀华  杨创  陈阳关  张家宁  吴桐  王振宇 |
| 1.1.0 | 2017.02.03 | 根据各省公司反馈意见修订版本 | 刘秀华 |
| 1.1.1 | 2017.02.17 | 修改描述不清晰不合理的文字 | 刘秀华 |
| 1.1.2 | 2017.03.07 | 修改描述错误和不准备的文字 | 刘秀华 |

目录

[1 综述 6](#_Toc475119095)

[1.1 应用场景和价值 6](#_Toc475119096)

[1.1.1 产品价值 6](#_Toc475119097)

[1.1.2 应用场景 7](#_Toc475119098)

[1.2 技术概述和主流产品 7](#_Toc475119099)

[1.3 术语解释 8](#_Toc475119100)

[2 技术指标 9](#_Toc475119101)

[2.1 技术要求 9](#_Toc475119102)

[2.2 刚性要求 10](#_Toc475119103)

[2.3 基础功能 11](#_Toc475119104)

[2.4 可用性 12](#_Toc475119105)

[2.5 可靠性 12](#_Toc475119106)

[2.6 监控/运维 12](#_Toc475119107)

[2.7 安全性 13](#_Toc475119108)

[2.8 性能 13](#_Toc475119109)

[3 测试准备 14](#_Toc475119110)

[3.1 产品文档 14](#_Toc475119111)

[3.2 测试代码 14](#_Toc475119112)

[3.3 测试环境资源 14](#_Toc475119113)

[3.3.1 部署示意图 14](#_Toc475119114)

[3.3.2 硬件配置 15](#_Toc475119115)

[3.3.3 数据准备 16](#_Toc475119116)

[4 测试用例 18](#_Toc475119117)

[4.1 基础功能 18](#_Toc475119118)

[4.1.1 安装部署 18](#_Toc475119119)

[4.1.2 建库建表（DDL SQL） 19](#_Toc475119120)

[4.1.3 表结构修改（DDL SQL） 20](#_Toc475119121)

[4.1.4 数据分片 22](#_Toc475119122)

[4.1.5 数据访问 25](#_Toc475119123)

[4.1.6 DML SQL 29](#_Toc475119124)

[4.1.7 关键字/汇聚函数 40](#_Toc475119125)

[4.1.8 单库事务支持 45](#_Toc475119126)

[4.1.9 全局序列 49](#_Toc475119127)

[4.1.10 分片索引 50](#_Toc475119128)

[4.1.11 全局表 53](#_Toc475119129)

[4.1.12 库内分表 56](#_Toc475119130)

[4.1.13 自定义分片数 57](#_Toc475119131)

[4.1.14 Hint语法/透传 58](#_Toc475119132)

[4.1.15 读写分离 59](#_Toc475119133)

[4.2 可用性 60](#_Toc475119134)

[4.2.1 数据库中间件集群高可用 60](#_Toc475119135)

[4.2.2 数据库集群高可用 61](#_Toc475119136)

[4.2.3 数据库中间件节点异常恢复 61](#_Toc475119137)

[4.2.4 数据库节点异常恢复 62](#_Toc475119138)

[4.2.5 过载保护 63](#_Toc475119139)

[4.2.6 数据库中间件集群分组 63](#_Toc475119140)

[4.3 可靠性 64](#_Toc475119141)

[4.3.1 从库网络断开 64](#_Toc475119142)

[4.3.2 主库网络断开 65](#_Toc475119143)

[4.3.3 主库数据库实例shutdown 66](#_Toc475119144)

[4.3.4 主库服务器crash 67](#_Toc475119145)

[4.3.5 疲劳测试 68](#_Toc475119146)

[4.3.6 分布式事务测试（正常情形） 69](#_Toc475119147)

[4.3.7 分布式事务测试（异常情形） 70](#_Toc475119148)

[4.4 监控/运维 72](#_Toc475119149)

[4.4.1 全局统一管理配置 72](#_Toc475119150)

[4.4.2 实时监控 72](#_Toc475119151)

[4.4.3 在线扩容 73](#_Toc475119152)

[4.4.4 组件灰度发布 73](#_Toc475119153)

[4.4.5 在线配置修改 74](#_Toc475119154)

[4.4.6 在线建表、建全局序列 75](#_Toc475119155)

[4.4.7 管理命令 75](#_Toc475119156)

[4.4.8 统计分析 76](#_Toc475119157)

[4.4.9 扩展DDL语法测试 77](#_Toc475119158)

[4.4.10 一键故障检测 77](#_Toc475119159)

[4.4.11 数据备份/恢复 78](#_Toc475119160)

[4.5 安全性 81](#_Toc475119161)

[4.5.1 权限验证 81](#_Toc475119162)

[4.5.2 账号有效期和密码过期时间测试 82](#_Toc475119163)

[4.5.3 SQL拦截 82](#_Toc475119164)

[4.5.4 黑白名单 83](#_Toc475119165)

[4.6 性能 84](#_Toc475119166)

[4.6.1 CRUD/综合语句性能（单数据库中间件+单数据库） 84](#_Toc475119167)

[4.6.2 CRUD/综合语句性能—线性扩展能力1 85](#_Toc475119168)

[4.6.3 CRUD/综合语句性能—线性扩展能力2 85](#_Toc475119169)

[4.6.4 集成场景综合测试性能 86](#_Toc475119170)

[4.6.5 全局序列获取性能 87](#_Toc475119171)

1. 综述

## 应用场景和价值

本产品是一种兼容主流数据库协议和 SQL92标准语法，支持自动水平拆分的高性能、高可靠分布式关系型数据库；数据访问对应用透明，每个分片默认采用主备架构，提供灾备、恢复、监控、不停机扩容等整套解决方案，适用于TB或PB级的海量数据场景。



图1.1-1产品部署示意图

### 产品价值

* 突破单机数据库存储容量和性能瓶颈
* 解决业务互联网化带来的峰值流量访问问题
* 使用X86 PC+廉价存储+开源软件及自研结合的“去IOE”模式 ，节约企业成本

### 应用场景

* **交易型应用**

大并发、大数据量、以联机事务处理为主的交易型应用，如电商、金融、O2O、电信CRM/计费等，服务能力支持线性增长、弹性扩缩，应用可按需选择不同的部署规模。

* **物联网数据**

在工业监控和远程控制、智慧城市的延展、智能家居、车联网等物联网场景下，传感监控设备多，采样率高，数据存储要求高，超大数据规模存储的问题。

* **海量数据存储**

存储海量结构化历史数据，提供给大数据分析使用，解决非结构化数据库难以表达和存储复杂结构化数据的问题。而结构化关系型数据是大数据分析中最有价值的数据之一。

* 1. 技术概述和主流产品

目前开源分布式关系型数据库的产品主要有阿里的Cobar、MyCat、360的Atlas等。阿里和360等大的互联网公司的开源软件一般都存在开源不彻底，文档不完善的问题，生产应用的版本与开源版本差异很大，如：阿里的Cobar在2012年6月正式开源，在13年10月之后基本上没再更新了，阿里内部生产系统用的也不是Cobar，阿里云主推的是DRDS；而由个人组织的开源软件MyCat，虽然目前社区较活跃，但存在发展方向不受约束且进度不可控等不确定性，另外版本发布不够严谨，未经过严格的功能、性能和高可用性测试，且在同行业中尚无成熟应用案例等问题。开源的分布式关系型数据库普遍存在以下一些问题：

* 可靠性较低：存在单点故障，无自动主备切换功能，无法保障数据服务的高可用及数据存储的高可靠
* 可维护性差：无自动化管理界面，缺少有效运维监控手段，故障支撑依赖开源社区，风险高
* 功能不完善：针对电信业务特点的复杂关系型业务场景的功能缺失，如：区域分片算法、库内分表、切片索引、事务支持等

|  |  |  |
| --- | --- | --- |
| **对比项** | **COBAR** | **MYCAT** |
| 开源协议 | APACHE | APACHE |
| 开发语言 | JAVA | JAVA |
| 高可用 | 具备，但比较初级和原始，缺陷比较多 | 具备，但比较初级和原始，缺陷比较多 |
| 线性扩展 | 集群模式具备线性扩展能力 | 集群模式具备线性扩展能力 |
| 社区活跃度 | 已经停止更新 | 活跃，较为强大的社区支持 |
| 开源生态圈 | 由于项目停更，生态圈的发展呈停滞状态 | 近年发展较快，在国内多个项目中屡有应用，社区上有一支由软件工程师、架构师、DBA等组成的较为强大的技术团队 |
| 主要技术优势 | 1 通过分库分表的方式实现分布式和水平扩展 | 1继承了Cobar的稳定性、可靠性、优秀的架构和性能  2提供了全局序列，读写分离、后端连接NIO等新功能特性 |
| 主要技术劣势 | 1不支持读写分离，序列等特性  2后端连接BIO模式，存在系统阻塞的可能  3 节点假死会导致主从自动切换后，发生数据库主从节点都有写操作  4 表数据分片算法单一 | 1分布式事务采用BATCH COMMIT的做法，只支持弱一致性  2非分片键查询时只能广播，性能较低  3 不支持跨分片JOIN操作 |

## 术语解释

|  |  |
| --- | --- |
| **术语名称** | **定义描述** |
| 数据库中间件 | 介于数据库与应用之间，进行数据处理与交互的中间服务。 |
| 数据分片 | 通过某种特定条件，将原先存放在同一数据库中的数据分散存放到多个数据库（主机）上面，从而达到分散单台设备负载的效果。 |
| 逻辑库 | 多个物理数据库分片（database）的集合，逻辑库中定义了所包含的逻辑表、分片规则、全局序列及分片索引。 |
| 全局序列 | 为应用提供唯一序列号，主要用于替换单机数据库的自增序列，保障分布式架构下数据库表主键全局唯一。 |
| 分片索引 | 不同于数据库表索引，分片索引专门用于记录索引列值与分片键值的对应关系，从而快速定位包含索引列值的记录所在数据分片，避免广播式查询，提升查询性能。 |
| SLB | 服务器负载均衡（Server Load Balancing） |
| DDL语句 | Data Definition Language 数据定义语句，定义了不同的数据段、数据库、表、列、索引等数据库对象。常用的语句关键字主要包括create、drop、alter等 |
| DML语句 | Data Manipulation Language 数据操纵语言，用于添加、删除、更新和查询数据库记录，并检查数据完整性。常用的语句关键字主要包括insert、delete、update和select等 |
| DCL语句 | Data Control Language 数据控制语句，用于控制不同数据段直接的许可和访问级别的语句。这些语句定义了数据库、表、字段、用户的访问权限和安全级别。主要的语句关键字包括grant、revoke等 |

1. 技术指标

## 技术要求

* **扩展性：**
* 自动水平拆分，支持字符串，数字，日期等多种拆分纬度
* 支持业务不中断平滑扩缩容
* **应用开发：**
* 数据访问对应用透明，支持JDBC驱动，支持业界或开源标准协议（如：MySQL 协议、Oracle协议等）
* 支持SQL 92标准
* 兼容常用数据库客户端
* **可用性：**
* 集群可用性超过99.99%
* 分布式架构无服务单点，服务稳定
* 故障隔离，具备过载保护机制，防止故障蔓延
* 具备故障自动恢复功能，如：网络中断重连
* 支持数据库主备故障自动切换，数据库主备发生切换时，数据库中间件能迅速响应（秒级）进行切换
* **可靠性：**
* 数据库主从节点的数据在异常场景下仍能保持100%一致
* 对外提供一致性的访问接口，屏蔽内部因为节点故障等原因造成的数据存储位置不确定性
* 支持分布式事务，保障分布式环境下的数据一致性
* **性能：**
* 对比单机数据库服务能力线性提升，线性系数 > 0.7
* 支持分片索引，减少广播查询
* 支持全局表同步功能，提升全局表关联查询效率
* 数据库中间件单节点写入性能大于9w TPS（4路8核/64GB内存）
* 支持应用透明读写分离和应用级读写分离方案，可根据业务需要灵活调整，有效提高读性能
* **监控/运维：**
* 提供配置、发布、监控、运维一体化WEB管理平台
* 支持灰度发布
* 提供完善日志监控和运维分析工具，快速定位及解决问题
* 提供一键数据库扩容功能（分片数不变）
* 提供统计分析功能，为应用提供合理的数据切分及分片索引建议
* 具备完善的数据备份/恢复机制
* **安全：**
* 数据库中间件具备完善的权限控制体系，可设置用户账号权限，权限控制粒度可细化到表级
* SQL拦截
* 支持IP黑白名单设置
* **成本：**
* 支持使用X86 PC+廉价存储的部署模式

## 刚性要求

|  |  |
| --- | --- |
| **指标项** | **要求** |
| 系统可用度 | 大于99.99% |
| 平均故障发生间隔 | 大于90天 |
| 平均故障修复时间 | 要求由软件引发的故障一年内平均修复时间小于1小时 |
| 数据存储支持容量 | 无限制 |
| 集群性能 | 10节点内单机扩展系数 > 0.7 |
| TPS | >=给定值 |
| SQL语法支持 | 支持SQL92标准，覆盖99%以上应用开发常用SQL |
| 应用开发工作量 | 支持JDBC驱动,应用开发与集中数据库相比工作量增幅在20%以内 |

