# TiDB Range分区的测试方案

目录

**[1. 目的 2](#_Toc1182136506_WPSOffice_Level1)**

**[2. TiDB的Range分区功能概述 2](#_Toc1718229654_WPSOffice_Level1)**

**[3. 测试需求分析 3](#_Toc1073193569_WPSOffice_Level1)**

**[4. 测试建模 3](#_Toc449163030_WPSOffice_Level1)**

[4.1 功能特性划分 4](#_Toc1718229654_WPSOffice_Level2)

[4.2 功能特性测试分解 4](#_Toc1073193569_WPSOffice_Level2)

[4.3 测试类型说明 9](#_Toc449163030_WPSOffice_Level2)

[4.4 测试用例设计常用的工程方法 9](#_Toc678026005_WPSOffice_Level2)

[4.5 生成最终测试用例集 9](#_Toc1034835053_WPSOffice_Level2)

## 1. 目的

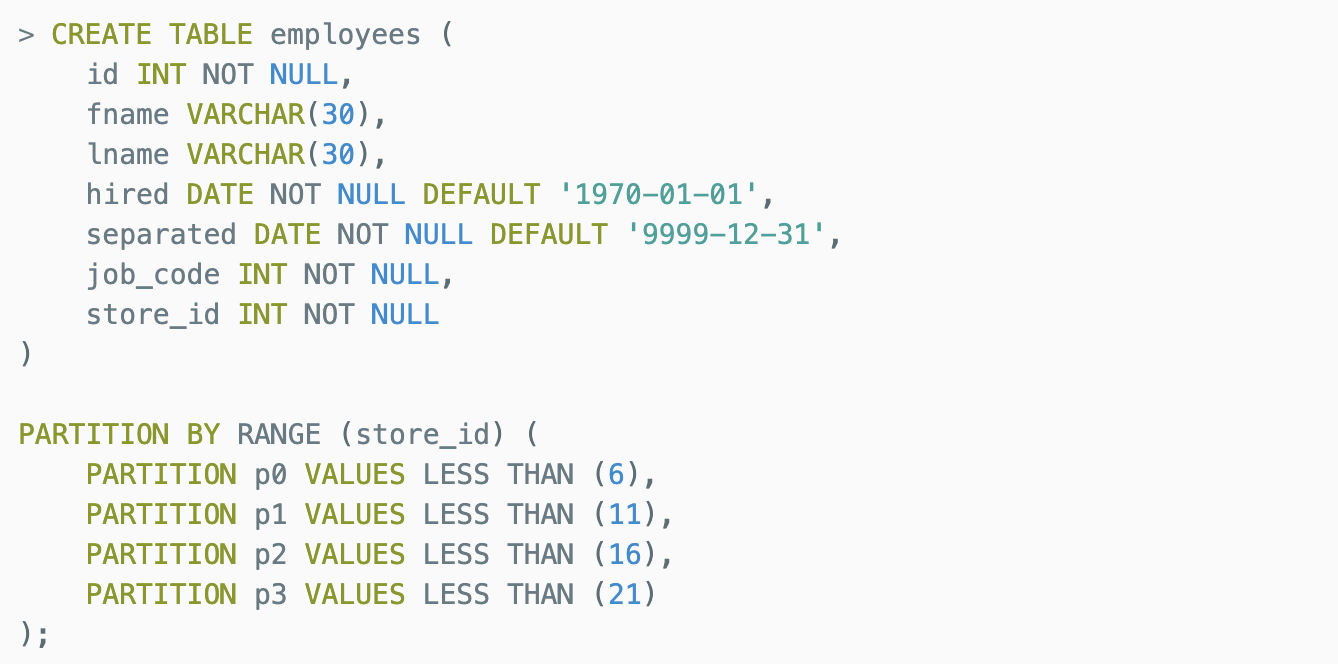
本文用于对TiDB的range分区功能进行测试方案设计。

## 2. TiDB的Range分区功能概述

TiDB支持的分区类型包括 Range 分区和 Hash 分区。Range 分区可以用于解决业务中大量删除带来的性能问题，支持快速删除分区。Hash 分区则可以用于大量写入场景下的数据打散。

一个表按 Range 分区是指，对于表的每个分区中包含的所有行，按分区表达式计算的值都落在给定的范围内。Range 必须是连续的，并且不能有重叠，通过使用 VALUES LESS THAN 进行定义。

