

1.储存引擎

InnoDB和MyISAM

2.索引

什么是索引吗?

2.1 索引的数据结构

哈希索引

BTree索引

非聚簇索引

聚簇索引

2.2 索引类型

2.3 索引失效的场景

3. 事务

3.1 事务的4大特性ACID

3.2 并发事务带来的问题

不可重复读和幻读

3.3 隔离级别

4.内外连接

SQL

5.其他杂项

1.数据库三大范式

2.drop,delete和turncate区别

3.慢查询

Redis

性能好的原因

数据类型

String类型

Hash类型

List类型

Set类型

Sorted Set类型

redis持久化

1.储存引擎

InnoDB和MyISAM

MyISAM	InnoDB	
文件格式	数据和索引是分别存储的， 数据.MYD，索引.MYI	数据和索引是集中存储的， .ibd
文件能否移动	能，一张表就对应.frm、 MYD、MYI3个文件	否，因为关联的还有data 下的其它文件
记录存储顺序	按记录插入顺序保存	按主键大小有序插入
空间碎片（删除记录并flush table 表名 之后，表文件大小不变）	产生。定时整理：使用命令 optimize table 表名实现	不产生
事务	不支持	支持
外键	不支持	支持
锁支持（锁是避免资源争用的一个机制， MySQL锁对用户几乎是透明的）	表级锁定	行级锁定、表级锁定，锁 定力度小并发能力高

1.2 “Where” “Having” 区别

“Where” 是一个约束声明，使用Where来约束来之数据库的数据，Where是在结果返回之前起作用的，且Where中不能使用聚合函数。

“Having”是一个过滤声明，是在查询返回结果集以后对查询结果进行的过滤操作，在Having中可以使用聚合函数。

1.3 聚合函数的查询

max\min\sum\avg\count

2.索引

<https://www.cnblogs.com/williamjie/p/11187470.html>

什么是索引吗？

索引其实是一种数据结构，能够帮助我们快速的检索数据库中的数据

2.1 索引的数据结构

BTree索引和哈希索引

哈希索引

哈希索引只适用于=,不能用范围查询

哈希索引的查询性能最佳,用于绝大多数需求为单条查询的时候

BTree索引

btree（多路平衡查找树）是一种广泛应用于磁盘上实现索引功能的一种数据结构，也是大多数数据库索引表的实现。

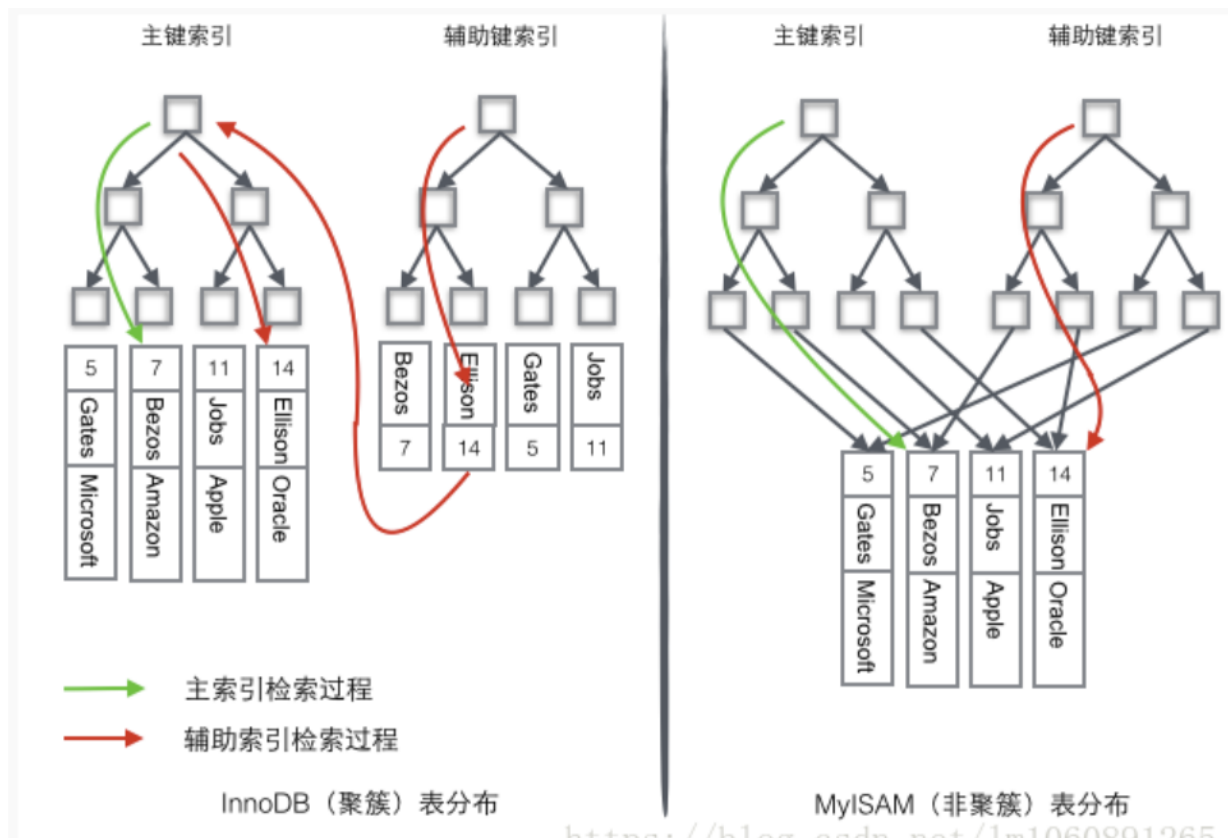
BTree的一个node可以存储多个关键字，node的大小取决于计算机的文件系统，因此我们可以通过减小索引字段的长度使结点存储更多的关键字。