## 基础功能

|  |  |
| --- | --- |
| **指标项** | **指标描述** |
| 安装部署 | 整套分布式数据库软件的安装部署，包括：数据库中间件集群、数据库集群 |
| 建库建表（DDL 操作） | 创建逻辑库，提供对创建/修改/删除表、创建/删除索引等DDL语句的支持； |
| 数据分片 | 支持多种数据切分算法，提供水平扩展能力，允许用户自定义表的分片数量及指定分片节点 |
| 数据访问 | 数据访问对应用透明，支持JDBC驱动，支持业界或开源标准协议；兼容常用数据库客户端 |
| DML SQL 操作 | 提供对常用的SELECT, UPDATE, INSERT, DELETE, UNION, JOIN等 SQL 语法的支持； |
| 关键字/汇聚函数 | 提供多节点数据的关键字支持如：ORDER BY、GROUP BY、DESTINCT、分页等，提供多节点数据的函数支持，如：SUM、COUNT、MAX、MIN 等； |
| 单库事务支持 | 完整支持单库事务 |
| 全局序列 | 为应用提供全局唯一的序列号，保障数据库表主键的唯一性 |
| 分片索引 | 可实时构建分片索引数据，基于索引数据查询到非分片键对应的分片键数据，避免广播查询 |
| 全局表 | 将数据变化少的单表（全局表）同步到需要关联全局表查询的分库上，提升关联查询效率 |
| Hint语法/透传 | 通过特殊语法，允许应用指定SQL在哪个分片执行、获取表分片策略信息、控制读写分离等，方便开发和运维 |
| 库内分表 | 对分片表在单个分片内实现二次分表策略，支持大数据量表存储，库内分表须同样支持分表策略 |

## 可用性

|  |  |
| --- | --- |
| **指标项** | **指标描述** |
| 集群高可用 | 没有节点失效的问题：不会因为某个节点服务器的宕机，导致整个集群崩溃 |
| 异常恢复 | 运行环境异常（如：网络异常、主机异常）导致节点无法提供服务，在运行环境恢复正常后，节点能在5分钟内快速恢复提供服务 |
| 过载保护 | 可设置连接数限制，超过限制后拒绝连接并返回错误，后端数据库发生过载时部分请求直接在数据库中间件返回，不发向后端数据库执行。 |
| 集群分组 | 对集群内的节点进行集群分组，按分组级别进行流控，提供差异化服务，保障核心业务访问的稳定性 |

## 可靠性

|  |  |
| --- | --- |
| **指标项** | **指标描述** |
| 数据库主从节点数据一致性 | 数据库主从节点的数据在异常场景下仍能保持100%一致 |
| 运行稳定性 | 进行疲劳压试，要求达到系统最大事务处理数的80%，CPU占用率达到70%左右，系统能够平稳运行24小时以上 |
| 跨分片数据操作的数据一致性 | 提供的分布式数据库事务功能，实现最终一致性事务支持 |

## 监控/运维

|  |  |
| --- | --- |
| **指标项** | **指标描述** |
| 全局统一管理配置 | 配置集中存放，统一管理 |
| 实时监控 | 提供TPS、CPU、内存、慢SQL语句等的实时监控，提供一键故障检测功能 |
| 在线扩容 | 支持不停业务平滑扩容 |
| 组件灰度发布 | 支持在一个集群中发布不同版本的节点服务，可设置每个节点的负载系数 |
| 在线配置修改 | 支持在管理平台修改配置参数，大部分配置修改后在各个实例节点能实时生效 |
| 在线建表、建切片索引、建全局序列 | 支持在线创建数据库表、创建切片索引、创建全局序列，实时生效 |
| 管理命令 | 为方便用户使用和维护数据库中间件，数据库中间件提供配置查看、连接状态、统计监控、序列运维等管理命令 |
| 统计分析 | 提供SQL执行、事务、TPS、慢SQL等数据的统计分析，为应用提供合理的数据切分及分片索引建议 |
| 一键故障检测 | 提供一键检测功能，快速检测当前服务中各组件或模块的异常点，方便运维人员快速发现问题，包括：数据库节点运行状态、数据库中间件运行状态、统一配置服务、分片索引接口状态、全局序列接口状态、服务响应时延等 |
| 扩展DDL语法 | 扩展DDL语法，支持通过脚本方式指定分片算法及建表、建序列、修改表结构等 |
| 数据备份/恢复 | 容灾备份，提供自动化的备份工具，实现多种备份策略（全量备份、全量+增量备份、时间点备份）和自动恢复的能力 |

## 安全性

|  |  |
| --- | --- |
| **指标项** | **指标描述** |
| 权限验证 | 支持数据库中间件用户管理，可设置用户的访问权限，访问权限支持到表级，并且支持账号有效期和密码过期时间设置 |
| SQL拦截 | 支持配置数据库SQL拦截，对常见危险SQL操作可以通过SQL拦截功能进行截获处理不执行具体数据操作，保障分布式数据库性能及安全；要求开启SQL拦截后对整理性能影响要保持可控 < 给定影响值 |
| IP黑白名单 | 可设置IP地址的黑白名单，在黑名单中的IP地址无法访问数据库中间件 |

## 性能

|  |  |
| --- | --- |
| **指标项** | **指标描述** |
| insert语句 | 测试insert、select、update、delete语句的性能指标，包括响应时间、每秒处理事务数、CPU占比、可用内存数、磁盘使用率（%）、网络吞吐量和出错情况等，测试场景包括：单节点单库、单节点多库、多节点多库，单机平均性能大于9w TPS（4路8核/64GB内存）， 线性扩展系数>0.7 |
| select语句 |
| update语句 |
| delete语句 |
| 集成场景业务性能测试 | 采用集成场景订单受理做为综合性能标准测试， TPS > 给定值 |
| 全局序列获取 | 测试全局序列获取性能，单机平均性能大于20w TPS（4路8核/64GB内存）， 线性扩展系数>0.7 |

1. 测试准备

## 产品文档

*请列举测试之前需参测厂商提供的产品文档。*

至少包括：产品安装部署文档、操作指南文档

## 测试代码

*请列举测试之前需参测厂商提供的测试代码（独立程序、开发SDK、SHELL脚本，特定工具的脚本等）*

至少包括：数据库备份恢复（程序/脚本）、通用数据库客户端、java jdbc测试程序、全局序列测试程序

## 测试环境资源

### 部署示意图



图3.1-1产品部署示意图

上图为测试环境的部署示意图，不同厂家的产品部署方式可能会不一样，可以根据实际情况进行细化，但测试时将基于相同的硬件设备进行测试。按示意图所示，测试服务器将包括：

2台SLB服务器（云主机）、3台配置服务器（云主机）、3台数据库中间服务器（物理机）、5台数据库服务器（物理机，每台上安装2个数据库实例），具体的服务器配置见下节。

### 硬件配置

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **用处** | **类型** | **台数** | **CPU** | **内存** | **操作系统** | **备注** |
| 压力测试机 | 虚拟机 | 3 | 8 | 16G | CentOS7 |  |
| SLB服务器 | 虚拟机 | 2 | 8 | 16G | CentOS7 |  |
| 配置服务器 | 虚拟机 | 3 | 8 | 16G | CentOS7 |  |
| 数据库中间件 | 物理机 | 3 | 2路8核CPU | 64G | CentOS7 |  |
| 数据库 | 物理机 | 5 | 2路8核CPU | 128G | CentOS7 |  |

### 数据准备

#### 数据模型



#### 数据库脚本

 

本测试规范，包括数据库模型设计、建库脚本及测试用例的SQL语句均采用 MySQL协议和SQL92标准语法编写，若使用的产品采用其他协议或者语法有差异的（如：采用Oracle协议和语法），请自行调整模型和SQL语句。

#### 分片规则

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **序号** | **表名** | **分片类型** | **分片键** | **分片说明** | **备注** |
| 1 | PRODUCT  产品 | 全局表 |  | 数据在每个分片上有完整一份数据 |  |
| 2 | OFFER  销售品 | 全局表 |  | 数据在每个分片上有完整一份数据 |  |
| 3 | OFFER\_PROD\_REL  销售品产品关联 | 全局表 |  | 数据在每个分片上有完整一份数据 |  |
| 4 | PROD\_INST  产品实例 | 分片表 | OWNER\_CUST\_ID | 数据按owner\_cust\_id均匀分散到20个分片上 |  |
| 5 | OFFER\_INST  销售品实例 | 分片表 | OWNER\_CUST\_ID | 数据按owner\_cust\_id均匀分散到20个分片上 |  |
| 6 | OFFER\_PROD\_INST\_REL  销售品产品实例关联 | 分片表 | OWNER\_CUST\_ID | 数据按owner\_cust\_id均匀分散到20个分片上 |  |
| 7 | CUSTOMER  客户 | 分片表 | CUST\_ID | 数据按cust\_id均匀分散到20个分片上 |  |
| 8 | CUSTOMER\_ORDER  客户订单 | 分片表 | CUST\_ID | 数据按cust\_id均匀分散到20个分片上 |  |
| 9 | ORDER\_ITEM  订单项 | 分片表 | BELONG\_CUST\_ID | 数据按  belong\_cust\_id均匀分散到20个分片上 |  |
| 10 | ORD\_PROD\_INST  订单产品实例 | 分片表 | OWNER\_CUST\_ID | 数据按owner\_cust\_id均匀分散到20个分片上 |  |
| 11 | ORD\_OFFER\_INST  订单销售品实例 | 分片表 | OWNER\_CUST\_ID | 数据按owner\_cust\_id均匀分散到20个分片上 |  |
| 12 | ORD\_OFFER\_PROD\_INST\_REL  订单销售品产品实例关联 | 分片表 | OWNER\_CUST\_ID | 数据按owner\_cust\_id均匀分散到20个分片上 |  |
| 13 | TEST\_DDL\_PROD  测试DDL订单表 | 全局表 |  | 数据在每个分片上有完整一份数据 | 临时测试表 |
| 14 | TEST\_DDL\_PROD\_INST  测试DDL订单实例表 | 分片表 | CUST\_ID | 数据按cust\_id均匀分散到20个分片上 | 临时测试表 |

1. 测试用例

## 基础功能

### 安装部署

#### 安装数据库集群

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 数据库集群部署测试 | | |
| 权值 | 基础 | | |
| 测试目的与范围 | 验证是否能简单快速部署数据库集群 | | |
| 预置条件 | 本地为干净环境，或无冲突应用 | | |
| 测试过程 | 按照提供的文档/安装脚本/管理平台等搭建数据库集群 | | |
| 预期结果 | 1. 能够快速搭建集群服务； 2. 能正常提供服务；   评估点：安装自动化程度、部署耗时、部署难易程度 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

#### 安装数据库中间件集群

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 数据库中间件集群部署测试 | | |
| 权值 | 基础 | | |
| 测试目的与范围 | 验证是否能简单快速部署数据库中间件集群 | | |
| 预置条件 | 本地为干净环境，或无冲突应用 | | |
| 测试过程 | 按照提供的文档/安装脚本/管理平台/数据库中间件等搭建数据库中间件集群 | | |
| 预期结果 | 1. 正确操作，且中间件关联的数据库集群能正常提供服务时，能够快速搭建集群服务，并且能正常提供服务； 2. 部署中间件集群中的某个实例时，如果该实例的IP和端口与已存在的中间件实例相同，则不能搭建并会给出错误提示； 3. ；   评估点：安装自动化程度、部署耗时、部署难易程度 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 建库建表（DDL SQL）

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 建库及DDL SQL语句测试 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 测试对DDL SQL语句的支持情况 | | |
| 预置条件 | 启动中间件且能提供正常服务 | | |
| 测试过程 | 测试使用 3.3.3.2 数据库脚本：测试数据库脚本.sql   * + - 1. 创建逻辑库bss\_crm\_test，可选择字符集和存储引擎， 字符集指定为UTF8，分片数量设置为20；       2. 为数据库中间件添加用户test，密码为test,该用户拥有逻辑库bss\_crm\_test的所有权限；       3. 在逻辑库上执行“测试数据库脚本.sql”       4. 按照3.3.3.3规定的分片规则设置各个表的分片规则       5. 检查库表创建结果 | | |
| 预期结果 | * + - 1. 在中间件上创建逻辑库bss\_crm\_test时，可指定逻辑库的字符集、存储引擎及分片数量；       2. 支持通过界面和脚本两种不同的方式设置表的分片规则       3. 可通过界面查看逻辑库及逻辑表结构       4. 逻辑库和表的创建符合预期，可用作后续用例测试的基准库   其他评估点：操作是否便利、提示是否友好、执行结果是否直观 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 表结构修改（DDL SQL）

