53 倍性能提升！TiDB 全局索引如何优化分区表查询?

PingCAP 2025-02-17 20 阅读12分钟

智能总结 复制 重新生成

文章介绍了 TiDB 全局索引，包括其工作原理、发展历程、语法、优势、限制与注意事项、性能测试数据、最佳实践等。全局索引打破索引与分区一对一映射，提升跨分区查询效率，适用特定场景，但也影响部分 DDL 性能等。合理使用能优化性能，满足业务需求。

关联问题: 全局索引如何创建 全局索引适用哪些 全局索引有何限制

基于该文章内容继续向AI提问

导读

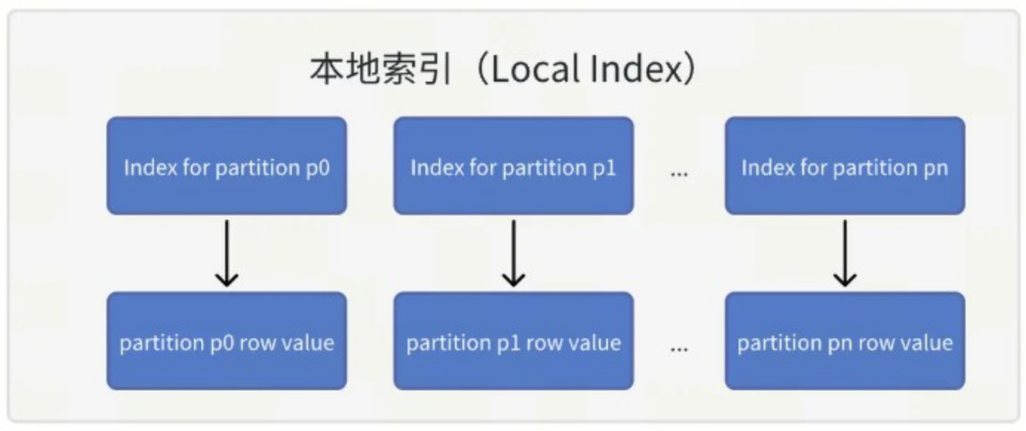
TiDB 全局索引在分区表中提供了一种优化查询性能的新方式。与本地索引不同，全局索引通过打破索引与分区的一对一映射关系，提升了跨分区查询的效率。本文将详细介绍 TiDB 全局索引的工作原理、发展历程以及创建方法，并通过性能测试和最佳实践，帮助用户更好地理解和应用全局索引，提高数据库的查询性能和整体效率。

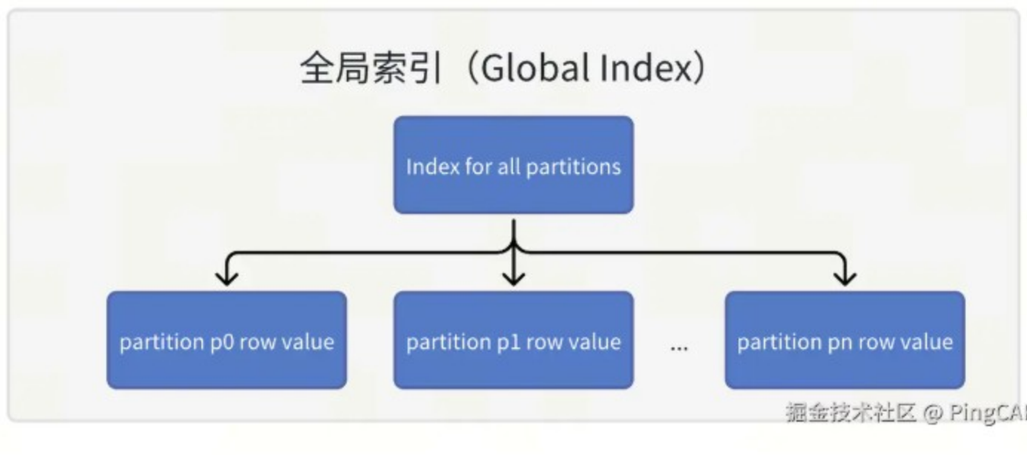
什么是 TiDB 全局索引

在 TiDB 中，全局索引是一种定义在分区表上的索引类型，它允许索引分区与表分区之间建立一对多的映射关系，即一个索引分区可以对应多个表分区。这与 TiDB 早期版本中的本地索引（Local Index）不同，本地索引的索引分区与表分区之间是一对一的映射关系，即一个分区对应一个局部的索引块。

