MySQL中varchar(50)和varchar(500)区别是什么?

捡田螺的小男孩 2025年02月25日 09:02 广东

一. 问题描述

我们在设计表结构的时候，设计规范里面有一条如下规则:

- 对于可变长度的字段，在满足条件的前提下，尽可能使用较短的变长字段长度。

为什么这么规定？我在网上查了一下，主要基于两个方面

- 基于存储空间的考虑
- 基于性能的考虑

网上说 `Varchar(50)` 和 `varchar(500)` 存储空间上是一样的,真的是这样吗?

基于性能考虑,是因为过长的字段会影响到查询性能?

本文我将带着这两个问题探讨验证一下

二.验证存储空间区别

1.准备两张表

```
CREATE TABLE `category_info_varchar_50` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT '主键',
  `name` varchar(50) NOT NULL COMMENT '分类名称',
  `is_show` tinyint(4) NOT NULL DEFAULT '0' COMMENT '是否展示：0 禁用，1启用',
  `sort` int(11) NOT NULL DEFAULT '0' COMMENT '序号',
  `deleted` tinyint(1) DEFAULT '0' COMMENT '是否删除',
  `create_time` datetime NOT NULL COMMENT '创建时间',
  `update_time` datetime NOT NULL COMMENT '更新时间',
  PRIMARY KEY (`id`) USING BTREE,
  KEY `idx_name` (`name`) USING BTREE COMMENT '名称索引'
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='分类';

CREATE TABLE `category_info_varchar_500` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT '主键',
  `name` varchar(500) NOT NULL COMMENT '分类名称',
  `is_show` tinyint(4) NOT NULL DEFAULT '0' COMMENT '是否展示：0 禁用，1启用',
```

```
  `sort` int(11) NOT NULL DEFAULT '0' COMMENT '序号',
  `deleted` tinyint(1) DEFAULT '0' COMMENT '是否删除',
  `create_time` datetime NOT NULL COMMENT '创建时间',
  `update_time` datetime NOT NULL COMMENT '更新时间',
  PRIMARY KEY (`id`) USING BTREE,
  KEY `idx_name` (`name`) USING BTREE COMMENT '名称索引'
) ENGINE=InnoDB AUTO_INCREMENT=288135 DEFAULT CHARSET=utf8mb4 COMMENT='分类';
```

2.准备数据

给每张表插入相同的数据,为了凸显不同,插入100万条数据

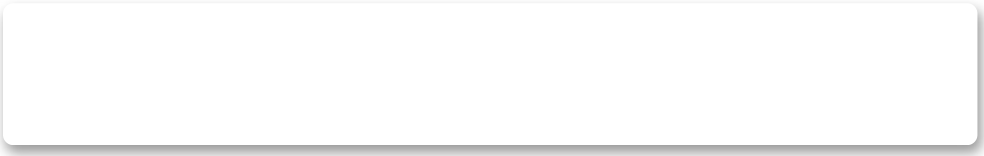
```
DELIMITER $$
CREATE PROCEDURE batchInsertData(IN total INT)
BEGIN
    DECLARE start_idx INT DEFAULT 1;
    DECLARE end_idx INT;
    DECLARE batch_size INT DEFAULT 500;
    DECLARE insert_values TEXT;

    SET end_idx = LEAST(total, start_idx + batch_size - 1);

    WHILE start_idx <= total DO
        SET insert_values = '';
        WHILE start_idx <= end_idx DO
            SET insert_values = CONCAT(insert_values, CONCAT('(\'name', start_i
            SET start_idx = start_idx + 1;
        END WHILE;
        SET insert_values = LEFT(insert_values, LENGTH(insert_values) - 1); --
        SET @sql = CONCAT('INSERT INTO category_info_varchar_50 (name, is_show,

        PREPARE stmt FROM @sql;
        EXECUTE stmt;
       SET @sql = CONCAT('INSERT INTO category_info_varchar_500 (name, is_show,
       PREPARE stmt FROM @sql;
        EXECUTE stmt;

        SET end_idx = LEAST(total, start_idx + batch_size - 1);
    END WHILE;
END$$
DELIMITER ;

CALL batchInsertData(1000000);
```