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 表结构修改测试 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 可提供统一的表结构修改入口，可通过DDL语句修改表结构， | | |
| 预置条件 | 数据库集群、数据库中间件集群已搭建完成并正常运行 | | |
| 测试过程 | 测试使用 3.3.3.2 数据库脚本：测试数据库脚本-2.sql   1. 创建逻辑库test\_ddl 2. 在逻辑库上执行“测试数据库脚本-2.sql” 3. 为test\_ddl\_prod表添加索引   CREATE INDEX ix\_product\_nbr ON test\_ddl\_prod (product\_nbr);   1. 为test\_ddl\_prod、test\_ddl\_prod\_inst设置分片规则，test\_ddl\_prod为全局表、test\_ddl\_prod\_inst为分片表（分片键为cust\_id），分片数量20，检查20个分片上的表结构是否正确，test\_ddl\_prod是否存在索引ix\_product\_nbr 2. 执行修改表结构语句   ALTER TABLE test\_ddl\_prod ADD `test` INT DEFAULT 0;  ALTER TABLE test\_ddl\_prod\_inst ADD `test` INT DEFAULT 0;  ALTER TABLE test\_ddl\_prod DROP `test`;  ALTER TABLE test\_ddl\_prod\_inst DROP `test`;  ALTER TABLE test\_ddl\_prod MODIFY status\_cd VARCHAR(10);  ALTER TABLE test\_ddl\_prod\_inst MODIFY prod\_inst\_nbr VARCHAR(32);  ALTER TABLE test\_ddl\_prod CHANGE status\_cd status\_cd\_test VARCHAR(6);  ALTER TABLE test\_ddl\_prod\_inst CHANGE prod\_inst\_nbr prod\_inst\_nbr\_test VARCHAR(30); 在物理数据库分片上查看表结构修改是否符合预期   1. 执行create index(加索引)   CREATE INDEX ix\_product\_id ON test\_ddl\_prod\_inst (product\_id); 查看物理数据库是否相应增加了索引   1. 执行drop index(除索引)   DROP INDEX ix\_product\_nbr ON test\_ddl\_prod;  DROP INDEX ix\_product\_id ON test\_ddl\_prod\_inst; 查看物理数据库是否相应删除了索引   1. 执行drop table(删除表)   DROP TABLE test\_ddl\_prod;  DROP TABLE test\_ddl\_prod\_inst; 查看物理数据库是否已经删除了这两个表 | | |
| 预期结果 | 1. DDL语句执行后，表结构修改符合预期 2. DDL语句执行有详细的日志记录 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 数据分片

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 数据分片 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 测试分片算法的完备性，是否支持字符串，数字，日期等多种拆分纬度； | | |
| 预置条件 | 1. 用例4.1.3执行成功 2. 逻辑库test\_ddl中不存在任何表 3. 分片数量为20 | | |
| 测试过程 | 1. 创建test\_ddl\_prod\_inst表 2. 以cust\_id为分片键，选择一种分片算法（分片键的类型支持字符串，数字，日期型）   插入数据，检查数据是否正确存储到分片上   1. 以acc\_nbr为分片键，选择一种分片算法 2. 插入数据，检查数据是否正确存储到分片上 3. 以create\_date为分片键，选择一种分片算法 4. 插入数据，检查数据是否正确存储到分片上 5. 以area\_id + cust\_id为分片键，选择一种分片算法 6. 插入数据，检查数据是否正确存储到分片上   以cust\_id为分片键按整数取模构造测试数据如下：（其他分片键和分片算法自行构造测试数据）  insert into test\_ddl\_prod\_inst(cust\_id, product\_id, prod\_inst\_id,prod\_inst\_nbr,acc\_nbr,create\_date, area\_id)  values(1, 1, 111, '1111', '1111',now(), 1);  insert into test\_ddl\_prod\_inst(cust\_id, product\_id, prod\_inst\_id,prod\_inst\_nbr,acc\_nbr,create\_date, area\_id)  values(2, 1, 222, '1111', '1111',now(), 1);  insert into test\_ddl\_prod\_inst(cust\_id, product\_id, prod\_inst\_id,prod\_inst\_nbr,acc\_nbr,create\_date, area\_id)  values(3, 1, 333, '1111', '1111',now(), 1);  insert into test\_ddl\_prod\_inst(cust\_id, product\_id, prod\_inst\_id,prod\_inst\_nbr,acc\_nbr,create\_date, area\_id)  values(4, 1, 444, '1111', '1111',now(), 1);  insert into test\_ddl\_prod\_inst(cust\_id, product\_id, prod\_inst\_id,prod\_inst\_nbr,acc\_nbr,create\_date, area\_id)  values(5, 1, 555, '1111', '1111',now(), 1);  insert into test\_ddl\_prod\_inst(cust\_id, product\_id, prod\_inst\_id,prod\_inst\_nbr,acc\_nbr,create\_date, area\_id)  values(6, 1, 666, '1111', '1111',now(), 1);  insert into test\_ddl\_prod\_inst(cust\_id, product\_id, prod\_inst\_id,prod\_inst\_nbr,acc\_nbr,create\_date, area\_id)  values(7, 1, 777, '1111', '1111',now(), 1);  insert into test\_ddl\_prod\_inst(cust\_id, product\_id, prod\_inst\_id,prod\_inst\_nbr,acc\_nbr,create\_date, area\_id)  values(8, 1, 888, '1111', '1111',now(), 1);  insert into test\_ddl\_prod\_inst(cust\_id, product\_id, prod\_inst\_id,prod\_inst\_nbr,acc\_nbr,create\_date, area\_id)  values(9, 1, 999, '1111', '1111',now(), 1);  insert into test\_ddl\_prod\_inst(cust\_id, product\_id, prod\_inst\_id,prod\_inst\_nbr,acc\_nbr,create\_date, area\_id)  values(10, 1, 1010, '1111', '1111',now(), 1);  insert into test\_ddl\_prod\_inst(cust\_id, product\_id, prod\_inst\_id,prod\_inst\_nbr,acc\_nbr,create\_date, area\_id)  values(11, 1, 1111, '1111', '1111',now(), 1);  insert into test\_ddl\_prod\_inst(cust\_id, product\_id, prod\_inst\_id,prod\_inst\_nbr,acc\_nbr,create\_date, area\_id)  values(12, 1, 1212, '1111', '1111',now(), 1);  insert into test\_ddl\_prod\_inst(cust\_id, product\_id, prod\_inst\_id,prod\_inst\_nbr,acc\_nbr,create\_date, area\_id)  values(13, 1, 1313, '1111', '1111',now(), 1);  insert into test\_ddl\_prod\_inst(cust\_id, product\_id, prod\_inst\_id,prod\_inst\_nbr,acc\_nbr,create\_date, area\_id)  values(14, 1, 1414, '1111', '1111',now(), 1);  insert into test\_ddl\_prod\_inst(cust\_id, product\_id, prod\_inst\_id,prod\_inst\_nbr,acc\_nbr,create\_date, area\_id)  values(15, 1, 1515, '1111', '1111',now(), 1);  insert into test\_ddl\_prod\_inst(cust\_id, product\_id, prod\_inst\_id,prod\_inst\_nbr,acc\_nbr,create\_date, area\_id)  values(16, 1, 1616, '1111', '1111',now(), 1);  insert into test\_ddl\_prod\_inst(cust\_id, product\_id, prod\_inst\_id,prod\_inst\_nbr,acc\_nbr,create\_date, area\_id)  values(17, 1, 1717, '1111', '1111',now(), 1);  insert into test\_ddl\_prod\_inst(cust\_id, product\_id, prod\_inst\_id,prod\_inst\_nbr,acc\_nbr,create\_date, area\_id)  values(18, 1, 1818, '1111', '1111',now(), 1);  insert into test\_ddl\_prod\_inst(cust\_id, product\_id, prod\_inst\_id,prod\_inst\_nbr,acc\_nbr,create\_date, area\_id)  values(19, 1, 1919, '1111', '1111',now(), 1);  insert into test\_ddl\_prod\_inst(cust\_id, product\_id, prod\_inst\_id,prod\_inst\_nbr,acc\_nbr,create\_date, area\_id)  values(20, 1, 2020, '1111', '1111',now(), 1);  20条数据按照cust\_id取模结果分别存储在20个分片上 | | |
| 预期结果 | * + - 1. 创建分片算法成功；       2. 成功访问数据库中间件，并且执行语句成功；       3. 数据能按分片规则存储到正确的分片上       4. 分片键的类型支持字符串，数字，日期型       5. 至少支持一种两个或两个以上字段作为分片键的算法，如：区域+客户标识，先通过区域进行分片，相同的区域内再通过cust\_id进行分片 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 数据访问

#### 使用JDBC测试链接分布式数据库服务

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 数据访问 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 测试是否支持使用jdbc连接访问 | | |
| 预置条件 | 1. 启动中间件服务； 2. 用例4.1.2 中的库表已存在 3. 测试中使用到的表都为空表； | | |
| 测试过程 | 编写程序使用jdbc连接到中间件，并在程序中执行一些select,insert语句  **测试分片表：**  编写程序测试以下SQL语句：  INSERT INTO customer(cust\_id, credit\_id, service\_grade\_id, credit\_limit\_id, party\_id, cust\_name, common\_region\_id, cust\_source\_id, cust\_order\_id) VALUES(200124353, 1112, 3, 10010, 10000, '测试用户', 200, 101, 10011);  INSERT INTO customer(cust\_id, credit\_id, service\_grade\_id, credit\_limit\_id, party\_id, cust\_name, common\_region\_id, cust\_source\_id, cust\_order\_id) VALUES(200124354, 1112, 3, 10010, 10000, '测试用户', 200, 101, 10011);  INSERT INTO customer(cust\_id, credit\_id, service\_grade\_id, credit\_limit\_id, party\_id, cust\_name, common\_region\_id, cust\_source\_id, cust\_order\_id) VALUES(200124355, 1112, 3, 10010, 10000, '测试用户', 200, 101, 10011);  SELECT \* FROM customer;  SELECT \* FROM customer WHERE cust\_id = 200124355;  DELETE FROM customer;  **测试全局表：**  编写程序测试以下SQL语句：  INSERT INTO product(prod\_id, prod\_nbr, prod\_name, base\_offer\_id) VALUES(1001010, 9934, '固定电话', 1200);  INSERT INTO product(prod\_id, prod\_nbr, prod\_name, base\_offer\_id) VALUES(1001011, 9935, '移动电话', 1200);  INSERT INTO product(prod\_id, prod\_nbr, prod\_name, base\_offer\_id) VALUES(1001012, 9936, 'PSTN 电话', 1200);  SELECT \* FROM product;  DELETE FROM product; | | |
| 预期结果 | 能正常访问组件，并可以返回正确的结果 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

#### 使用客户端测试链接分布式数据库服务

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 数据访问 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 测试是否支持使用Workbench , Navicat, SQLyog等主流客户端的连接访问 | | |
| 预置条件 | * + - 1. 启动组件服务；       2. 用例4.1.2中的库表已存在       3. 测试中使用的表都为空表； | | |
| 测试过程 | 1. 分别使用Workbench , Navicat, SQLyog客户端连接组件，执行一些select, insert 语句   **测试分片表：**  使用客户端测试以下SQL语句：  INSERT INTO customer(cust\_id, credit\_id, service\_grade\_id, credit\_limit\_id, party\_id, cust\_name, common\_region\_id, cust\_source\_id, cust\_order\_id) VALUES(200124353, 1112, 3, 10010, 10000, '测试用户a', 200, 101, 10011);  INSERT INTO customer(cust\_id, credit\_id, service\_grade\_id, credit\_limit\_id, party\_id, cust\_name, common\_region\_id, cust\_source\_id, cust\_order\_id) VALUES(200124354, 1112, 3, 10010, 10000, '测试用户b', 200, 101, 10011);  INSERT INTO customer(cust\_id, credit\_id, service\_grade\_id, credit\_limit\_id, party\_id, cust\_name, common\_region\_id, cust\_source\_id, cust\_order\_id) VALUES(200124355, 1112, 3, 10010, 10000, '测试用户c', 200, 101, 10011);  SELECT \* FROM customer;  SELECT \* FROM customer WHERE cust\_id = 200124355;  DELETE FROM customer;  **测试全局表：**  使用客户端测试以下SQL语句：  INSERT INTO product(prod\_id, prod\_nbr, prod\_name, base\_offer\_id) VALUES(1001010, 9934, '固定电话', 1200);  INSERT INTO product(prod\_id, prod\_nbr, prod\_name, base\_offer\_id) VALUES(1001011, 9935, '移动电话', 1200);  INSERT INTO product(prod\_id, prod\_nbr, prod\_name, base\_offer\_id) VALUES(1001012, 9936, 'PSTN 电话', 1200);  SELECT \* FROM product;  DELETE FROM product; | | |
| 预期结果 | 能正常访问组件，并可以返回正确的结果 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成（*支持一种主流*）， D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

### DML SQL

#### INSERT语句测试

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | INSERT支持语句测试 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 验证单表单分库插入语句正确性 | | |
| 预置条件 | 用例4.1.2中的库表已存在  CUSTOMER为空表 | | |
| 测试过程 | 执行SQL语句:  INSERT INTO customer(cust\_id, credit\_id, service\_grade\_id, credit\_limit\_id, party\_id, cust\_name, common\_region\_id, cust\_source\_id, cust\_order\_id) VALUES(200124353, 1112, 3, 10010, 10000, '测试用户', 200, 101, 10011);  INSERT INTO customer(cust\_id, credit\_id, service\_grade\_id, credit\_limit\_id, party\_id, cust\_name, common\_region\_id, cust\_source\_id, cust\_order\_id) VALUES(200124354, 1112, 3, 10010, 10000, '测试用户1', 200, 101, 10011);  INSERT INTO customer(cust\_id, credit\_id, service\_grade\_id, credit\_limit\_id, party\_id, cust\_name, common\_region\_id, cust\_source\_id, cust\_order\_id) VALUES(200124355, 1112, 3, 10010, 10000, '测试用户2', 200, 101, 10011); | | |
| 预期结果 | 测试程序运行正常  数据能正确写入数据库 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