全局索引能覆盖整个表的数据，使得主键和唯一键在不包含分区键的情况下仍能保持全局唯一性。此外，全局索引可以在一次操作中访问多个分区的索引数据，而无需对每个分区的本地索引逐一查找，显著提升了针对非分区键的查询性能。

下图简单展示了本地索引和全局索引的区别





TiDB 全局索引的发展历程

- **v7.6.0 版本之前**： TiDB 仅支持分区表的本地索引。这意味着，对于分区表上的唯一键，必须包含表分区表达式中的所有列。如果查询条件中没有使用分区键，那么查询将不得不扫描所有分区，这会导致查询性能下降。
- **v7.6.0 版本**： 引入了系统变量 tidb_enable_global_index ，用于开启全局索引功能。然而，当时该功能仍在开发中，不推荐用户启用。
- **v8.3.0 版本**： 全局索引功能作为实验性特性发布。用户可以通过在创建索引时显式使用 GLOBAL 关键字来创建全局索引。
- **v8.4.0 版本**： 全局索引功能正式成为一般可用（GA）特性。用户可以直接使用 GLOBAL 关键字创建全局索引，而无需再设置系统变量 tidb_enable_global_index 。从这个版本开始，该系统变量被弃用，并且始终为 ON 。
- **v8.5.0 版本**： 全局索引功能支持了包含分区表达式中的所有列。
- **v9.0.0 版本**： 全局索引功能支持了非唯一索引的情况。在分区表中，除聚簇索引外都可以被创建为全

PingCAP LV.5
@http://pingcap.com
作者榜No.9 优秀作者
789 文章 549k 阅读 7.6k 粉丝
关注 私信

目录 收起

局索引。

TiDB 全局索引的语法

在 TiDB 中，创建全局索引（Global Index）时，可以在 CREATE INDEX 或 ALTER TABLE 语句中使用 GLOBAL 关键字，或在建表时通过 GLOBAL 关键字或 /*T![global_index] GLOBAL */ 注释指定。

创建全局索引的语法：

```
Plaintext
```



```
CREATE [UNIQUE] INDEX index_name ON table_name (column_list) [GLOBAL];
ALTER TABLE table_name ADD [UNIQUE] INDEX index_name (column_list) [GLOBAL];
```

示例：

1. **创建全局唯一索引：**

```
Plaintext
```



```
CREATE UNIQUE INDEX idx_global ON employees (email) GLOBAL;
```

此语句在 employees 表的 email 列上创建一个全局唯一索引，确保每个电子邮件地址在整个表中唯一。

2. **添加全局索引：**

```
Plaintext
```



```
ALTER TABLE orders ADD INDEX idx_global_order_date (order_date) GLOBAL;
```

此语句向 orders 表添加一个名为 idx_global_order_date 的全局索引，索引列为 order_date 。

3. **在建表时创建全局索引：**

```
Plaintext
```



```
CREATE TABLE `sbtest` (
  `id` int NOT NULL,
  `k` int NOT NULL DEFAULT '0',
  `c` char NOT NULL DEFAULT '',
  KEY `idx1` (`k`) GLOBAL,
  KEY `idx2` (`k`) /*T![global_index] GLOBAL */
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
PARTITION BY HASH (`id`) PARTITIONS 5;
```

此语句在创建 sbtest 表时同时创建了两个名为 idx1 和 idx2 的全局索引，两个索引的索引列都为 k 。

TiDB 全局索引的优势

提升查询性能

全局索引能够有效提高检索非分区列的效率。当查询涉及非分区列时，全局索引可以快速定位相关数据，避免了对所有分区的全表扫描，可以显著降低 cop task 的数量，这对于分区数量庞大的场景尤为有效 。