以下是创建表及Range分区的示例：



## 3. 测试需求分析

作为一个用户，希望通过Range分区功能，以便能够在以下场景提升性能：

* 删除旧数据时，通过删除或者清空分区以便快速删除分区中的数据
* 频繁查询分区使用的列时，通过分区裁剪来提升查询性能
* 在对包含时间或者日期的列进行过滤查询时，即使为这类的列创建了索引，查询效率提升并不明显，而希望通过Range分区及分区裁剪可大幅提升查询性能

因此，验证Range分区是否能达到预期的目的，是测试的一个重点之一。

为了提供Range分区功能，需要支持对分区进行管理，并支持对分区中的数据进行管理和查询。

同时，TiDB的Range分区功能用法跟MySQL类似，但又有一些限制和不同，需要进行针对性的测试。

另外，TiDB是一个分布式的系统，需要完成针对性的分布式场景专项测试工作。

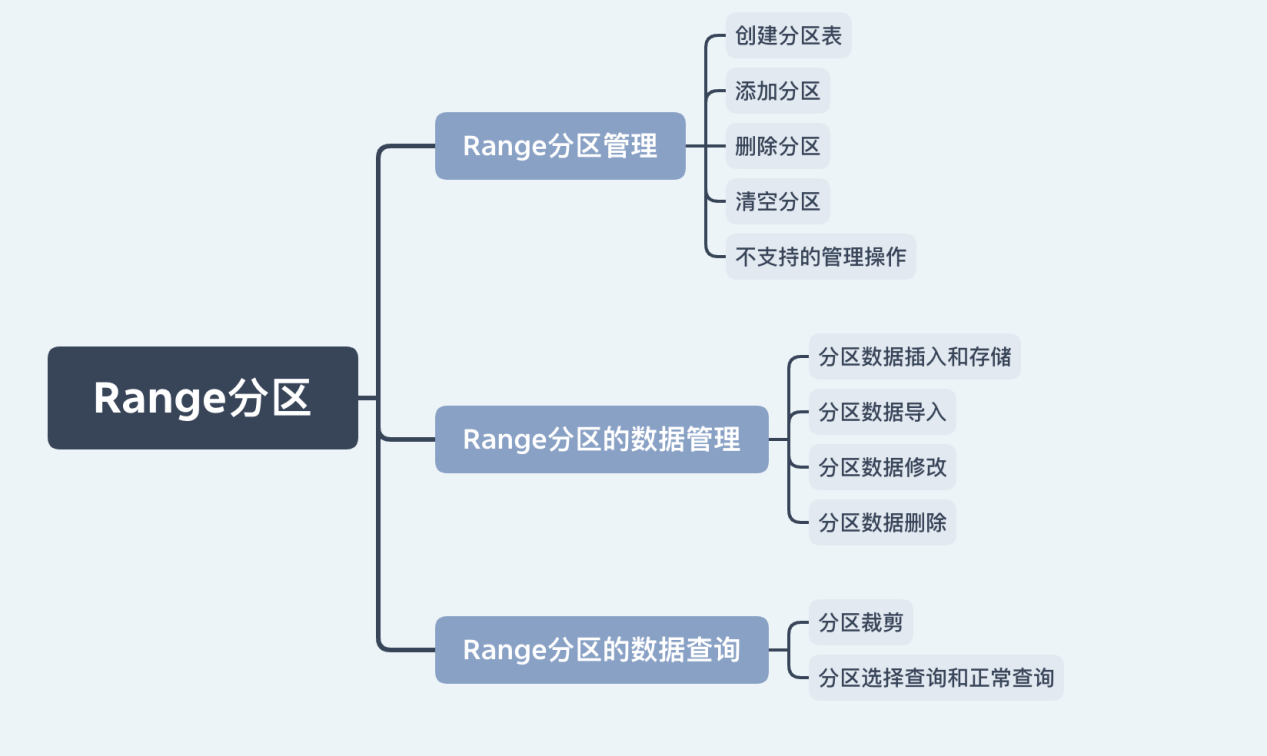
## 4. 测试建模

测试建模是指通过对被测对象进行全面和系统的分析和划分，构建测试模型并将测试思路或内容形成条理清晰的文档。

本节主要从功能特性划分、功能特性测试分解、测试类型说明等方面开展测试建模。

### 4.1 功能特性划分

根据Range分区的需求和对用户提供的功能，按照以下结构划分Range分区的功能特性：



### 4.2 功能特性测试分解

结合第4.1节梳理的功能特性划分，以及在“TiDB学习笔记.docx”中整理的Range分区功能点，可进一步对逐个功能特性进行测试分解。

测试分解就是将各个功能特性分解为详细的测试面和基础的测试点。

4.2.1 Range分区管理

4.2.1.1 创建分区表

支持在整数类型的列上直接指定 Range分区：

* INTEGER/INT
* TINYINT
* SMALLINT
* MEDIUMINT
* BIGINT

Range分区可使用分区表达式，能够使用的函数包括：

* ABS()
* CEILING()
* DATEDIFF()
* DAY()
* DAYOFMONTH()
* DAYOFWEEK()
* DAYOFYEAR()
* EXTRACT() (see EXTRACT() function with WEEK specifier)
* FLOOR()
* HOUR()
* MICROSECOND()
* MINUTE()
* MOD()
* MONTH()
* QUARTER()
* SECOND()
* TIME\_TO\_SEC()
* TO\_DAYS()
* TO\_SECONDS()
* UNIX\_TIMESTAMP() (with TIMESTAMP columns)
* WEEKDAY()
* YEAR()
* YEARWEEK()

创建分区的约束和限制:

* Range 必须是连续的，并且不能有重叠
* 仅能使用单列用于Range划分
* 分区表的每个组件和唯一键，必须包含分区表达式中用到的所有列；如果既没有主键，也没有唯一键，可以创建Range分区
* 环境变量 tidb\_enable\_table\_partition 可以控制是否启用分区表功能。如果该变量设置为 off，则建表时会忽略分区信息，以普通表的方式建表。