非聚簇索引

叶节点存放数据的地址

聚簇索引

主键索引采用聚簇索引,节点保存整行数据



2.2 索引类型

普通索引 (key)，唯一索引 (unique key)，主键索引 (primary key) 复合索引 三种索引的索引方式是一样的，只不过对索引的关键字有不同的限制：

- 普通索引：对关键字没有限制
- 唯一索引：要求记录提供的关键字不能重复
- 主键索引：要求关键字唯一且不为null

最左前缀匹配原则

在mysql建立联合索引时会遵循最左前缀匹配的原则，即最左优先，在检索数据时从联合索引的最左边开始匹配，示例：

对列col1、列col2和列col3建一个联合索引

```
KEY test_col1_col2_col3 on test(col1,col2,col3);
```

联合索引 test_col1_col2_col3 实际建立了(col1)、(col1,col2)、(col,col2,col3)三个索引。

```
SELECT * FROM test WHERE col1="1" AND col2="2" AND col4="4"
```

上面这个查询语句执行时会依照最左前缀匹配原则，检索时会使用索引(col1,col2)进行数据匹配。

2.3 索引失效的场景

1. 字段不独立

第一个使用索引，第二个不使用

```
select * from user where id = 20-1;  
select * from user where id+1 = 20;
```

2. like 通配符开头

不使用索引

```
select * from article where title like '%mysql%';
```

使用索引

```
select * from article where title like 'mysql%';
```

3. 复合索引的第二个字段

2.4 索引的缺点

1. 索引文件需要占用额外的磁盘空间

2. 写操作会变慢, 需要额外操作

3. 事务

3.1 事务的4大特性ACID

1. 原子性: 事务是最小的执行单元, 不允许分割. 要么全部完成, 要么完全不起作用
2. 一致性: 执行事务前后, 数据保持一致, 多个事务对同一个数据读取的结果是相同的
3. 隔离性: 并发访问数据库, 一个用户不被其他事务所干扰, 各并发事务之间数据库是独立的
4. 持久性: 一个事务被提交后, 数据库的改变是持久的

3.2 并发事务带来的问题

脏读: 读取到未提交的数据

更新修改: 最后的更新覆盖了其他事务之前的更新, 而事务之间并不知道, 发生更新丢失。更新丢失, 可以完全避免, 应用对访问的数据加锁即可

不可重复读: 在一个事务内多次读同一个数据, 在事务中, 另一个事务访问并修改数据. 第一个事务前后读到的数据可能不一样, 因此称为不可重复读.