经过测试，在分区数量为 100 的情况下，sysbench select_random_points 场景得到了 **53 倍** 的性能提升。

增强应用灵活性

全局索引的引入，消除了分区表上唯一键必须包含所有分区列的限制。这使得用户在设计索引时更加灵活，可以根据实际的查询需求和业务逻辑来创建索引，而不再受限于表的分区方案。这种灵活性有助于更好地优化查询性能，满足多样化的业务需求。

减少应用修改工作量

在数据迁移和应用修改过程中，全局索引可以减少对应用的修改工作量。如果没有全局索引，在迁移数据或修改应用时，可能需要调整分区方案或重写查询语句以适应索引的限制。有了全局索引之后，这些修改可以被避免，从而降低了开发和维护成本。

如在将 Oracle 数据库中的某张表迁移到 TiDB 时，因为 Oracle 支持全局索引，可能在某些表上存在一些不包含分区列的唯一索引，在迁移过程需要对表结构进行调整，以适应 TiDB 的分区表限制。然而，随着 TiDB 对全局索引的支持，用户只需简单地修改索引定义，将其设置为全局索引，即可与 Oracle 保持一致，从而显著降低迁移成本。

TiDB 全局索引的工作原理

基本思想

在 TiDB 的分区表中，本地索引的键值前缀是分区表的 ID 而全局索引的前缀是表的 ID。这样的改动确保了全局索引的数据在 TiKV 上分布是连续的，降低了查询索引时 RPC 的数量。

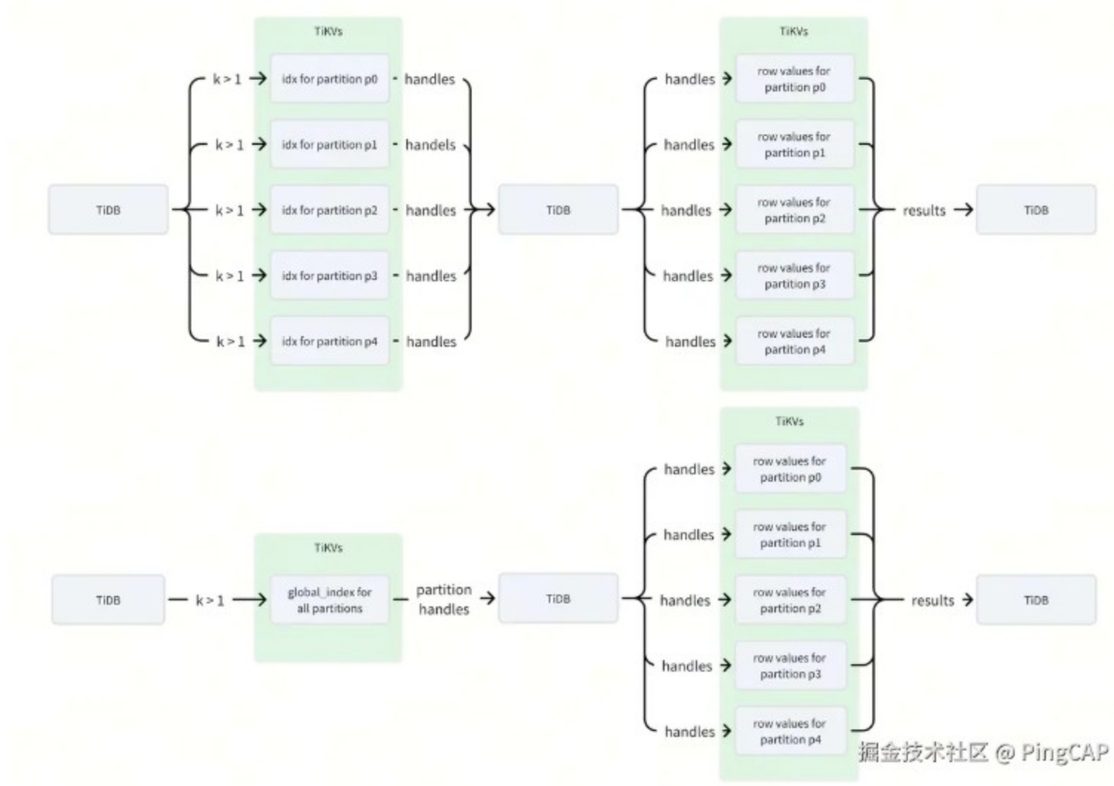
```
Plaintext
```

```
CREATE TABLE `sbtest` (
  `id` int(11) NOT NULL,
  `k` int(11) NOT NULL DEFAULT '0',
  `c` char(120) NOT NULL DEFAULT '',
  KEY idx(k),
  KEY global_idx(k) GLOBAL
) partition by hash(id) partitions 5;
```