#### SELECT语句测试

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | SELECT语句支持测试 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 验证单表单分库查询语句正确性 | | |
| 预置条件 | * + - 1. 用例4.1.2中的库表已存在       2. 测试中使用到的表都为空表 | | |
| 测试过程 | 1. 在customer表中插入1000条记录  2. 执行SQL语句(根据具体的插入数据调整查询条件):  SELECT \* FROM customer;  SELECT \* FROM customer WHERE cust\_id = 200124355; | | |
| 预期结果 | 1． 测试程序运行正常。  2． 语句1正常返回该表的全部记录，语句2返回所有匹配数据的记录。 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

#### UPDATE语句测试

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 测试UPDATE功能语句的执行 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 验证update语句在单表多分片情况的下执行正确性 | | |
| 预置条件 | 用例4.1.2的库表已存在 | | |
| 测试过程 | 1. 已建好CUSTOMER表；  2. 准备100条初始化数据，按分片规则均匀平均切分数据；  2. 执行update操作(根据具体的初始化数据调整更新条件)  UPDATE customer SET cust\_name='修改用户名a' WHERE cust\_id=200124353;  UPDATE customer SET cust\_name='修改用户名1' WHERE cust\_id=200124354;  UPDATE customer SET cust\_name='修改用户名2' WHERE cust\_id=200124355;  3. 查询更新后的结果与预期的结果是否一致，一致则测试通过，否则失败 | | |
| 预期结果 | 1． 测试程序运行正常。  2． 数据更新成功 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

#### DELETE语句测试

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | DELETE支持测试用例 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 验证单表单分库删除语句正确性 | | |
| 预置条件 | 用例4.1.2的库表已存在 | | |
| 测试过程 | 执行SQL语句:  DELETE FROM customer;  INSERT INTO customer(cust\_id, credit\_id, service\_grade\_id, credit\_limit\_id, party\_id, cust\_name, common\_region\_id, cust\_source\_id, cust\_order\_id) VALUES(200124353, 1112, 3, 10010, 10000, '测试用户1', 200, 101, 10011);  INSERT INTO customer(cust\_id, credit\_id, service\_grade\_id, credit\_limit\_id, party\_id, cust\_name, common\_region\_id, cust\_source\_id, cust\_order\_id) VALUES(200124354, 1112, 3, 10010, 10000, '测试用户2', 200, 101, 10011);  INSERT INTO customer(cust\_id, credit\_id, service\_grade\_id, credit\_limit\_id, party\_id, cust\_name, common\_region\_id, cust\_source\_id, cust\_order\_id) VALUES(200124355, 1112, 3, 10010, 10000, '测试用户3', 200, 101, 10011);  SELECT \* FROM customer;  DELETE FROM customer WHERE cust\_id=200124353;  SELECT \* FROM customer;  DELETE FROM customer WHERE cust\_id in (200124354);  SELECT \* FROM customer;  DELETE FROM customer WHERE cust\_name = '测试用户3';  SELECT \* FROM customer; | | |
| 预期结果 | 程序执行INSERT后，可以查询到3条数据；  执行第一条DELETE后，能查询到两条数据，并且没有cust\_id=200124353的数据；  执行第二条DELETE后，能查询到一条数据，并且没有cust\_id=200124354的数据；  执行第三条DELETE后，查询结果为空 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

#### SELECT条件查询

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | SELECT语句支持测试 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 验证单表单分库条件查询语句正确性 | | |
| 预置条件 | * + - 1. 用例4.1.2中的库表已存在       2. 分片表customer中存在多条数据 | | |
| 测试过程 | 执行SQL语句(根据具体的数据调整查询条件):  SELECT \* FROM customer WHERE cust\_id=200124353;  SELECT \* FROM customer WHERE cust\_name='测试用户a';  SELECT \* FROM customer WHERE cust\_name like '%测试用户%';  SELECT \* FROM customer WHERE cust\_name='测试用户a' or cust\_name='测试用户3';  SELECT \* FROM customer WHERE cust\_id <200124360;  SELECT \* FROM customer WHERE cust\_id between 200124350 and 200124360;  SELECT \* FROM customer WHERE cust\_id in (200124353, 200124354, 200124355); | | |
| 预期结果 | 1． 测试程序运行正常。  2． 正确返回该表的对应记录。 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

#### SELECT子查询与join操作

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | SELECT支持测试用例 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 验证单分库条件查询语句正确性 | | |
| 预置条件 | 用例4.1.2中的库表已存在  customer、product、prod\_inst表为空表 | | |
| 测试过程 | 执行SQL语句，初始化数据:  DELETE FROM prod\_inst;  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334132, 1001010, 200124353, 10001, 9909, 200124353);  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334133, 1001010, 200124354, 10001, 9909, 200124354);  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334134, 1001010, 200124353, 10001, 9909, 200124355);  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334144, 1001011, 200124355, 10001, 9909, 200124355);  DELETE FROM customer;  INSERT INTO customer(cust\_id, credit\_id, service\_grade\_id, credit\_limit\_id, party\_id, cust\_name, common\_region\_id, cust\_source\_id, cust\_order\_id) VALUES(200124353, 1112, 3, 10010, 10000, '测试用户a', 200, 101, 10011);  INSERT INTO customer(cust\_id, credit\_id, service\_grade\_id, credit\_limit\_id, party\_id, cust\_name, common\_region\_id, cust\_source\_id, cust\_order\_id) VALUES(200124354, 1112, 3, 10010, 10000, '测试用户b', 200, 101, 10011);  INSERT INTO customer(cust\_id, credit\_id, service\_grade\_id, credit\_limit\_id, party\_id, cust\_name, common\_region\_id, cust\_source\_id, cust\_order\_id) VALUES(200124355, 1112, 3, 10010, 10000, '测试用户c', 200, 101, 10011);  DELETE FROM product;  INSERT INTO product(prod\_id, prod\_nbr, prod\_name, base\_offer\_id) VALUES(1001010, 9934, '固定电话', 1200);  INSERT INTO product(prod\_id, prod\_nbr, prod\_name, base\_offer\_id) VALUES(1001011, 9935, '移动电话', 1200);  INSERT INTO product(prod\_id, prod\_nbr, prod\_name, base\_offer\_id) VALUES(1001012, 9936, 'PSTN 电话', 1200);  执行测试：  select \* from prod\_inst p, customer c where p.use\_cust\_id = c.cust\_id;  select \* from prod\_inst p left join product c on p.prod\_id = c.prod\_id;  select \* from prod\_inst union select \* from prod\_inst;  select \* from prod\_inst p where p.use\_cust\_id in(select c.cust\_id from customer c); | | |
| 预期结果 | 1． 测试程序运行正常。  2． 正确返回该表的对应记录。 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

#### UPDATE分片键报错

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 对于 update 分片键的 SQL，数据库中间件会给报错 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 程序或人为误操作试图更新分片键的键值，验证更新分片键时，数据库中间件会报错 | | |
| 预置条件 | * + - 1. 用例4.1.2中的库表已存在       2. 存在分片表prod\_inst | | |
| 测试过程 | 执行SQL语句，初始化数据:  DELETE FROM prod\_inst;  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334132, 1001010, 200124353, 10001, 9909, 200124353);  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334133, 1001010, 200124354, 10001, 9909, 200124354);  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334134, 1001010, 200124355, 10001, 9909, 200124355);  执行更新分片键操作，程序报错  UPDATE prod\_inst SET owner\_cust\_id=200124354 WHERE owner\_cust\_id=200124353; | | |
| 预期结果 | 1. 测试程序运行返回提示不允许修改分片键错误信息； 2. owner\_cust\_id=200124353 记录没有发生变化； | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

#### select ... for update

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 测试select for update 语句 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 1. 验证默认情况下select for update 是自动提交的 2. 验证在事务中select for update能够锁住相应的记录不被修改 | | |
| 预置条件 | * + - 1. 用例4.1.2中的库表已存在       2. 在prod\_inst中插入5条记录，插入语句可参考用例4.1.7.1 | | |
| 测试过程 | 新建两个数据库连接db1及db2   1. 用db1执行select prod\_inst\_id, account from prod\_inst where owner\_cust\_id = 200124353 for update; 2. 用db2执行update prod\_inst set account = 'xxxx' where owner\_cust\_id = 200124353;能够执行成功 3. db1显示开启事务 4. 用db1执行select prod\_inst\_id, account from prod\_inst where owner\_cust\_id = 200124353 for update; 5. 用db2 执行update prod\_inst set account = 'xxxx' where owner\_cust\_id = 200124353;执行报错   抛出SQLException异常，异常原因类似：Lock wait timeout exceeded; try restarting transaction | | |
| 预期结果 | 1. 测试程序运行正常。 2. 正确返回该表的对应记录。 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

#### 操作的表不存在时能准确报错

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 对一个不存在的数据库表进行增删改查操作，数据库中间件能准确报错 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 验证操作不存在的数据库表时，统一访问层能准确报错 | | |
| 预置条件 | * + - 1. 用例4.1.2中的库表已存在       2. 逻辑库bss\_crm\_test中不存在表prod\_inst\_1 | | |
| 测试过程 | insert into prod\_inst\_1(prod\_inst\_id,use\_cust\_id) values(999, 200124353);  select prod\_inst\_id,use\_cust\_id from prod\_inst\_1;  update prod\_inst\_1 set account = 200124353 where prod\_inst\_id = 999;  delete from prod\_inst\_1 where prod\_inst\_id = 999; | | |
| 预期结果 | 1. 测试程序运行正常 2. 能准确报错。例如：can't find table define in schema prod\_inst\_1 schema:it\_test | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 关键字/汇聚函数

#### MAX() & MIN()

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | MAX()和MIN()函数支持测试 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 通过数据库中间件，可以支持带MAX or MIN的SQL语句 | | |
| 预置条件 | 用例4.1.2中的库表已存在  表prod\_inst为空表 | | |
| 测试过程 | 向表中插入下面的数据:  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334132, 1001010, 200124353, 10001, 9909, 200124353);  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334133, 1001010, 200124354, 10001, 9909, 200124354);  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334134, 1001010, 200124355, 10001, 9909, 200124355);  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334134, 1001010, 200124356, 10001, 9909, 200124355);  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334134, 1001010, 200124357, 10001, 9909, 200124355);  执行下面的SQL语句：  select max(prod\_inst\_id) as maxId from prod\_inst;  select min(prod\_inst\_id) as minId from prod\_inst; | | |
| 预期结果 | 测试程序运行正常;  正常返回正确的结果； | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

#### SUM() & COUNT() & AVG()

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | SUM()、COUNT()、AVG()函数支持测试 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 通过数据库中间件，可以支持带SUM、COUNT、AVG等函数的SQL语句 | | |
| 预置条件 | 用例4.1.7.1中的插入sql语句已成功执行后，prod\_inst表中存在5条数据 | | |
| 测试过程 | 执行SQL语句:  select sum(prod\_inst\_id) as sumId from prod\_inst;  select count(prod\_inst\_id) as cntId from prod\_inst;  select avg(prod\_inst\_id) as avgId from prod\_inst; | | |
| 预期结果 | 测试程序运行正常；  正常返回正确的结果； | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

#### GROUP BY

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | GROUP BY关键字支持测试 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 通过数据库中间件，可以支持带GROUP BY的SQL语句 | | |
| 预置条件 | 用例4.1.7.1中的插入sql语句已成功执行后，prod\_inst表中存在5条数据 | | |
| 测试过程 | 执行SQL语句:  SELECT COUNT(1) AS cnt,use\_cust\_id FROM prod\_inst GROUP BY use\_cust\_id ORDER BY cnt; | | |
| 预期结果 | 测试程序运行正常；  正常返回合并与聚合后的结果； | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

#### GROUP BY ... HAVING ...

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | GROUP BY HAVING关键字支持测试 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 通过数据库中间件，可以支持带GROUP BY的SQL语句 | | |
| 预置条件 | 1. 用例4.1.7.1中的插入sql语句已成功执行后，prod\_inst表中存在5条数据 | | |
| 测试过程 | 执行SQL语句:  SELECT COUNT(1) AS cnt,use\_cust\_id FROM prod\_inst GROUP BY use\_cust\_id having cnt > 2; | | |
| 预期结果 | 测试程序运行正常；  正常返回合并与聚合后的结果； | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

#### ORDER BY

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | ORDER BY关键字支持测试 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 通过数据库中间件，可以支持带ORDER BY的SQL语句 | | |
| 预置条件 | 1. 用例4.1.7.1中的插入sql语句已成功执行后，prod\_inst表中存在5条数据 | | |
| 测试过程 | 执行带Order by的sql语句。  select prod\_inst\_id, account, owner\_cust\_id from prod\_inst order by owner\_cust\_id limit 0,4;  select prod\_inst\_id, account, owner\_cust\_id from prod\_inst order by owner\_cust\_id desc; | | |
| 预期结果 | 测试程序运行正常。  正常返回合并与排序后的结果。 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