幻读: 和不可重复读类似, 但是有着明显区别. 在一个事务(T1)的过程中, 另一个事务(T2)插入了几行数据. T1在随后的查询中发现了原来根本不存在的记录, 称之为幻读.

不可重复读和幻读

不可重复读的重点是前后读同一个值发生变化

幻读是前后读取时插入或者删除了新的数据

3.3 隔离级别

读未提交

读已提交

可重复读

串行化

4.内外连接

4.1 内连接-inner join

两个数据库的交集部分

4.2 外连接

01. 左外连接 left join

left join 是left outer join的简写，它的全称是左外连接，是外连接中的一种。

左(外)连接，左表(a_table)的记录将会全部表示出来，而右表(b_table)只会显示符合搜索条件的记录。右表记录不足的地方均为NULL。

02. 右外连接 right join

right join是right outer join的简写，它的全称是右外连接，是外连接中的一种。

左表(a_table)只会显示符合搜索条件的记录，而右表(b_table)的记录将会全部表示出来。左表记录不足的地方均为NULL。

*https://blog.csdn.net/weixin_43858201/article/details/90707592

SQL

crud

INSERT INTO table_name (列1, 列2,...) VALUES (值1, 值2,...)

select * from table_name where

UPDATE 表名称 SET 列名称 = 新值 WHERE 列名称 = 某值

DELETE FROM 表名称 WHERE 列名称 = 值

多表查询

内连接

方言版: select * from 表1 , 表2 where 表1.XX=表2.XX;

标准版: select * from 表1 inner join 表2 where 表1.XX=表2.XX;

自然连接: select * from 表1 natural join 表2;

外连接

外连接有一主一次,当主表中所有的记录无论满足不满足条件, 都打印出来。

左外连接: select * from 表1 left join 表2 on 表1.xx=表2.xx;

右外连接: select * from 表1 right join 表2 on 表1.xx=表2.xx;

Alter修改表

```
ALTER TABLE table_name ADD column_name datatype;  
alter table user_index add UNIQUE KEY (id_card);
```

建表

CREATE TABLE 表名称

```
(  
列名称1 数据类型,  
列名称2 数据类型,  
列名称3 数据类型,  
....  
)
```

顺逆序排列

mySQL里desc和asc的意思

desc是descend 降序意思

asc 是ascend 升序意思

sql = "select 表内容名 from 数据库表名 Putout=true order by 读取的排序表名 asc"

例如:

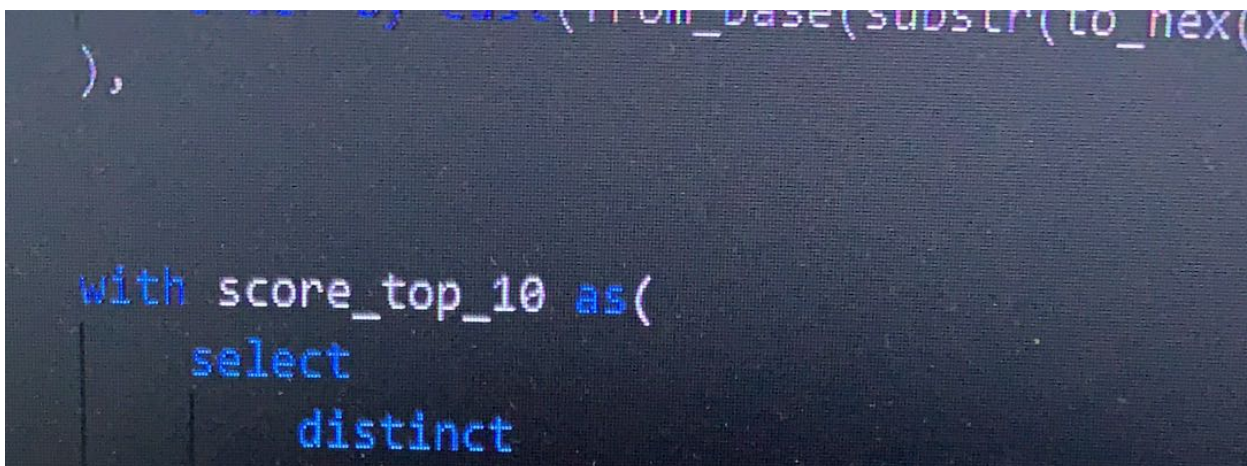
```
select * from user where Putout=true order by time desc //按最新时间来排序  
select * from user where Putout=true order by time asc //按早时间来排序
```