另外，可以使用MAXINT来定义最后一个分区。

4.2.1.2 添加分区

支持在整数类型的列上直接指定 Range分区：

* INTEGER/INT
* TINYINT
* SMALLINT
* MEDIUMINT
* BIGINT

Range分区可使用分区表达式，能够使用的函数包括（考虑到在创建分区时已覆盖了这些场景，测试用例设计时选取部分场景即可）：

* ABS()
* CEILING()
* DATEDIFF()
* DAY()
* DAYOFMONTH()
* DAYOFWEEK()
* DAYOFYEAR()
* EXTRACT() (see EXTRACT() function with WEEK specifier)
* FLOOR()
* HOUR()
* MICROSECOND()
* MINUTE()
* MOD()
* MONTH()
* QUARTER()
* SECOND()
* TIME\_TO\_SEC()
* TO\_DAYS()
* TO\_SECONDS()
* UNIX\_TIMESTAMP() (with TIMESTAMP columns)
* WEEKDAY()
* YEAR()
* YEARWEEK()

创建分区的约束和限制:

* 仅能使用单列用于Range划分
* 分区表的每个组件和唯一键，必须包含分区表达式中用到的所有列；如果既没有主键，也没有唯一键，可以创建Range分区
* 添加新分区时，只能在分区列表的最后面添加，如果是添加到已存在的分区范围则会报错
* Range 必须是连续的，并且不能有重叠
* 环境变量 tidb\_enable\_table\_partition 不影响添加分区的操作。

同时，对于已经有大量数据的表，添加新分区时，需要关注对表数据查询的影响。

4.2.1.3 删除分区

Range分区的删除，删除分区后该分区的数据也会一同被删除。

分区被删除之后，TiKV会通过GC回收存储空间。

4.2.1.4 清空分区

清空某个Range分区，可以快速清空该分区包含的所有数据。

数据被清空之后，TiKV会通过的GC回收存储空间。

4.2.1.5 不支持的管理操作

相比MySQL，不支持的管理操作有：

* REORGANIZE PARTITION
* 修改 Range 分区表的范围，合并分区，交换分区 都不支持

对于不支持的管理操作，验证操作执行有合理的提示即可

4.2.2 Range分区的数据管理

4.2.2.1 分区数据插入和存储

关注通过不同调用方式插入数据是否正常，并考虑到流程的相似性，数据的其他管理操作不再单独验证该流程：

* 利用MySQL CLI调用 INSERT语句
* 利用MySQL Driver插入数据等

对异常数据插入的处理，需要进行单独的验证：

* 如果插入一行到 Range 分区表，它的分区列的计算结果是 NULL，那么这一行会被插入到最小的那个分区。
* 插入不属于任何Range分区的数据，会报错

关注使用Range分区时，插入数据性能是否有显著的下降。

同一个表的不同Range分区在TiKV中对应不同的tableID，可以关注同一个Range分区对应的不同Region Leader副本是否在TiKV中相对均匀的分布存储。