#### DISTINCT

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | DISTINCT关键字支持测试 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 验证通过数据库中间件，可以支持带DISTINCT的SQL语句 | | |
| 预置条件 | 1. 用例4.1.7.1中的插入sql语句已成功执行后，prod\_inst表中存在5条数据 | | |
| 测试过程 | 执行SQL语句:  select distinct use\_cust\_id from prod\_inst; | | |
| 预期结果 | 测试程序运行正常。  返回的结果不重复，同在单数据库上执行结果相同。 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

#### 分页（LIMIT/ROWNUM）

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 分页关键字支持测试 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 通过数据库中间件，可以支持带分页关键字的sql语句 | | |
| 预置条件 | 用例4.1.7.1中的插入sql语句已成功执行后，prod\_inst表中存在5条数据 | | |
| 测试过程 | 查询第二页数据：  SELECT prod\_inst\_id, account, use\_cust\_id FROM prod\_inst ORDER BY prod\_inst\_id LIMIT 4,1;  **注：**如果是 Oracle 请改写上述语句的分页查询写法 | | |
| 预期结果 | 测试程序运行正常。  查询到的分页数据正确且唯一。 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 单库事务支持

#### 单分片不开启事务测试

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 单库事务支持测试 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 测试组件对事务的支持情况 | | |
| 预置条件 | 1. 用例4.1.2中的库表已存在 2. 分片表prod\_inst为空表 | | |
| 测试过程 | 设置事务autocommit=1，在单库上执行一些DML语句后commit,查看结果是否符合预期；  set autocommit=1;  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334133, 1001010, 200124354, 10001, 9909, 200124354);  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334133, 1001010, 200124354, 10001, 9909, 200124354); | | |
| 预期结果 | 重复插入数据，执行结果为第一条执行成功，第二条执行失败；  数据库成功插入一条记录； | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

#### 单分片开启事务测试

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 单库事务支持测试 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 测试组件对事务的支持情况 | | |
| 预置条件 | 1. 用例4.1.2中的库表已存在 2. 分片表prod\_inst为空 | | |
| 测试过程 | 设置事务autocommit=0，在单库上执行一些DML语句后commit,查看结果是否符合预期；   1. 执行下面的语句：   set autocommit=0;  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334133, 1001010, 200124354, 10001, 9909, 200124354);  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334135, 1001010, 200124354, 10001, 9909, 200124353);  Select \* from prod\_inst;prod\_inst表中有两条记录；   1. 清除表prod\_inst中的数据：   set autocommit=1;  Delete from prod\_inst;   1. 执行下面的语句：   set autocommit=0;  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334133, 1001010, 200124354, 10001, 9909, 200124354);  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334135, 1001010, 200124354, 10001, 9909, 200124353);  rollback;  select \* from prod\_inst;表中没有数据 | | |
| 预期结果 | 第一个事务执行成功，并且数据库有两条数据；  第二个事务执行失败，数据库没有数据； | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

#### 跨分片事务操作开关测试

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 在未开启分布式事务情况下执行跨分片开启事务失败测试 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 测试组件对事务的支持情况 | | |
| 预置条件 | 1. 用例4.1.2中的库表已存在 2. 分片表customer、prod\_inst为空表 | | |
| 测试过程 | 设置事务autocommit=0，执行一些DML语句后commit,查看结果是否符合预期；  set autocommit=0;  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334144, 1001011, 200124355, 10001, 9909, 200124355);  INSERT INTO customer(cust\_id, credit\_id, service\_grade\_id, credit\_limit\_id, party\_id, cust\_name, common\_region\_id, cust\_source\_id, cust\_order\_id) VALUES(200124353, 1112, 3, 10010, 10000, '测试用户a', 200, 101, 10011);  commit;  注：prod\_inst[owner\_cust\_id=200124355]和customer[cust\_id=200124353]两条数据不在同一个分片内 | | |
| 预期结果 | 在未开启分布式事务的情况下，执行第二条插入语句时报错提示“未开启分布式事务但存在跨分片操作”。  数据库中无数据； | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 全局序列

#### SQL语句获取序列测试

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 全局序列 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 测试组件的全局序列功能是否正常 | | |
| 预置条件 | 1. 用例4.1.2中的库表已存在 | | |
| 测试过程 | 1. 创建全局序列 2. 使用分布式数据库提供的获取序列语句语法获取序列值 | | |
| 预期结果 | 能够获取到递增有序无重复的序列  支持通过SQL语句方式获取序列值 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

#### 递增序列不能出现重复值测试

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 全局序列 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 测试组件的全局序列功能是否正常 | | |
| 预置条件 | 1. 用例4.1.2中的库表已存在 | | |
| 测试过程 | 1. 创建全局序列 2. 开启多个线程，不断地获取序列的值   记录所取到的序列值，查看是否有重复的序列 | | |
| 预期结果 | 无重复序列 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

#### 循环序列测试

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 全局序列 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 测试组件的全局序列功能是否正常 | | |
| 预置条件 | * + - 1. 用例4.1.2中的库表已存在 | | |
| 测试过程 | 1. 配置全局循环序列的最大最小值 2. 开启单个线程，不断地获取序列的值   观察获取到的序列是否能够在取完最大之后，重新从最小值开始获取 | | |
| 预期结果 | 观察到获取到的序列能够在取完最大之后，重新从最小值开始获取，并且获取序列时有序递增 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 分片索引

#### 创建新分片索引字段测试

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 分片索引功能测试 | | |
| 权值 | 基础 | | |
| 测试目的与范围 | 测试组件的新建分片索引字段功能 | | |
| 预置条件 | 1. 用例4.1.2中的库表已存在 2. 分片表prod\_inst为空表 | | |
| 测试过程 | 1. 在分片表prod\_inst上设置prod\_inst\_id->owner\_cust\_id(分片键)的分片索引。其中prod\_inst\_id为索引键，owner\_cust\_id为被索引键 2. 查看分片索引的创建情况 | | |
| 预期结果 | 能正常新建切片索引 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

#### 增加分片索引数据测试

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 分片索引功能测试 | | |
| 权值 | 基础 | | |
| 测试目的与范围 | 测试组件的新建分片索引字段功能 | | |
| 预置条件 | 1. 用例4.1.10.1已成功执行 | | |
| 测试过程 | 1. 向分片表prod\_inst中插入如下数据：   INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334132, 1001010, 200124353, 10001, 9909, 200124353);  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334133, 1001010, 200124354, 10001, 9909, 200124354);  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334134, 1001010, 200124355, 10001, 9909, 200124355);   1. 检查是否有相应的分片索引数据生成 | | |
| 预期结果 | 上述操作均能正常执行；  查看分片索引数据能查看到新增的分片索引数据； | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

#### 使用分片索引数据测试

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 分片索引功能测试 | | |
| 权值 | 基础 | | |
| 测试目的与范围 | 测试组件的使用分片索引的功能 | | |
| 预置条件 | 1. 用例4.1.10.2已成功执行 | | |
| 测试过程 | 1. 根据索引键进行查询，查看该语句是否只路由到特定的分片上，不会进行广播查询，执行下面的语句：   select \* from prod\_inst where prod\_inst\_id=2002016122334133; | | |
| 预期结果 | 1. 只路由到特定分片上执行，不会广播到所有分片执行  2. 能提供工具或命令确定执行语句的分片 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

#### 广播查询测试

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 分片索引功能测试 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 测试组件的新建分片索引字段功能 | | |
| 预置条件 | * + - 1. 用例4.1.10.3已成功执行 | | |
| 测试过程 | 1. 使用不带有索引键的 SQL进行查询，查看该语句是否进行广播查询,执行下面的语句：   select \* from prod\_inst where prod\_id=1001010; | | |
| 预期结果 | 上述执行均能正常完成；  语句路由到所有的分片上执行；  能提供工具或命令确定执行语句的分片 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 全局表

#### 全局表的简单 CRUD操作测试

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 全局表CRUD操作测试 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 测试全局表的CRUD语句支持情况 | | |
| 预置条件 | * + - 1. 用例4.1.2中的库表已存在       2. 全局表product为空表 | | |
| 测试过程 | 向全局表 product 中插入数据，执行下面的SQL语句：  INSERT INTO product(prod\_id, prod\_nbr, prod\_name, base\_offer\_id) VALUES(1001010, 9934, '固定电话', 1200);  INSERT INTO product(prod\_id, prod\_nbr, prod\_name, base\_offer\_id) VALUES(1001011, 9935, '移动电话', 1200);  INSERT INTO product(prod\_id, prod\_nbr, prod\_name, base\_offer\_id) VALUES(1001012, 9936, 'PSTN 电话', 1200)  在全局表上执行INSERT、UPDATE、DELETE语句变更表数据：  update product set prod\_name='yidongdianhua' where prod\_id=1001011;  delete from product where prod\_id=1001012;  在全局表上执行 SQL 语句执行 SELECT 语句：  select \* from product; | | |
| 预期结果 | 上述 SQL 语句都能执行成功；  检查INSERT、UPDATE、DELETE语句执行情况，此类语句在product的所有分片上都能执行成功，且各分片节点上数据一致；  检查SELECT语句返回结果，能正确返回某个节点的结果，而不是返回重复的内容； | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

#### 全局表与分片表的JOIN操作测试

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 全局表与分片表的JOIN操作测试 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 测试全局表与分片表的 JOIN语句支持情况 | | |
| 预置条件 | 用例4.1.2中的库表已存在  全局表product和分片表prod\_inst无数据 | | |
| 测试过程 | * + - 1. 向分片表prod\_inst中插入数据：   INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334132, 1001010, 200124353, 10001, 9909, 200124353);  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334133, 1001010, 200124354, 10001, 9909, 200124354);  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334134, 1001010, 200124355, 10001, 9909, 200124355);   * + - 1. 向全局表product中插入数据：   INSERT INTO product(prod\_id, prod\_nbr, prod\_name, base\_offer\_id) VALUES(1001010, 9934, '固定电话', 1200);  INSERT INTO product(prod\_id, prod\_nbr, prod\_name, base\_offer\_id) VALUES(1001011, 9935, '移动电话', 1200);  INSERT INTO product(prod\_id, prod\_nbr, prod\_name, base\_offer\_id) VALUES(1001012, 9936, 'PSTN 电话', 1200);   * + - 1. 执行下面的SQL语句：   SELECT \* FROM prod\_inst a left join product b on a.prod\_id = b.prod\_id WHERE a.use\_cust\_id = 200124353; | | |
| 预期结果 | 上述 SQL 语句都能执行成功；  JOIN 语句执行结果正确； | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 库内分表

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 库内分表功能 | | |
| 权值 | 基础 | | |
| 测试目的与范围 | 测试组件的库内分表功能 | | |
| 预置条件 | 1. 用例4.1.2中的库表已存在 2. 在管理平台上删除prod\_inst表 | | |
| 测试过程 | 1. 根据3.3.3.2中的数据库脚本“*测试数据库脚本.sql*”,执行prod\_inst的建表语句 2. 配置分片算法Mod20和Mod2； 3. 将prod\_inst设为分片表库内分表，其中使用Mod20为分片表的分片算法，分片键为owner\_cust\_id，Mod2为库内分表的分组算法，库内分表分片键为prod\_id，分片数为2； 4. 向分片表库内分表prod\_inst中插入数据，查看各个数据是否落在正确的分片分组上： 5. INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334132, 1001010, 200124353, 10001, 9909, 200124353); 6. INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334133, 1001011, 200124353, 10001, 9909, 200124354); 7. INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334134, 1001020, 200124355, 10001, 9909, 200124355); 8. INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334134, 1001021, 200124355, 10001, 9909, 200124355); | | |
| 预期结果 | * + - 1. 配置分片表库内分表成功       2. 向分片表库内分表插入数据后，数据能正确的落入到各个分片的分组上，步骤4中语句1)，因为owner\_cust\_id=200124353对20取模余数为13，所以该数据会落在分片dn14上，因为分组键prod\_id=1001010对2取模为0，所以语句会落在分片dn14的第一个表上；       3. 步骤4语句2)会落地分片dn14的第二个表上；       4. 步骤4语句3),4)会分别落在分片dn16的第一个表和第二个表上 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 自定义分片数

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 支持自定义分片数测试 | | |
| 权值 | 基础 | | |
| 测试目的与范围 | 分布式数据库允许用户自定义表的分片数量及指定分片节点 | | |
| 预置条件 | 用例4.1.2中的库表已存在，启动数据库中间件 | | |
| 测试过程 | 1. 逻辑数据库bss\_crm\_test关联了20个分片 2. 新建逻辑表test1、test2、test3 3. test1设置存放在一个单片上、 test2设置存放在其中的10个片上、test3设置存放在其中的3个片上 4. 向test1、test2、test3插入数据 5. 检查数据存放情况 | | |
| 预期结果 | * + - 1. 设置test1、test2、test3的分片规则时可以指定分片数       2. 插入的数据能正确存放到指定的分片上 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