3.验证存储空间

查询第一张表SQL

```
SELECT
    table_schema AS "数据库",
    table_name AS "表名",
    table_rows AS "记录数",
    TRUNCATE ( data_length / 1024 / 1024, 2 )  AS "数据容量 (MB) ",
    TRUNCATE ( index_length / 1024 / 1024, 2 )  AS "索引容量 (MB) "
FROM
    information_schema.TABLES
WHERE
    table_schema = 'test_mysql_field'
  and TABLE_NAME = 'category_info_varchar_50'
ORDER BY
    data_length DESC,
    index_length DESC;
```

查询结果



查询第二张表SQL

```
SELECT
    table_schema AS "数据库",
    table_name AS "表名",
    table_rows AS "记录数",
    TRUNCATE ( data_length / 1024 / 1024, 2 )  AS "数据容量 (MB) ",
    TRUNCATE ( index_length / 1024 / 1024, 2 )  AS "索引容量 (MB) "
FROM
    information_schema.TABLES
WHERE
    table_schema = 'test_mysql_field'
  and TABLE_NAME = 'category_info_varchar_500'
ORDER BY
    data_length DESC,
    index_length DESC;
```

查询结果

4.结论

两张表在占用空间上确实是一样的,并无差别

三.验证性能区别

1.验证索引覆盖查询

```
select name from category_info_varchar_50 where name = 'name100000'
-- 耗时0.012s
select name from category_info_varchar_500 where name = 'name100000'
-- 耗时0.012s
select name from category_info_varchar_50 order by name;
-- 耗时0.370s
select name from category_info_varchar_500 order by name;
-- 耗时0.379s
```

通过索引覆盖查询性能差别不大

1.验证索引查询

```
select * from category_info_varchar_50 where name = 'name100000'
--耗时 0.012s
select * from category_info_varchar_500 where name = 'name100000'
--耗时 0.012s
select * from category_info_varchar_50 where name in('name100','name1000','name
'name200','name2000','name200000','name20000','name2200000','name300','name3000
'name400','name4000','name400000','name40000','name4400000','name500','name5000
'name600','name6000','name600000','name60000','name6600000','name700','name7000
'name8000','name800000','name80000','name6600000','name900','name9000','name900
-- 耗时 0.011s -0.014s
-- 增加 order by name 耗时 0.012s - 0.015s

select * from category_info_varchar_50 where name in('name100','name1000','name
'name200','name2000','name200000','name20000','name2200000','name300','name3000
```

```
'name400','name4000','name400000','name40000','name4400000','name500','name5000
'name600','name6000','name600000','name60000','name6600000','name700','name7000
'name8000','name800000','name80000','name6600000','name900','name9000','name900
-- 耗时  0.012s -0.014s
-- 增加 order by name 耗时 0.014s - 0.017s
```

索引范围查询性能基本相同，增加了order By后开始有一定性能差别;

3.验证全表查询和排序

全表无排序

全表有排序

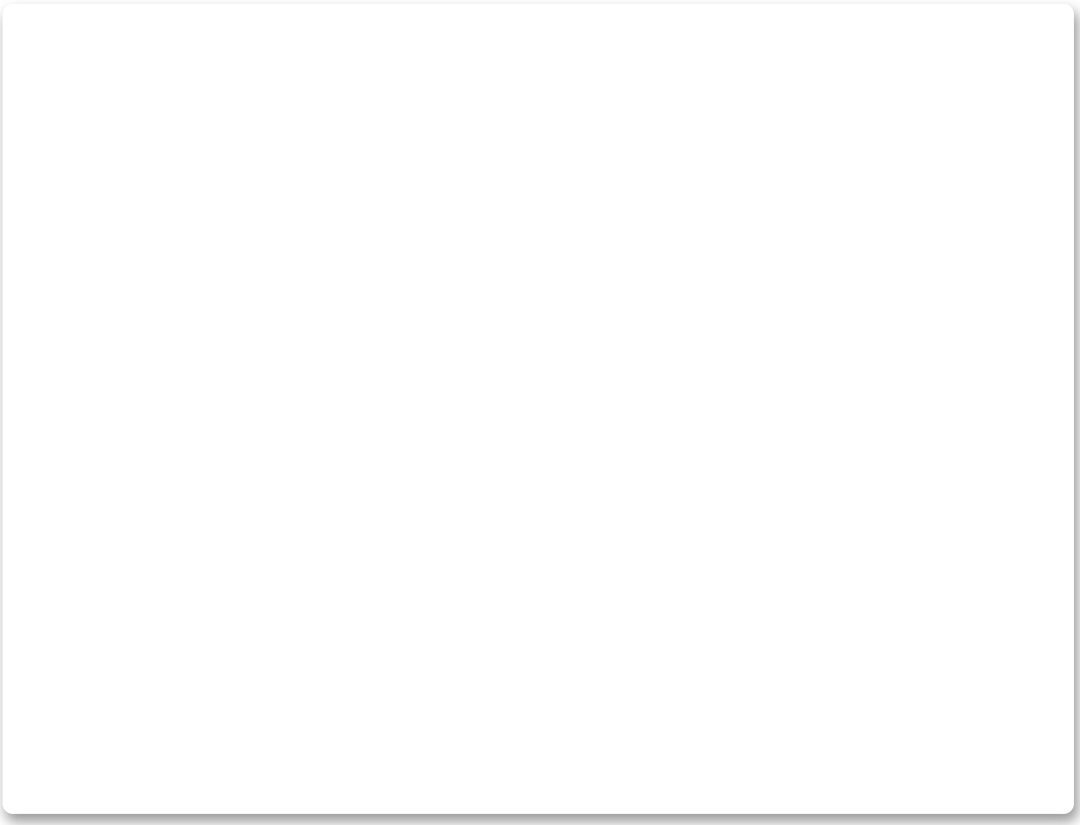
```
select * from category_info_varchar_50 order by  name ;
--耗时 1.498s
select * from category_info_varchar_500 order by  name  ;
--耗时 4.875s
```