查看表结构

DESC 表明;

limit

select * from user limit 5,10; 记录6-15 前一个是偏移位置,后一个是数量



```
... substr(to_next  
) ,  
  
with score_top_10 as(  
    select  
        distinct
```



```

        score
    from
        student_score
    order by
        score desc
    limit 10
)
select
    student_score.student,
    student_score.score
from
    student_score
join
    score_top_10
on
    student_score.score = score_top_10.score
order by
    score desc; I

```

having和group by

having 子句的作用是筛选满足条件的组，即在分组之后过滤数据，条件中经常包含聚合函数，使用having 条件显示特定的组，也可以使用多个分组标准进行分组。 与 WHERE 和

SELECT 的交互方式类似。WHERE 搜索条件在进行分组操作之前应用；而 HAVING 搜索条件在进行分组操作之后应用。

三表查询

employee_id	company_id
1	101
2	101
3	102

company_id	company_name
101	A
102	B

employee_id	employee_name	employee_age
1	李云	35
2	张飞	23
3	高丽	25

```
SELECT employee_name
FROM (table3 c LEFT JOIN TABLE1 a
ON c.employee_id=a.employee_id )
LEFT JOIN table2 b ON b.company_id=a.company_id
WHERE company_name ='A' and employee_age<30;
```

5.其他杂项

1.数据库三大范式

- 第一范式(确保每列保持原子性)
- 第二范式(确保表中的每列都和主键相关)
- 第三范式(确保每列都和主键列直接相关,而不是间接相关)
- BCNF-第三范式的加强版

2.drop,delete和turncate区别

drop是删除整个表,包含表的数据和表的结构

turncate是删除表的所有数据,保留表的结构,相当于重新初始化表

delete 删除表的某行数据

执行速度 drop>turncate>delete

3.慢查询

三、设置步骤

1.查看慢查询相关参数

```
mysql> show variables like 'slow_query%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| slow_query_log | OFF   |
| slow_query_log_file | /mysql/data/localhost-slow.log |
+-----+-----+

mysql> show variables like 'long_query_time';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| long_query_time | 10.000000 |
+-----+-----+
```

2.设置方法

方法一：全局变量设置

将 slow_query_log 全局变量设置为“ON”状态

```
mysql> set global slow_query_log='ON';
```

设置慢查询日志存放的位置

```
mysql> set global slow_query_log_file='/usr/local/mysql/data/slow.log';
```

查询超过1秒就记录

```
mysql> set global long_query_time=1;
```

方法二：配置文件设置

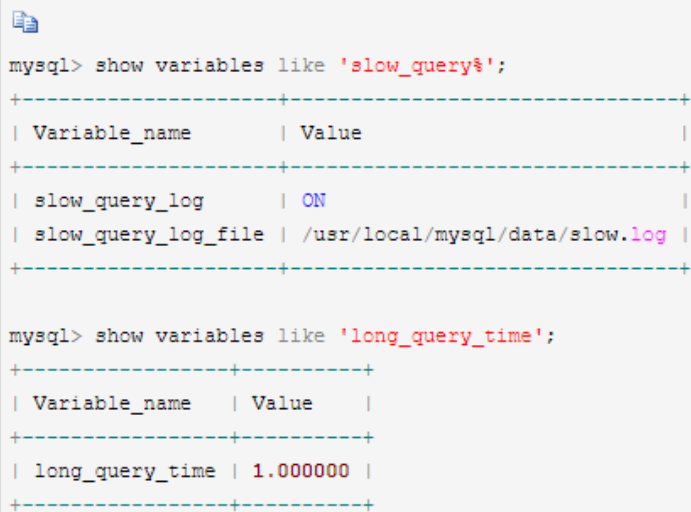
修改配置文件my.cnf，在[mysqld]下的下方加入

```
[mysqld]
slow_query_log = ON
slow_query_log_file = /usr/local/mysql/data/slow.log
long_query_time = 1
```