### Hint语法/透传

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | Hint语法测试 | | |
| 权值 | 扩展 | | |
| 测试目的与范围 | 测试组件的hint语法功能 | | |
| 预置条件 | 1. 用例4.1.2中的库表已存在，数据库中间件上存在分片表customer 2. 数据库中间件的后台物理库为逻辑主从式数据库 3. 主从库上都有分片表customer，为了验证读写分离功能，需保证逻辑主从数据库上的数据不一致 | | |
| 测试过程 | 1. 根据分布式数据库提供的hint语法，在数据库中间件的分片表customer上执行hint路由策略+sql语句，查看执行结果； 2. 根据分布式数据库提供的hint语法，在数据库中间件的分片表customer上执行hint强制读写分离+sql语句，查看执行结果； 3. 根据分布式数据库提供的hint语法，在数据库中间件的分片表customer上执行hint获取表的数据节点，查看执行结果； 4. 开启物理数据的慢日志，设置long\_query\_time=3，在数据库中间的上执行/\* for fun \*/ select sleep(4) from customer语句，查看物理库的慢日志。 | | |
| 预期结果 | 步骤1中的语句可以在指定的节点上执行成功  步骤2中的语句可以强制执行读写分离  步骤3中的语句可以获取指定的表的数据节点  步骤4中在物理库的慢日志中可以看到/\* for fun \*/ select sleep(10) from customer这条记录，语句前的注释没有被数据库中间层过滤 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 读写分离

#### 应用透明读写分离测试

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 读写分离 | | |
| 权值 | 基础 | | |
| 测试目的与范围 | 测试组件能否进行读写分离 | | |
| 预置条件 | * + - 1. 用例4.1.2中的库表已存在，数据库中间件上存在分片表customer       2. 数据库中间件的后台物理库为逻辑主从式数据库       3. 主从库上都有分片表customer，为了验证读写分离功能，需保证逻辑主从数据库上的数据不一致 | | |
| 测试过程 | 1. 在数据库节点上开启读写分离功能，设置为默认无事务写SQL在主节点执行，读SQL在从节点执行；事务内读写SQL都在主节点执行； 2. 在数据库中间件的分片表customer上只执行select(读)语句，查看其是否路由到只读实例(从库)上执行 3. 在数据库中间件的分片表customer上执行insert(写)语句，查看其是否路由到非只读实例(主库)上执行 4. 在数据库中间件上开启事务，并在分片表customer执行insert和select语句，均在主节点上执行 | | |
| 预期结果 | 读语句只路由到只读实例上去执行，写语句路由到非只读实例上执行  满足在数据库节点上开启读写分离功能，默认无事务写SQL在主节点执行，读SQL在从节点执行；事务内读写SQL都在主节点执行 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

#### 应用控制读写分离测试

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 读写分离 | | |
| 权值 | 基础 | | |
| 测试目的与范围 | 测试组件能否由应用控制读写分离 | | |
| 预置条件 | 用例4.1.2中的库表已存在，数据库中间件上存在分片表customer  数据库中间件的后台物理库为逻辑主从式数据库  主从库上都有分片表customer，为了验证读写分离功能，需保证逻辑主从数据库上的数据不一致 | | |
| 测试过程 | 1. 在数据库中间件上关闭读写分离功能   应用链接到数据库中间件上执行SQL时，由应用通过特殊SQL注释或开关控制SQL是到写节点上执行或到从节点上执行；  在分片表customer上测试 INSERT、SELECT等SQL的读写分离执行情况； | | |
| 预期结果 | 上述SQL均能正常执行；  应用控制的读写分离权限较高，会由应用的SQL注释决定读写分离情况； | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

## 可用性

### 数据库中间件集群高可用

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 数据库中间件集群高可用 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 测试数据库中间件集群内有节点异常崩溃时，数据库中间件集群仍可以正常对外提供服务 | | |
| 预置条件 | 1. 搭建数据库集群 2. 在1上部署数据库中间件集群，集群中包含3个节点 3. 启动组件的所有节点和服务 | | |
| 测试过程 | 1. 断开其中的一个节点（kill进程、断开网络或宕机），另外两个节点可以提供正常服务 2. 执行DML SQL语句，查看结果是否符合预期 3. 再次断开一个节点（kill进程、断开网络或宕机），剩下的一个节点可正常提供服务 4. 执行DML SQL语句，查看结果是否符合预期 | | |
| 预期结果 | 节点异常崩溃时，集群可对外正常提供服务，DML SQL语句执行结果符合预期   * 第一个节点断开后，集群可对外正常提供服务 * 第二个节点断开后，集群可对外正常提供服务 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 数据库集群高可用

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 数据库集群高可用 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 测试数据库集群内主节点或备节点崩溃时，数据库集群仍可正常对外提供数据库服务 | | |
| 预置条件 | * + - 1. 搭建数据库集群，集群中包含1主+2备3个节点       2. 在1上部署数据库中间件集群，集群中至少有一个节点       3. 启动组件的所有节点和服务 | | |
| 测试过程 | 1. 断开其中一个备节点  2. 在该数据库集群对应的分片上执行DML SQL语句，查看结果是否符合预期  3. 恢复断开的备节点  4. 断开主节点  5. 在该数据库集群对应的分片上执行DML SQL语句，查看结果是否符合预期 | | |
| 预期结果 | 分布式数据库可正常对外提供数据库服务，应用无感知   * 断开备节点，可正常对外提供数据库服务，应用无感知 * 断开主节点，数据库备节点升级为主节点，自动进行主库切换，可正常对外提供数据库服务，应用无感知 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，，D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 数据库中间件节点异常恢复

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 数据库中间件节点异常恢复 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 测试数据库中间件集群中的节点崩溃后， 能否在短时间内重启并接入集群，对外正常提供服务 | | |
| 预置条件 | 1. 搭建数据库集群  2．在1上部署数据库中间件集群，集群中包含3个节点  3．启动组件的所有节点和服务 | | |
| 测试过程 | 1. 断开其中一个节点（kill进程） 2. 重新启动该节点 3. 执行DML SQL语句，请求能均匀分配到刚恢复的节点，并能正确执行 | | |
| 预期结果 | 集群中节点在崩溃后，1分钟中内能完成重启并加入到了集群中，均匀分担集群负载，对外正常提供服务。 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 数据库节点异常恢复

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 数据库节点异常恢复 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 测试数据库集群中的节点崩溃后， 能否在短时间内重启并接入集群，对外正常提供服务 | | |
| 预置条件 | * + - 1. 搭建数据库集群，集群中包含1主+2备3个节点       2. 在1上部署数据库中间件集群，集群中至少有一个节点       3. 启动组件的所有节点和服务 | | |
| 测试过程 | 1. 断开其中一个备节点  2. 恢复断开的备节点  3. 在该数据库集群对应的分片上执行DML SQL语句，查看结果是否符合预期  4. 断开主节点  5. 恢复断开的主节点  6. 在该数据库集群对应的分片上执行DML SQL语句，查看结果是否符合预期 | | |
| 预期结果 | 集群中节点在崩溃后，5分钟中内能完成重启并加入到了集群中，对外正常提供服务。   * 断开备节点，5分钟中内完成节点恢复并正常提供服务 * 断开主节点，5分钟中内完成节点恢复并正常提供服务 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 过载保护

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 过载保护 | | |
| 测试目的与范围 | 测试数据库中间件能否对前端连接数进行限制，能否在后端数据库过载时提供保护功能，防止故障蔓延 | | |
| 权值 | 核心 | | |
| 预置条件 | 1. 用例4.1.2中的库表已存在 | | |
| 测试过程 | 1. 设置数据库中间件的前端连接数限制为100 2. 执行客户端测试程序让前端连接数超过100，查看测试程序返回的结果 3. 停止连接数测试 4. 开启过载保护开关并设置相应的参数值 5. 启动压测程序进行压测（执行一些慢SQL） 6. 查看客户端程序执行结果是否有请求直接返回“过载异常” 7. 查看数据库中间件日志是否有过载拦截相关的日志信息 | | |
| 预期结果 | 可设置连接数限制，超过限制后拒绝连接并返回类似“连接数超出最大限制数”错误，后端数据库发生过载时部分请求直接在数据库中间件返回，不发向后端数据库执行。  1.设置前端连接数限制，超过限制拒绝连接并返回错误  2. 后端数据库发生过载时部分请求直接在数据库中间件返回，不发向后端数据库执行 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 数据库中间件集群分组

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 数据库中间件集群分组 | | |
| 权值 | 扩展 | | |
| 测试目的与范围 | 测试数据库中间件集群内的节点能否进行分组操作，按分组级别进行流控，提供差异化服务，保障核心业务访问的稳定性 | | |
| 预置条件 | 1. 搭建数据库中间件集群，集群中包含6个节点 2. 数据库集群为逻辑主从数据库 | | |
| 测试过程 | 1. 在管理平台新建三个分组， 分别为分组一、分组二、分组三  2. 将6个节点分别加入到三个分组中，每个分组包含2个节点  3. 设置分组参数， 三个组对应到三个不同的SLB服务器、 三个组设置不同的前端连接数限制（10、20、30）、设置不同的后端连接数（10、20、30）  4. 将分组一设置为透明读写分离模式  5. 分别对三个分组执行客户端测试程序让前端连接数超过设置的值，查看测试程序返回的结果  6. 对分组一进行读写分离测试， 事务外的读语句可自动发向读节点执行 | | |
| 预期结果 | 1.不同分组可设置不同的接数限制，超过限制拒绝连接并返回错误  2. 分组一可实现透明读写分离，事务外的读语句可自动发向读节点执行 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

## 可靠性

### 从库网络断开

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称: | 数据库主从节点数据一致性 | | |
| 权值 | 核心 | | |
| 测试目的与范围: | 测试从库网络断开，主从数据是否保持一致 。 | | |
| 预置条件: | 1. 主从数据库正常运行； 2. 向主库不断插入数据 3. 关闭任意一从库的网络； | | |
| 测试过程: | 1. 确认主从数据库运行正常； 2. 向主库中不断插入数据 3. 关闭任意一从库的网络（执行service network stop命令）并停止向主库插入数据； 4. 分别查看并记录主库和从库的数据； 5. 重新开启从库的的网络服务（执行service network start命令）； 6. 查看从库slave是不是自动连接上主库； 7. 查看主从数据是否一致。 | | |
| 预期结果: | 1. 从库网络重启之后，能主动连接主库  2. 最终主从数据一致 | | |
| 测试结果: | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 主库网络断开

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称: | 数据库主从节点数据一致性 | | |
| 权值 | 核心 | | |
| 测试目的与范围: | 测试主库网络断开，主从数据是否保持一致 。 | | |
| 预置条件: | 1. 主从数据库正常运行； 2. 向主库不断插入数据 3. 关闭主库的网络； | | |
| 测试过程: | 1. 确认分布式数据库运行正常； 2. 向主库中不断插入数据 3. 关闭主库的网络并停止向主库插入数据； 4. 分别查看并记录原主库和从库的数据； 5. 查看新的主库状态； 6. 开启原主库的网络服务； 7. 查看原主库和新主库的状态 8. 查看主从数据是否一致。 | | |
| 预期结果: | 主库网络断开，主从数据一致   1. 主库网络断开后，从库提升为主库 2. 从库提升为新主库后，新的主从数据一致 3. 当原主库网络恢复后，会自动转成从库 4. 原主库和新主库数据最终一致 | | |
| 测试结果: | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，，D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 主库数据库实例shutdown

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称: | 数据库主从节点数据一致性 | | |
| 权值 | 核心 | | |
| 测试目的与范围: | 测试主库shutdown，主从数据是否保持一致 。 | | |
| 预置条件: | 1. 主从数据库正常运行； 2. 向主库不断插入数据 3. 关闭主库实例； | | |
| 测试过程: | 1. 确认分布式数据库运行正常； 2. 向主库中不断插入数据； 3. 关闭主库数据库（执行数据库的shutdown管理命令），并停止插入数据； 4. 查看新的主从数据状态； 5. 重新启动原主数据库实例； 6. 查看原主库和新主库的状态 7. 查看主从数据是否一致。 | | |
| 预期结果: | 主库shutdown，从库升级为主库，主从数据一致   1. 主库shutdown，从库提升为主库 2. 从库提升为新主库后，新的主从数据一致 3. 当原主库重新启动之后，会自动转成从库 4. 原主库和新主库数据最终一致 | | |
| 测试结果: | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 主库服务器crash

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称: | 数据库主从节点数据一致性 | | |
| 权值 | 核心 | | |
| 测试目的与范围: | 测试主库服务器crash，主从数据是否保持一致 。 | | |
| 预置条件: | 1. 主从数据库正常运行； 2. 向主库不断插入数据 3. 关闭主库的网络； | | |
| 测试过程: | 1. 确认分布式数据库运行正常； 2. 向主库中不断插入数据； 3. 拔除数据库主机节点电源(执行halt –p命令)； 4. 查看新的主从数据状态； 5. 重新启动原主库所在的服务器和数据库； 6. 查看原主库和新主库的状态 7. 查看主从数据是否一致。 | | |
| 预期结果: | 主库服务器crash，从库升级为主库，主从数据一致   1. 主库服务器crash，从库提升为主库 2. 从库提升为新主库后，新的主从数据一致 3. 当原主库启动之后，自动转成从库 4. 原主库和新主库数据最终一致 | | |
| 测试结果: | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 疲劳测试