结论:

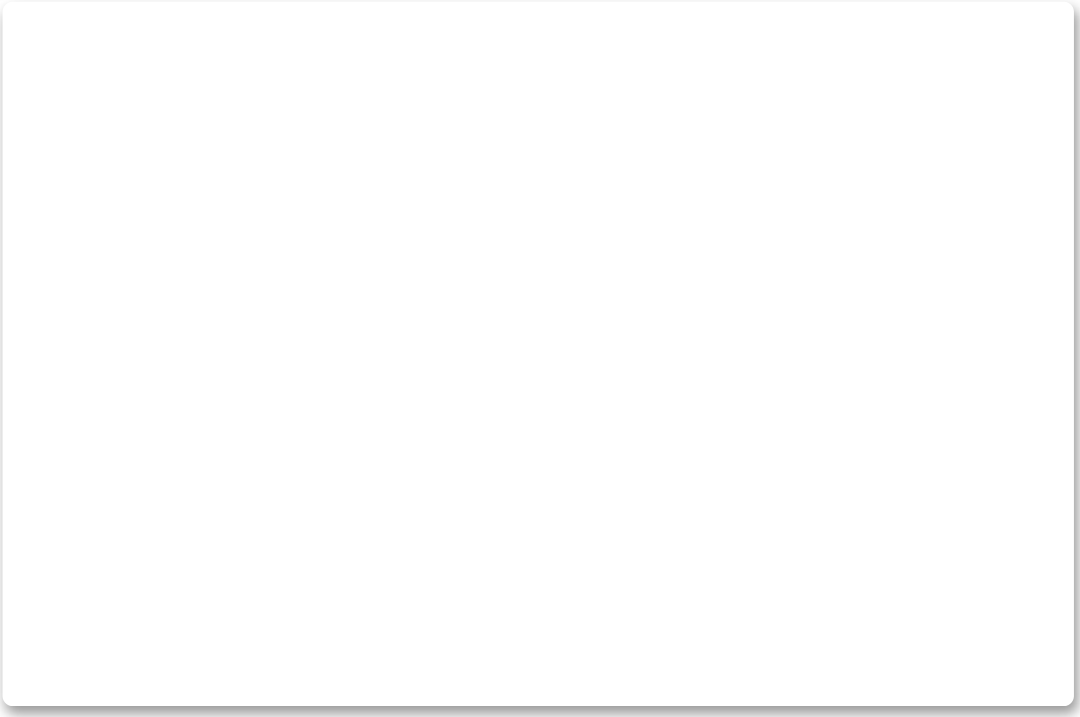
全表扫描无排序情况下,两者性能无差异,在全表有排序的情况下，两种性能差异巨大;