以上面的表结构为例， idx 为普通索引， global_idx 为全局索引。索引 idx 的数据会分布在 5 个不同的 ranges 中，如 PartitionID1_i_xxx , PartitionID2_i_xxx 等，而索引 global_idx 的数据则会集中在一个 range (TableID_i_xxx) 内。

这样当我们进行 k 相关的查询时，如 select * from sbtest where k > 1 ，通过索引 idx 会构造 5 个不同的 ranges，而通过全局索引 global_idx 则只会构造 1 个 range，每个 range 在 TiDB 中对应一个或多个 RPC 请求，这样使用全局索引可以降低数倍的 RPC 请求数，从而提升查询索引的性能。

下图更加直观地展示了在使用 idx 和 global_idx 两个不同索引执行 select * from sbtest where k > 1 查询语句在 RPC 请求和数据流转过程中的差异。



编码方式

在 TiDB 中，索引项被编码为键值对。对于分区表，每个分区在 TiKV 层被视为一个独立的物理表，拥有自己的 partitionID 。因此，分区表的索引项也被编码为：

Plaintext 代码解读 复制代码

```
唯一键
Key:
- PartitionID_indexID_ColumnValues

Value:
- IntHandle
 - TailLen_IntHandle

- CommonHandle
 - TailLen_IndexVersion_CommonHandle

非唯一键
Key:
- PartitionID_indexID_ColumnValues_Handle

Value:
- IntHandle
 - TailLen_Padding

- CommonHandle
 - TailLen_IndexVersion
```

在全局索引中，索引项的编码方式有所不同。 为了使全局索引的键布局与当前索引键编码保持兼容，新的索引编码布局为：

Plaintext 代码解读 复制代码

```
唯一键
Key:
- TableID_indexID_ColumnValues

Value:
- IntHandle
 - TailLen_PartitionID_IntHandle

- CommonHandle
 - TailLen_IndexVersion_CommonHandle_PartitionID

非唯一键
Key:
- TableID_indexID_ColumnValues_Handle

Value:
- IntHandle
 - TailLen_PartitionID

- CommonHandle
 - TailLen_IndexVersion_PartitionID
```

这种编码方式使得全局索引的键以 TableID 开头，而 PartitionID 被放置在 Value 中。这样设计的优点是，它与现有的索引键编码方式兼容，但同时也带来了一些挑战，例如在执行 DROP PARTITION, TRUNCATE PARTITION 等 DDL 操作时，由于索引项不连续，需要进行额外的处理。

TiDB 全局索引的限制与注意事项

影响部分 DDL 性能

当分区表中存在全局索引时，执行诸如 DROP PARTITION（删除分区）、TRUNCATE PARTITION（清空分区）、REORG PARTITION（重组分区）等部分 DDL 操作时，需要同步更新全局索引的值，这会显著增加 DDL 操作的执行时间。

在 v8.5.0 默认参数下，测试显示对包含全局索引的 sysbench 表执行 DROP PARTITION 或 TRUNCATE PARTITION 操作时， oltp_read_write 负载的性能会下降 15% 至 20%。

聚簇索引(Clustered Index)