3.重启MySQL服务

```
service mysqld restart
```

4.查看设置后的参数



The terminal screenshot shows two MySQL commands and their outputs. The first command, 'show variables like 'slow_query%';', displays a table with two rows: 'slow_query_log' set to 'ON' and 'slow_query_log_file' set to '/usr/local/mysql/data/slow.log'. The second command, 'show variables like 'long_query_time';', displays a table with one row: 'long_query_time' set to '1.000000'. Both tables have columns 'Variable_name' and 'Value'.

```
mysql> show variables like 'slow_query%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| slow_query_log | ON    |
| slow_query_log_file | /usr/local/mysql/data/slow.log |
+-----+-----+

mysql> show variables like 'long_query_time';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| long_query_time | 1.000000 |
+-----+-----+
```

select greatest和select least

<https://www.yiibai.com/mysql/greatest-least.html>

MySQL GREATEST和LEAST函数介绍

`GREATEST` 和 `LEAST` 函数都使用 `N` 个参数，并分别返回最大和最小值。下面说明 `GREATEST` 和 `LEAST` 函数的语法：

```
GREATEST(value1, value2, ...);  
LEAST(value1,value2,...);
```

参数可能具有混合数据类型。以下比较规则适用于这两个函数：

- 如果任何参数为 `NULL`，则两个函数都将立即返回 `NULL`，而不进行任何比较。
- 如果在 `INT` 或 `REAL` 上下文中使用函数，或者所有参数都是整数值或 `REAL` 值，那么它们将分别作为 `INT` 和 `REAL` 来比较。
- 如果参数由数字和字符串组成，则函数将它们作为数字进行比较。
- 如果至少一个参数是非二进制(字符)字符串，则函数将参数作为非二进制字符串进行比较。
- 在所有其他情况下，函数将参数作为二进制字符串进行比较

以下示例演示了 `GREATEST` 和 `LEAST` 函数的工作原理。

```
SELECT GREATEST(10, 20, 30), -- 30  
       LEAST(10, 20, 30); -- 10  
  
SELECT GREATEST(10, null, 30), -- null  
       LEAST(10, null, 30); -- null
```

Redis

性能好的原因

1. 单进程单线程
2. 基于内存

数据类型

String类型

```
set key value  
get key
```

Hash类型

```
hset key key value  
hget key key  
hgetall key
```

List类型

双向链表

lpush

rpush

lpop

rpop

lrange

Set类型

自动排重

sadd

spop

Sorted Set类型

zadd

zrange

zrem

redis持久化

1. redis是一个内存数据库，当redis服务器重启，获取电脑重启，数据会丢失，我们可以将redis内存中的数据持久化保存到硬盘的文件中。

2. redis持久化机制：

a. RDB：默认方式，不需要进行配置，默认就使用这种机制

- 在一定的间隔时间中，检测key的变化情况，然后持久化数据

1. 编辑redis.windows.conf文件

```
#after 900 sec (15 min) if at least 1 key changed
save 900 1
#after 300 sec (5 min) if at least 10 keys changed
save 300 10
#after 60 sec if at least 10000 keys changed
save 60 10000
```

2. 重新启动redis服务器，并指定配置文件名称

D:\JavaWeb2018\day23_redis\资料\redis\windows-64\redis-

2.8.9>redis-server.exe redis.windows.conf

b. AOF：日志记录的方式，可以记录每一条命令的操作。可以每一次命令操作后，持久化数据

■ 编辑redis.windows.conf文件

appendonly no (关闭aof) --> appendonly yes (开启aof)

appendfsync always : 每一次操作都进行持久化

appendfsync everysec : 每隔一秒进行一次持久化

appendfsync no : 不进行持久化