分析原因

varchar50 全表执行sql分析



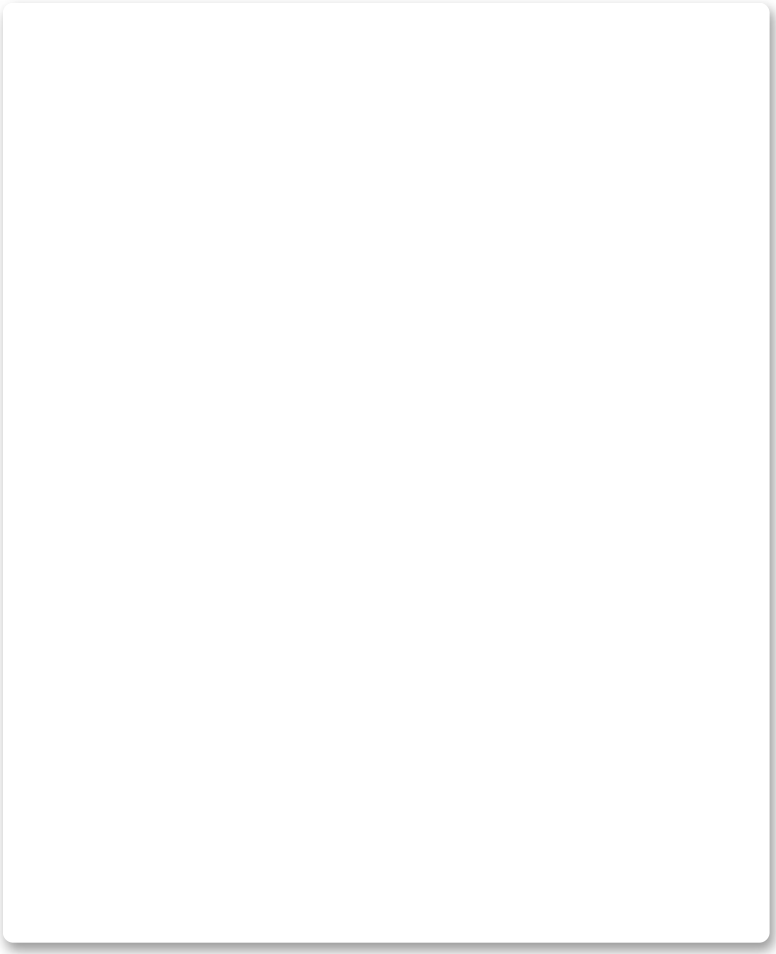
我发现86%的时花在数据传输上,接下来我们看状态部分,关注Created_tmp_files和sort_merge_passes