聚簇索引不能成为全局索引，是因为如果聚簇索引是全局索引，则表将不再分区。这是因为聚簇索引的键是分区级别的行数据的键，但全局索引是表级别的，这就造成了冲突。如果需要将主键设置为全局索引，则需要显式设置该主键为非聚簇索引，如 PRIMARY KEY(col1, col2) NONCLUSTERED GLOBAL 。

性能测试数据

- select_random_points in sysbench

示例表结构

```plaintext
CREATE TABLE `sbtest` (
  `id` int(11) NOT NULL,
  `k` int(11) NOT NULL DEFAULT '0',
  `c` char(120) NOT NULL DEFAULT '',
  `pad` char(60) NOT NULL DEFAULT '',
  PRIMARY KEY (`id`) /*T![clustered_index] CLUSTERED */,
  KEY `k_1` (`k`)
  /* Key `k_1` (`k`, `c`) GLOBAL */
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
/* Partition by hash(`id`) partitions 100 */
/* Partition by range(`id`) xxxx */
```

负载 SQL

```plaintext
SELECT id, k, c, pad
FROM sbtest1
WHERE k IN (xx, xx, xx)
```

Range Partition (100 partitions)				
Concurrency	1	32	64	Average RU
Clustered non-partitioned table	225	19,999	30,293	7.92
Clustered table range partitioned by PK	68	480	511	114.87
Clustered table range partitioned by PK, with Global Index on `k, c` columns	207	17,798	27,707	11.73
Hash Partition (100 partitions)				
Concurrency	1	32	64	Average RU
Clustered non-partitioned table	166	20361	28922	7.86
Clustered table hash partitioned by PK	60	244	283	119.73
Clustered table hash partitioned by PK, with Global Index on `k, c` columns	156	18233	15581	10.77

- 通过上述测试可以看出，在高并发环境下，全局索引能够显著提升分区表查询性能，提升幅度可达 50 倍。同时，全局索引还能够显著降低资源（RU）消耗。随着分区数量的增加，这种性能提升的效果将愈加明显。

最佳实践

全局索引和本地索引

全局索引适用场景：

- **数据归档不频繁：** 例如，医疗行业的部分业务数据需要保存 30 年，通常按月分区，然后一次性创建 360 个分区，且很少进行 DROP 或 TRUNCATE 操作。在这种情况下，使用全局索引更为合适，因为它能提供跨分区的一致性和查询性能。
- **查询需要跨分区的数据：** 当查询需要访问多个分区的数据时，全局索引可以避免跨分区扫描，提高查询效率。

本地索引适用场景：

- **数据归档需求：** 如果数据归档操作很频繁，且主要查询集中在单个分区内，本地索引可以提供更好的性能。
- **需要使用分区交换功能：** 在银行等行业，可能会将处理后的数据先写入普通表，确认无误后再交换到分区表，以减少对分区表性能的影响。此时，本地索引更为适用，因为在使用了全局索引之后，分区表将不再支持分区交换功能。

全局索引和聚簇索引

由于聚簇索引和全局索引的原理限制，一个索引不能同时作为聚簇索引和全局索引。然而，这两种索引在不同查询场景中能提供不同的性能优化。在遇到需要同时兼顾两者的需求时，我们可以将分区列添加到聚簇索引中，同时创建一个不包含分区列的全局索引。

假设我们有如下表结构：

Plaintext 代码解读 复制代码

```
CREATE TABLE `t` (
  `id` int DEFAULT NULL,
  `ts` timestamp NULL DEFAULT NULL,
  `data` varchar(100) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
PARTITION BY RANGE (UNIX_TIMESTAMP(`ts`))
(PARTITION `p0` VALUES LESS THAN (1735660800)
 PARTITION `p1` VALUES LESS THAN (1738339200)
 ...)
```

在上面的 t 表中， id 列的值是唯一的。为了优化点查和范围查询的性能，我们可以选择在建表语句中定义一个聚簇索引 PRIMARY KEY(id, ts) 和一个不包含分区列的全局索引 UNIQUE KEY id(id) 。这样在进行基于 id 的点查询时，会走全局索引 id ，选择 PointGet 的执行计划；而在进行范围查询时，聚簇索引则会被选中，因为聚簇索引相比全局索引少了一次回表操作，从而提升查询效率。