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称: | 分布式数据库疲劳测试 | | |
| 权值 | 核心 | | |
| 测试目的与范围: | 测试分布式数据库的运行稳定性。检测其是否能在高负载的情形之下（系统最大事务处理数的80%，CPU占用率达到70%左右），长时间地稳定运行 | | |
| 预置条件: | 1. 分布式数据库正常运行 2. 在系统中成功安装测试工具软件JMETER 3. 确认数据库内的测试表数据已导入 | | |
| 测试过程: | 1. 开启JMETER以300个线程读写分布式数据库 2. 观察分布式数据库的系统负载，确保系统负载达到一定程度（数据库节点的CPU负载达到70%左右）。如未达到，调整JMETER的测试参数设置 3. 疲劳测试24小时以上 4. 通过命令、日志和JMETER图形报表的方式察看分布式数据库在24小时之中的运行状态，是否有处理异常和剧烈的性能波动。 | | |
| 预期结果: | 分布式数据库平稳运行，无异常，无剧烈的性能波动   1. JMETER图形报表中显示分布式数据库不存在剧烈的性能波动 2. JMETER显示处理失败的事务没有堆积 3. 分布式数据库的运行状态一直正常 | | |
| 测试结果: | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 分布式事务测试（正常情形）

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称: | 分布式事务数据最终一致性测试 | | |
| 权值 | 基础 | | |
| 测试目的与范围: | 测试分布式数据库是否提供的分布式事务支持，实现分布式事务的数据最终一致性 | | |
| 预置条件: | 1. 用例4.1.2中的库表已存在，分布式数据库正常运行，且开启分布式事务支持的开关 2. 数据库中间件关联的数据库集群有5个，逻辑库test\_ddl上有20个分片 3. 分片表prod\_inst为空表 4. 准备好统一的测试脚本模板   start;  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334133, 1001010, 200124350, 10001, 9909, 200124354);  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334133, 1001010, 200124355, 10001, 9909, 200124354);  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334133, 1001010, 200124360, 10001, 9909, 200124354);  commit/rollback; | | |
| 测试过程: | 1. 在分布式数据库中执行测试脚本 2. 查看分布式事务的数据最终一致性是否得到保证 | | |
| 预期结果: | 三条记录同时存在于相应的数据库节点（COMMIT）或者三条记录同时不存在于预期的数据库节点（ROLLBACK）   1. 测试脚本执行成功 2. 三条记录同时插入相应的数据库节点成功或者三条记录同时插入失效 | | |
| 测试结果: | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 分布式事务测试（异常情形）

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称: | 分布式事务数据最终一致性测试 | | |
| 权值 | 基础 | | |
| 测试目的与范围: | 测试分布式数据库是否提供的分布式事务支持，实现分布式事务的数据最终一致性 | | |
| 预置条件: | 1. 用例4.1.2-中的库表已存在，分布式数据库正常运行，且开启分布式事务支持的开关 2. 数据库中间件关联的数据库集群有5个，逻辑库test\_ddl上有20个分片 3. 分片表prod\_inst为空表 4. 准备好统一的测试语句   start;  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334133, 1001010, 200124350, 10001, 9909, 200124354);  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334133, 1001010, 200124355, 10001, 9909, 200124354);  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334133, 1001010, 200124360, 10001, 9909, 200124354);  commit;   1. 确认测试脚本中用到的测试数据行owner\_cust\_id=200124350，200124355，200124360，根据当前的分片规则，会插入到不同的数据库分片上。 | | |
| 测试过程: | 1. 在分布式数据库中执行测试语句   start;  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334133, 1001010, 200124350, 10001, 9909, 200124354);  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334133, 1001010, 200124355, 10001, 9909, 200124354);  INSERT INTO prod\_inst(prod\_inst\_id, prod\_id, owner\_cust\_id, address\_id, create\_ord\_id, use\_cust\_id) VALUES(2002016122334133, 1001010, 200124360, 10001, 9909, 200124354);  确认DML语句执行成功   1. 断开测试第三个插入语句所在的数据库节点主机的网络连接 2. 在分布式数据库中执行COMMIT语句 3. 查看分布式数据库的报错情况 4. 恢复测试第三行插入语句所在的数据库节点主机的网络连接 5. 查看分布式事务的数据最终一致性是否得到保证 | | |
| 预期结果: | 三条记录同时插入相应的数据库节点成功或者三条记录同时插入失败 | | |
| 测试结果: | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

## 监控/运维

### 全局统一管理配置

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 全局统一管理配置 | | |
| 权值 | 基础 | | |
| 测试目的与范围 | 分布式数据库是否对配置进行了集中存放，统一管理 | | |
| 预置条件 | 1. 数据库中间件已部署完成 | | |
| 测试过程 | 1. 查看数据库中间件配置文件信息， 管理中间件配置信息 2. 启动数据库中间件 | | |
| 预期结果 | 数据库中间件配置信息修改成功 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 实时监控

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 实时监控 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 分布式数据库提供实时监控功能 | | |
| 预置条件 | 1. 启动管理平台 2. 部署并启动2-3个数据库中间件、2-3个数据库节点 | | |
| 测试过程 | 1. 使用管理平台的实时监控功能，查看TPS、CPU、前后端连接的实时监控 | | |
| 预期结果 | 管理平台可以正常查看TPS、CPU、前后端连接等实时监控信息 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 在线扩容

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 在线扩容（分片数不变） | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 分布式数据库提供在线扩容功能 | | |
| 预置条件 | 1. 部署并且启动2个数据库节点A和B，数据库节点A包含2个数据分片a和b 2. 部署并启动1个数据库中间件 3. 所使用的逻辑库中的分片表中有1000W数据 4. 启动数据迁移服务 | | |
| 测试过程 | 1. 使用在线扩容功能，在平台上操作将数据库节点A的数据分片b，扩容到数据库节点B上 2. 使用测试工具不间断对分布式数据库进行CRUD操作 3. 查看在扩容操作整个过程中的CRUD执行状态 4. 扩容结束后，对比数据库节点B的数据分片b与数据库节点A的数据分片b | | |
| 预期结果 | 扩容过程中，分布式数据库可以正常的提供服务，但在很短时间内仅能提供查询操作  数据库节点B的数据分片b包含数据库节点A的数据分片b的所有数据  数据库节点A的数据库分片b上的数据最后自动清除 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 组件灰度发布

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 组件灰度发布 | | |
| 权值 | 扩展 | | |
| 测试目的与范围 | 分布式数据库支持组件灰度发布 | | |
| 预置条件 | 1. 启动管理平台 2. 准备2个不同版本数据库中间件A和B 3. 准备2个不同版本的数据库节点DB1和DB2 | | |
| 测试过程 | 1. 通过管理平台部署一个数据库中间件集群，发布DB1版本2个数据库节点，发布A版本2个数据库中间件，查看分布式数据库集群可用性 2. 使用测试工具不间断访问分布式数据库 3. 修改负载系数，针对其中1个数据中间件发布B版本 4. 查看数据库中间件发布过程中，集群是否可用性 | | |
| 预期结果 | 在数据库中间件发布过程中，集群保持可用 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 在线配置修改

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 在线配置修改 | | |
| 权值 | 基础 | | |
| 测试目的与范围 | 分布式数据库是否支持在线配置修改 | | |
| 预置条件 | 1. 启动管理平台 2. 部署并启动数据库中间件 | | |
| 测试过程 | 1. 通过管理平台查看数据库中间件连接闲置时间，将值设置为10s 2. 新建数据库连接到中间件，等待15秒，执行select 1操作查看是否重建连接或执行成功，修改wait\_timeout值设置为20s 3. 等待15秒，执行select 1操作查看是否重建连接或执行成功 4. 通过管理平台查看数据库中间件最大连接数，将值设置为5，利用mysql客户端，创建6个连接，查看是否创建成功 5. 修改最大连接数10，利用mysql客户端，创建6个连接，查看是否创建成功 | | |
| 预期结果 | 闲置时间为10s时，执行select 1发现连接丢失，闲置时间为20s时，执行select 1操作成功。  修改最大连接数前超过5个后创建失败，修改后创建成功 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，，D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 在线建表、建全局序列

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 在线建表、建全局序列 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 分布式数据库是否支持在线建表、建全局序列 | | |
| 预置条件 | 1. 部署并启动数据库中间件 | | |
| 测试过程 | 1. 在线建表查看是否可用(sql同用例4.1.2) 2. 在线建全局序列查看是否可用 | | |
| 预期结果 | 在线建表、全局序列全部可以正常使用 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，，D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 管理命令

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 管理命令 | | |
| 权值 | 基本 | | |
| 测试目的与范围 | 分布式数据库的管理命令功能 | | |
| 预置条件 | 1. 部署并启动数据库中间件 2. 创建可以访问管理端口的用户 | | |
| 测试过程 | 1. 通过数据库客户端和指定用户访问分布式数据库 2. 使用提供的文档中的管理命令执行查看连接状态、kill连接操作、查看当前执行SQL、线程状态，后端数据库服务状态等，查看结果是否符合预期 | | |
| 预期结果 | 管理命令执行的结果符合预期 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 统计分析

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 统计分析 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 测试能否对分布式数据库的运行情况进行统计分析 | | |
| 预置条件 | 1. 启动管理平台 2. 部署并启动数据库中间件 | | |
| 测试过程 | 1. 运行测试程序，对分布式数据库保持一定的访问量 2. 进入统计分析功能页面，能通过图表方式展示各种统计数据，包括：  * SQL执行统计：可以按小时、按天选择时间段进行统计，展示的数据包括SQL执行总数、广播SQL数、错误SQL数、以及各类型SQL的最大/最小耗时和平均耗时，还可展示慢SQL语句 * SQL访问统计：可以按小时、按天选择时间段进行统计，展示的数据包括最高TPS、最低TPS、最大耗时、最小耗时、平均耗时 * 事务统计：可以按小时、按天选择时间段进行统计，展示的数据包括事务总数、分布式事务总数 * 分片访问统计：可以按小时、按天选择时间段进行统计，展示的数据包括每个schema上的每个分片的访问次数、执行错误的次数、最大耗时、最小耗时、平均耗时 | | |
| 预期结果 | 能够直观、正确的按图表方式展示各类统计数据   1. SQL执行统计能直观、正确的按图表方式展示 2. SQL访问统计能直观、正确的按图表方式展示 3. 事务统计能直观、正确的按图表方式展示 4. 分片访问统计能直观、正确的按图表方式展示 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 扩展DDL语法测试

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | DDL语句扩展测试 | | |
| 权值 | 扩展 | | |
| 测试目的与范围 | 扩展 DDL 语法，能够支持通过脚本方式指定分片算法及建表、建序列、修改表结构等 | | |
| 预置条件 | 1. 用例4.1.2中的库表已存在 2. 在管理平台上删除 prod\_inst 表 | | |
| 测试过程 | * + - 1. 编写建表脚本，能够在建表脚本里面，通过DDL SQL指定 prod\_inst 表分片数量为20，并且采用Mod20分片算法。       2. 编写修改prod\_inst表结构脚本，增加字段 mod\_table\_date,类型为datetime()   ALTER TABLE prod\_inst ADD mod\_table\_data DATETIME DEFAULT NULL;   * + - 1. 编写建序列脚本，指定创建序列seq\_prod\_inst\_id，非循环序列，并且步长为1000，最小值为1。 | | |
| 预期结果 | 1. 执行步骤1脚本后，查看数据库节点，能够看到prod\_inst在 20 个分片上均创建了表  2. 执行步骤2脚本后，查看prod\_inst表结构修改成功  3. 执行步骤3脚本后，通过查询序列语句，能获取到seq\_prod\_inst\_id序列值 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 一键故障检测

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 一键检测分布式数据库故障或异常点 | | |
| 权值 | 扩展 | | |
| 测试目的与范围 | 提供一键检测功能，快速检测当前服务中各组件或模块的异常点，方便运维人员快速发现问题  检测范围包括：数据库节点运行状态、数据库中间件运行状态、统一配置服务、分片索引接口状态、全局序列接口状态、服务响应时延等 | | |
| 预置条件 | 启动组件服务 | | |
| 测试过程 | 1. 先启动服务，并确保能正常运行；  2. 手工停止部分数据库节点；  3. 运行一键检测功能，发现故障节点；  4. 重新启动上面停止的数据库节点；  3. 运行一键检测功能，发现服务正常； | | |
| 预期结果 | 1. 停止部分数据库节点后，通过一键检测功能能发现故障的节点，并反馈故障现象；  2. 重启数据库节点后，通过一键检测功能发现服务已全部正常运行； | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 数据备份/恢复

* + - 1. **单节点数据库全量备份/恢复**

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 单节点数据库的全量备份 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 验证分布式数据库的单节点数据库全量备份功能 | | |
| 预置条件 | 1. 分布式数据库正常运行； 2. 指定好目标备份存储主机及目标恢复主机 | | |
| 测试过程 | 1. 选定备份的目标数据库； 2. 配置备份参数：配置备份数据库实例的用户，数据库实例密码;备份存储主机;备份存储目录;备份存储主机用户名;备份存储主机密码；指定全量备份开始的时间；指定全量备份的周期 3. 保存备份配置，并执行； 4. 指定目标恢复主机，执行恢复操作 5. 查看数据恢复的结果 | | |
| 预期结果 | 分布式数据库单节点数据库全量备份和恢复成功，可在看到指定备份的机器的全备目录新生成的备份文件。 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