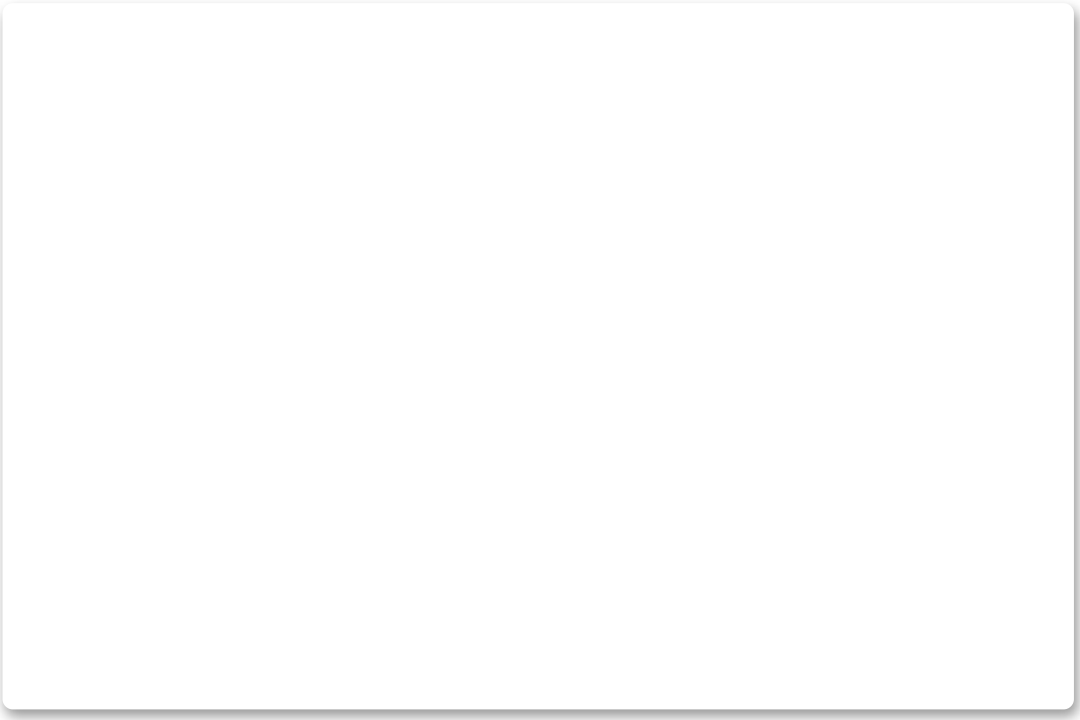
Created_tmp_files为3

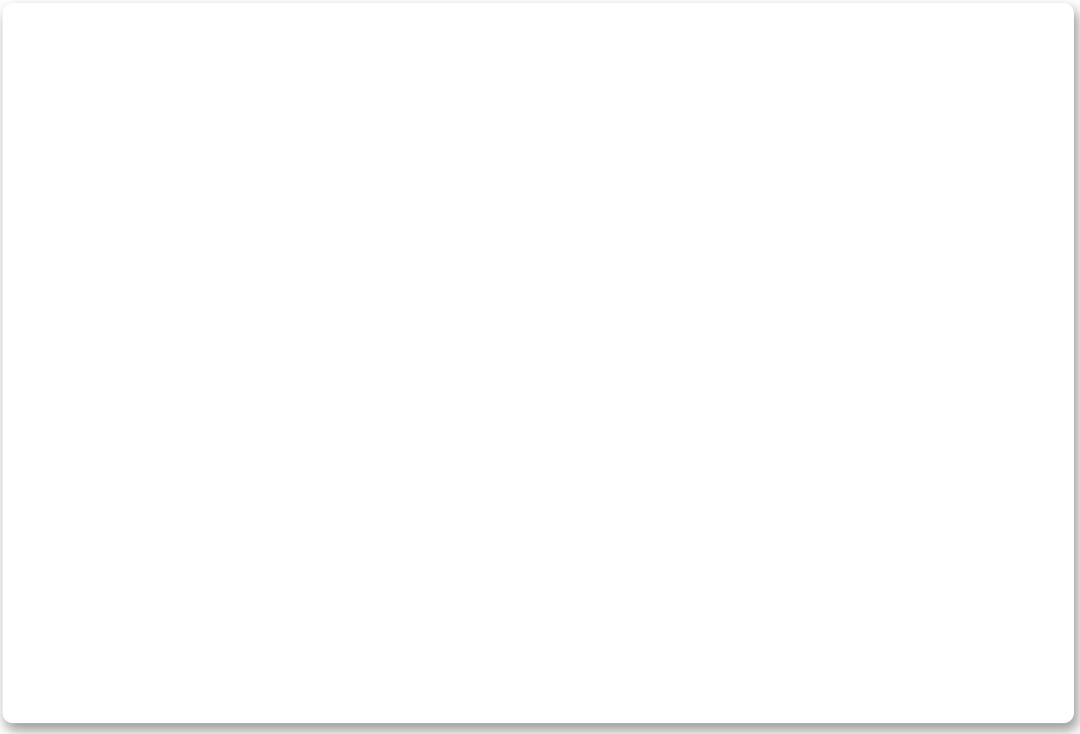
sort_merge_passes为95

varchar500 全表执行sql分析



增加了临时表排序





Created_tmp_files 为 4

sort_merge_passes为645

关于sort_merge_passes，Mysql给出了如下描述:

Number of merge passes that the sort algorithm has had to do. If this value is large, you may want to increase the value of the sort_buffer_size.

其实sort_merge_passes对应的就是MySQL做归并排序的次数，也就是说，如果sort_merge_passes值比较大，说明sort_buffer和要排序的数据差距越大，我们可以通过增大sort_buffer_size或者让填入sort_buffer_size的键值对更小来缓解sort_merge_passes归并排序的次数。

四.最终结论

至此,我们不难发现,当我们最该字段进行排序操作的时候,Mysql会根据该字段的设计的长度进行内存预估，如果设计过大的可变长度，会导致内存预估的值超出sort_buffer_size的大小，导致mysql采用磁盘临时文件排序,最终影响查询性能;

> 来源：juejin.cn/post/7350228838151847976

捡田螺的小男孩
专注后端技术栈
266篇原创内容

公众号