修改后的表结构如下所示：

Plaintext 代码解读 复制代码

```
CREATE TABLE `t` (
  `id` int NOT NULL,
  `ts` timestamp NOT NULL,
  `data` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`id`, `ts`) /*T![clustered_index] CLUSTERED */,
  UNIQUE KEY `id` (`id`) /*T![global_index] GLOBAL */
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
PARTITION BY RANGE (UNIX_TIMESTAMP(`ts`))
(PARTITION `p0` VALUES LESS THAN (1735660800),
 PARTITION `p1` VALUES LESS THAN (1738339200) ...)
```

通过这种方式，我们既能优化基于 id 的点查询，又能提升范围查询的性能，同时确保表的分区列在基于时间戳的查询中能得到有效的利用。

总结

TiDB 全局索引是 TiDB 在分区表索引方面的重要特性，它通过允许索引分区与表分区之间提供一对多的映射关系，提供了更灵活的索引设计和更高效的查询性能。全局索引的引入，不仅提升了 TiDB 分区表在处理复杂查询和大数据量场景下的能力，还为用户在数据库设计和优化方面提供了更多的选择。

然而，全局索引也带来了一些挑战，如维护成本的增加。在使用全局索引时，需要根据具体的业务需求和数据特点，合理设计索引，权衡查询性能和数据修改性能，以达到最佳的数据库性能。

总之，TiDB 全局索引是一个强大且灵活的特性，能够帮助用户更好地优化数据库性能，满足多样化的业务需求。在实际应用中，合理使用全局索引，可以显著提升查询性能，提高数据库的整体效率。

标签： TiDB 数据库

评论 0

登录 / 注册 即可发布评论!

暂无评论数据

为你推荐

三种MySQL大表优化方案，过目不忘！
Java小叮当 | 4年前 | 283 | 2 | 评论 Java

狂飙 50 倍 | TiDB DDL 框架优化深度解析
PingCAP | 1月前 | 51 | 1 | 评论 数据库

TiDB 3.0 GA，稳定性和性能大幅提升
PingCAP | 5年前 | 555 | 1 | 评论 TiDB

TiDB 5.1 发版，打造更流畅的企业级数据库体验
PingCAP | 3年前 | 357 | 3 | 评论 数据库 后端

数据库性能优化，究竟该如何下手?
HeapDump性能社区 | 3年前 | 552 | 点赞 | 评论 数据库

记录一次MySQL两千万数据的大表优化解决过程，提供3种解决方案
编程学习网 | 4年前 | 348 | 点赞 | 评论 MySQL

MySQL“被动”性能优化汇总！
Java中文社群 | 4年前 | 1.6k | 12 | 2 Java MySQL

维护不了就跑路，然后我被坑了！记一次采坑优化记录
codeGoogle | 4年前 | 2.4k | 28 | 4 Java MySQL

我把慢sql优化后性能提升30倍！
临时工 | 1年前 | 1.1k | 3 | 评论 MySQL 运维

亿级表优化「TIDB 分区篇」，值得收藏
我是Allen | 10月前 | 4.3k | 10 | 1 后端 数据库 架构

MySQL两千万数据大表优化过程，三种解决方案！
编程学习网 | 4年前 | 371 | 点赞 | 评论 MySQL

MySQL单表数据量过千万，采坑优化记录，完美解决方案
Redick01 | 3年前 | 357 | 点赞 | 评论 数据库

TiDB 的列式存储引擎是如何实现的?
PingCAP | 4年前 | 1.1k | 4 | 评论 数据库

蔚来汽车 x TiDB | 单表超 20 亿条数据，从 MySQL 到 TiDB 的迁移思考与实践
PingCAP | 6月前 | 541 | 3 | 1 数据库 MySQL TiDB

三种解决方案 优化MySQL两千万数据大表
阿布阿布 | 4年前 | 1.4k | 7 | 评论 MySQL