4.2.2.2 分区数据导入

MySQL的Load Data功能可以导入数据到包含分区的表中，但不允许指定数据属于哪个分区。

4.2.2.3 分区数据修改

分区中的数据仍然可以被修改，包括用于Range计算的列和其他列。

需要关注修改数据时，数据所在Range分区发生变化，这是按照分区和查询和删除是否有效。

4.2.2.4 分区数据删除

关注分区的数据可以仍然可以通过DELETE FROM命令正常删除，。

4.2.3 Range分区的数据查询

4.2.3.1 分区裁剪

在SELECT查询时，分区裁剪的一些限制：

* 规则优化是在查询计划的生成阶段，对于执行阶段才能获取到过滤条件的场景，无法利用分区裁剪的优化
* 查询条件无法下推到 TiKV 的表达式，不支持分区裁剪
* 对于 Range 分区类型，分区表达式必须是 col 或者 fn(col) 的简单形式，查询条件是 >、<、=、>=、<= 时才能支持分区裁剪。如果分区表达式是 fn(col) 形式，还要求 fn 必须是单调函数，才有可能分区裁剪。目前只有：unix\_timestamp、to\_days和floor(unix\_timestamp(ts))

因此，对于能够做分区裁剪的场景，验证分区裁剪是正确生效的。而对于不支持分区裁剪的场景，数据查询仍然可以正常工作。

同时，需要验证对于是否进行分区裁剪，EXPLAIN执行计划分析是否正确。

4.2.3.2 分区选择

分区选择示例：

SELECT \* FROM employees PARTITION (p0, p2)

说明：

* 可与where 条件，以及 ORDER BY 和 LIMIT 等选项。使用 HAVING 和 GROUP BY 等聚合选项的组合
* 也可用于INSERT INTO ... SELECT FROM 语句中

因此，可以设计一组测试用例，符合场景的SELECT查询语句能够配合分区选择正常工作。

### 4.3 测试类型说明

在设计最终测试用例时，主要按照以下类型来考虑测试点，并生成测试用例：

* 功能测试：从业务功能的角度来考虑测试点。
* 容错测试：从非法数据和非法操作等角度来考虑测试点。
* 故障注入测试：对关键流程注入故障，以验证系统的可靠性。
* 分布式场景测试：从数据一致性、可用性、分区容忍性、负载均衡，以及扩容和缩容等场景来验证系统的分布式特性。
* 功能交互测试：跟整个系统其他子特性如果有功能上的交互，需要设计测试点进行相关验证。
* 性能测试：通过高用户并发、大数据量、负载测试和长时间稳定性等测试来验证系统的关键性能指标是否符合要求。
* 安全性测试：对敏感信息泄露、权限控制、加密通信等角度考虑测试点。
* DFX测试：对系统的可见性、可靠性和可维护性，对功能的易用性和可测试性等方面考虑测试点。

### 4.4 测试用例设计常用的工程方法

测试用例设计时运用工程方法，可以在保证测试用例质量和覆盖率的前提下，尽量减少测试用例的数量。

测试用例设计常用的工程方法有：

* 边界值和等价类
* 因果图和判定表
* 错误推测法
* Pair-wise正交分析法：该方法通过对各个独立因子进行正交分析，在生成测试用例时利用Pair-wise方法在保证覆盖率的情况下可以大幅缩减测试用例的数量；一般采用双因子组合，如果对质量要求更高则采用3因子组合。

### 4.5 生成最终测试用例集

基于功能特性测试分解得到的测试面和基础测试点，再利用测试用例设计的常用工程方法，并从合适的测试类型来考虑更详细的测试点，就可以转化为仅包含标题的测试用例了。

最后，将仅包含标题的测试用例完善为包含预置条件、测试步骤和预期结果等字段的最终测试用集。

详见：“Range分区-测试用例.xmind”和“Range分区-最终测试用例集.xlsx”。