* + - 1. **单节点数据库增量备份/恢复**

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 单节点数据库的增量备份 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 分布式数据库的单节点数据库增量备份 | | |
| 预置条件 | 1. 分布式数据库正常运行 2. 指定好目标备份存储主机及目标恢复主机。 | | |
| 测试过程 | 1. 选定备份的目标数据库； 2. 配置备份参数：配置备份数据库实例的用户，数据库实例密码;备份存储主机;备份存储目录;备份存储主机用户名;备份存储主机密码等等 3. 保存备份配置，并执行； 4. 指定目标恢复主机，执行恢复操作 5. 查看数据恢复的结果 | | |
| 预期结果 | 分布式数据库单节点数据库增量备份和恢复成功，可在看到指定备份的机器的增备目录新生成的备份文件。 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

* + - 1. **指定时间点的指定数据库实例恢复**

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 指定时间点的指定数据库实例恢复 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 分布式数据库指定时间点的指定数据库实例恢复 | | |
| 预置条件 | 1. 分布式数据库正常运行 2. 指定好目标备份存储主机及目标恢复主机。 | | |
| 测试过程 | 1. 选定恢复目标数据库； 2. 配置恢复参数：类型为数据库实例恢复；目标数据库ip；目标数据库port；目标数据库配置文件绝对路径；目标数据库用户名为；目标数据库密码；策略选择基于时间；操作结束点为选取合理的指定恢复的时间点；指定临时目录； 3. 保存该配置，并执行指定时间点的指定数据库实例恢复任务； | | |
| 预期结果 | 可以在指定恢复的机器上看到指定端口号的数据库实例恢复到指定的时间点。 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

* + - 1. **指定时间点的指定单表恢复**

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 指定时间点的指定单表恢复 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 分布式数据库的指定时间点的指定数据库实例单表恢复 | | |
| 预置条件 | 1. 分布式数据库正常运行 2. 指定好目标备份存储主机及目标恢复主机。 | | |
| 测试过程 | 1. 选定恢复用的目标数据库； 2. 配置恢复参数：类型为单表恢复；待恢复数据库ip；待恢复数据库的port;登录数据库用户；登录密码；待回复的数据库；待恢复table；待恢复的新table名；目标数据库配置文件绝对路径；策略选择基于时间；指定临时目录； 3. 保存该配置，并执行指定时间点的指定表恢复任务； | | |
| 预期结果 | 可以在指定恢复的机器上看到指定端口号的数据库实例中的指定数据库中的指定表被恢复到指定的时间点，恢复为指定的新表名。 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

* + - 1. **数据库中间件配置的备份与恢复**

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 数据库中间件和数据库配置的备份与恢复 | | |
| 权值 | 扩展 | | |
| 测试目的与范围 | 数据库中间件配置的定时备份和恢复 | | |
| 预置条件 | 1. 分布式数据库正常运行 2. 指定好需要备份的配置信息所在的机器和路径以及存放配置信息的目标机器和路径。 | | |
| 测试过程 | 1. 执行定时任务对配置信息进行定时备份，查看目标机器的路径下是否有相关的配置信息 2. 恢复时，将目标机器的路径下的配置信息复制到数据库中间件的配置信息所在的机器的路径下 | | |
| 预期结果 | 可以对指定的配置信息进行备份以及恢复 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成D:不可用* | | |
| 备  注 |  | | |
| 测试人员 |  | 测试日期 |  |

## 安全性

### 用户模型和权限验证

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 用户模型和权限设置测试 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 测试组件的权限设置功能 | | |
| 预置条件 | 启动组件服务 | | |
| 测试过程 | 1. 创建用户user1和角色role1和role2，role1对逻辑库1有所有的权限，role2对逻辑库2中的表1有所有的权限 2. 把role1授权给user1 3. 使用user1访问中间层，在逻辑库1上执行select操作，查看结果 4. 使用user1访问中间层，在逻辑库2上执行select操作，查看结果 5. 撤销角色role1对user1的授权 6. 使用user1访问中间层，在逻辑库1上执行select操作，查看结果 7. 把role2授权给user1 8. 使用user2访问中间层，在逻辑库2的表1上执行select操作，查看结果 9. 使用user2访问中间层，在逻辑库2的表2上执行select操作，查看结果 10. 撤销角色role2对user1的授权 11. 使用user2访问中间层，在逻辑库2的表1上执行select操作，查看结果 12. 把role1，role2授权给user1 13. 使用user1访问中间件，在逻辑库1上执行select操作，查看结果 14. 使用user1房中中间件，在逻辑库2的表1上执行select操作，查看结果 15. 使用user1房中中间件，在逻辑库2的表2上执行select操作，查看结果 | | |
| 预期结果 | 步骤3中可以显示正确的结果  步骤4中的操作被拒绝  步骤6中的操作被拒绝  步骤8中可以显示正确的结果  步骤9中的操作被拒绝  步骤11中的操作被拒绝  步骤13可以显示正确的结果  步骤14可以显示正确的结果  步骤15中的操作被拒绝 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |



### 账号有效期和密码过期时间测试

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 账号有效期和密码过期时间测试 | | |
| 权值 | 扩展 | | |
| 测试目的与范围 | 分布式数据库能够设置账号有效期和密码过期时间，超过有效期的帐户不允许链接数据库，并返回错误 | | |
| 预置条件 | 启动分布式数据库服务 | | |
| 测试过程 | 1. 在分布式数据库上创建用户，并设置用户帐号有效期为3天，修改服务器的系统时间，调整为5天后，通过客户端连接分布式数据库；  2. 在分布式数据库上创建用户，并设置用户密码过期为30天，修改服务器的系统时间，调整为40天后，通过客户端连接分布式数据库； | | |
| 预期结果 | 1. 使用客户端连接分布式数据库失败，并且返回用户帐号过期错误；  2. 使用客户端连接分布式数据库失败，并且返回用户密码有效期过期错误 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

### SQL审计

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | SQL审计 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 根据审计规则对SQL进行拦截或输出审计报告，审计内容包括(select \*语句、delete/update语句不带where条件、delete/update语句带有order by,limit条件、select记录数过多、语句执行时间过长、truncate语句、跨库更新、DDL语句校验等) | | |
| 预置条件 | 1. 用例4.1.2中的库表已存在 2. 分片表customer中有5条记录 | | |
| 测试过程 | * + - 1. 配置SQL审计规则为select返回的记录数不超过4条，拦截方式为拒绝；       2. 在中间件的的逻辑库bss\_crm\_test的表customer执行下面的语句：   select \* from customer;   * + - 1. 打开select \* 语句审计开关，拦截方式配置为警告；       2. 在中间件的的逻辑库bss\_crm\_test的表customer执行下面的语句：   select \* from customer limit 1;   * + - 1. 其他的拦截测试同步骤1,2,3,4 | | |
| 预期结果 | 步骤2中语句执行报错，因为返回的的记录数超过了4条；  步骤3中语句的执行相关信息（执行用户、执行时间、用户地址、执行SQL等）记录到审计日志，以报表方式提供给用户查看 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 黑白名单

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 权限设置测试 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 测试组件的权限设置功能 | | |
| 预置条件 | * + - 1. 数据库中间件启动成功，正常提供服务 | | |
| 测试过程 | * + - 1. 将IP1设为黑名单，IP2设为白名单       2. 使用IP1访问中间层，并执行一些sql语句，查看结果       3. 使用IP2访问中间层，并执行一些sql语句,查看结果 | | |
| 预期结果 | 使用IP1访问中间层时，该请求被禁止，sql语句不能执行  使用IP2访问中间层时，可以正常访问，并能显示正确的结果 | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成， D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

## 性能

### CRUD/综合语句性能（单数据库中间件+单数据库）

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | CRUD/综合语句性能（单数据库中间件+单数据库） | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 测试单数据库中间件+单数据库各种语句所能达到性能峰值 | | |
| 预置条件 | 1. 已安装好1台数据库中间件,1台数据库实例； 2. 配置好数据库中间件和数据库实例，并且优化好相关参数； 3. Schema策略已配置； 4. CUSTOMER表无数据 | | |
| 测试过程 | 1. 启动测试线程，并发数从16->32->64->128，总共插入N1条数据，监控服务器状态并且输出每秒的TPS速度。 2. 启动测试线程，并发数从16->32->64->128，总共更新N2条数据，监控服务器状态并且输出每秒的TPS速度。 3. 启动测试线程，并发数从16->32->64->128，总共查询N3条数据，监控服务器状态并且输出每秒的TPS速度。 4. 启动测试线程，并发数从16->32->64->128，总共删除N4条数据，监控服务器状态并且输出每秒的TPS速度。 5. 启动测试线程，并发数从16->32->64->128，在以上四种同时混合状态下的服务器状态并且输出每秒TPS速度。 6. 启动测试线程，并发数分别稳定为20，50个，测试上述操作的时候TPS和服务器状态； (酌情增加线程数，测试出单数据库性能峰值。) | | |
| 预期结果 | 1. 记录每秒的TPS和线程数； 2. 记录执行完的总耗时； 3. 绘制出根据线程增长的TPS曲线图； 4. 记录步骤6部署模式下的平均调用时耗(ms) ； | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

### CRUD/综合语句性能—线性扩展能力1

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | CRUD/综合语句性能—线性扩展能力1 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 测试在数据库足够性能的情况下单数据库中间件 多数据库的线性扩展能力 | | |
| 预置条件 | 1. 已安装好1台数据库中间件,2台数据库实例； 2. 配置好数据库中间件和数据库实例，并且优化好相关参数； 3. Schema策略已配置； 4. CUSTOMER表无数据 | | |
| 测试过程 | 1. 在4.6.1的测试并发数基础上增加并发数，分别测试4.6.1的1-6个的场景。 | | |
| 预期结果 | 1. 记录每秒的TPS和线程数； 2. 记录执行完的总耗时； 3. 绘制出根据线程增长的TPS曲线图； 4. 记录步骤6部署模式下的平均调用时耗(ms) ； | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

### CRUD/综合语句性能—线性扩展能力2

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | CRUD/综合语句性能—线性扩展能力2 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 测试在数据库足够性能的情况下数据库中间件的线性扩展能力 | | |
| 预置条件 | 1. 已安装好2台数据库中间件,4台数据库实例； 2. 配置好数据库中间件和数据库实例，并且优化好相关参数； 3. Schema策略已配置； 4. CUSTOMER表无数据 | | |
| 测试过程 | 1. 启动测试线程，并发数从16->32->64->128，总共插入N1条数据，监控服务器状态并且输出每秒的TPS速度。 2. 启动测试线程，并发数从16->32->64->128，总共更新N2条数据，监控服务器状态并且输出每秒的TPS速度。 3. 启动测试线程，并发数从16->32->64->128，总共查询N3条数据，监控服务器状态并且输出每秒的TPS速度。 4. 启动测试线程，并发数从16->32->64->128，总共删除N4条数据，监控服务器状态并且输出每秒的TPS速度。 5. 启动测试线程，并发数从16->32->64->128，在以上四种同时混合状态下的服务器状态并且输出每秒TPS速度。 6. 启动测试线程，并发数分别稳定为20，50个，测试上述操作的时候TPS和服务器状态； (酌情增加线程数，测试出单数据库性能峰值。) | | |
| 预期结果 | 1. 记录每秒的TPS和线程数； 2. 记录执行完的总耗时； 3. 绘制出根据线程增长的TPS曲线图； 4. 记录步骤6部署模式下的平均调用时耗(ms) ； | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 集成场景综合测试性能

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 集成场景性能测试 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 集成场景性能测试 | | |
| 预置条件 | 1. 已安装好2台数据库中间件,4台数据库实例； 2. 配置好统一访问层，数据库中间件和数据库实例，并且优化好相关参数； 3. Schema策略已配置； 4. 建议使用订单受理的测试脚本。 | | |
| 测试过程 | 使用订单测试脚本，启动测试线程，线程数分别稳定为20，50个，爬坡时间S秒，总共做N笔订单受理，测试上述操作的时候TPS和服务器状态； | | |
| 预期结果 | 1. 记录每秒的TPS和线程数； 2. 记录执行完的总耗时； 3. 绘制出根据线程增长的TPS曲线图； 4. 记录平均调用时耗(ms) ； | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |

### 全局序列获取性能

|  |  |  |  |
| --- | --- | --- | --- |
| 用例名称 | 全局序列获取性能 | | |
| 权值 | 核心 | | |
| 测试目的与范围 | 全局序列获取性能 | | |
| 预置条件 | 1. 已安装好2台数据库中间件,4台数据库实例； 2. 配置好统一访问层，数据库中间件和数据库实例，并且优化好相关参数； 3. Schema策略已配置， 4. 已经建好全局序列； | | |
| 测试过程 | 启动测试线程，线程数从16->32->64\_>128，不断获取序列，监控服务器状态并且输出每秒的TPS速度。  (酌情增加线程数) | | |
| 预期结果 | 全局序列生成性能 > 20wTPS | | |
| 测试结果与结论 | *量化的结果直接填结果值和度量单位*  *非量化结果给出如下的结果选项：*  *A:较好完成，B:基本完成，C:需要改进，D:不可用* | | |
| 备 注 |  | | |
| 测试人员 |  | 测试日期 |  |