MySQL优化及索引

##

1.配置

1.1 mysql核心目录

- Linux环境下：
  - `/var/lib/mysql`： mysql安装目录
  - `/usr/share/mysql`:配置文件
  - `/etc/init.d/mysql`:mysql启动/停止脚本

1.2 mysql字符编码

- 查看字符编码：`show variables like '%char%'`
- 清屏： `ctrl + L`

1.2.1修改字符编码

```
vim /etc/my.xnf

#添加以下内容
[mysql]
default-character-set = utf8
[client]
default-character-set = utf8

[mysqld]
character_set_server = utf8
character_set_client = utf8
collation_server = utf8_general_ci

#重启mysql
service mysql restart

#注意事项：修改编码，只对之后创建的数据库生效
```

2.原理

2.1 逻辑分层

1.连接层

2.服务层

3.引擎层

4.存储层

2.2 存储引擎

- 查看支持引擎：`show engines`
- 查看默认引擎：`show variables like '%storage_engine%'`
- 指定数据库对象的引擎:

```
use #{dbname}
create table {tablename}{
#...
} engine = MyISAM;
```

2.2.1InnoDB

- 事务优先（适合高并发操作；行锁）

2.2.2 MyISAM

- 性能优先（表锁）

2.2.3 区别

3.SQL优化前准备

3.1 优化原因

1. 性能低
2. 等待时间长
3. SQL语句欠佳（连接查询）
    - 编写过程
        - `select dins ... from ...join ..on ..where ..group by ..having...ordeer by ..limit..`
    - 编译过程
        - `from .. on .. join ..where ..group by .. having...select .. order by .. limit..`
    - SQL解析顺序帮助文档
4. 索引失效
5. 服务器参数设置不合理

3.2 优化索引

3.2.1 概述

- 索引：
  - 相当于书的目录
  - index是帮助MYSQL高效获取数据的数据结构
  - 数据结构（树：B树（mysql默认），Hash树...）
- B树索引：
  - 小的放左，大的放右
- 索引的弊端：
  1. 索引本身很大，可以存放在内存/硬盘（通常为 硬盘）
  2. 以下情况不适用：
     1. 少量数据
     2. 频繁更新的字段
     3. 很少使用的字段
  3. 降低增删改的效率
- 索引的优势：
  1. 提高查询的效率（降低IO使用率）
  2. 降低CPU使用率（...order by ... desc）
     - 无索引情况下，会先将表数据排序再取数据
     - 有索引情况下，默认已经排序

3.2.2 索引分类

1. 单值索引
   - 单列， age；一个表可以多个单值索引， name
2. 唯一索引
   - 不能重复 id， 可以是null
3. 主键索引
   - 不能重复 id 不能是null
4. 复合索引
   - 多个列构成的索引（相当于二级目录）

3.2.3 创建索引

1.语句：

- 方式一：`create 索引类型 索引名 on 表名（字段）;`
- 方式二：`alter table 表名 add 索引类型 索引名 （字段名）`

2.单值

- 方式一：`create index dept_index on tb(dept);`
- 方式二：`alter table tb add indexdept_index(dept);`

3.唯一

- 方式一：`create unique index name_index on tb(name);`
- 方式二：`alter table tb add unique index name_index(name);`

4.复合

- 方式一：`create index dept_name_indeex on tb(dept,name);`
- 方式二：`alter table tb add index dept_name_index(dept,name);`

3.2.4 删除索引

`drop idnex 索引名 on 表名`

3.2.5 查看索引

`show index from 表名`

4.SQL执行计划

4.1 实例讲解

4.1.1 准备

```
create table course(
  cid int(3),
  cname varchar(20),
  tid int(3)
);

create table teacher(
  tid int(3),
  tname varchar(20),
  tcid int(3)
);

create table teacherCard(
  tcid int(3),
  tedesc varchar(200)
);
```

插入数据：

```
insert into course values(1,'java',1);
insert into course values(2,'html',1);
insert into course values(3,'sql',2);
insert into course values(4,'web',3);
```

```
insert into teacher values(1,'tz',1);
insert into teacher values(2,'tw',2);
insert into teacher values(3,'tl',3);
```

```
insert into teacherCard values(1,'tzdesc');
insert into teacherCard values(2,'twdesc');
insert into teacherCard values(3,'tldesc');
```

4.1.2 explain命令返回内容

```
        id: 编号
 select_type: 查询类型
      table: 表
  partitions: NULL
       type: 类型
possible_keys: 预测用到的索引
        key: 实际用到的索引
    key_len: 实际使用索引长度
        ref: 表之间单引用
       rows: 通过索引查询到的数据量
   filtered: 100.00
      Extra: 额外的信息
```

4.1.3 explain中的id详解

- 查询课程编号为2 或 教师编号为3 的老师信息

```
select t.* from teacher t, course c , teacherCard tc where t.tid = c.tid and t.tcid = tc.tcid and (c.cid = 2 or
tc.tcid = 3);

查询结果：
+------+-------+------+
| tid  | tname | tcid |
+------+-------+------+
|   1 | tz    |   1 |
|   3 | tl    |   3 |
+------+-------+------+

explain查看执行计划：
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-----------------------
------------------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra
                       |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-----------------------
------------------------+
| 1 | SIMPLE     | t    | NULL     | ALL | NULL        | NULL | NULL    | NULL |    3 |   100.00 | NULL
         |
| 1 | SIMPLE     | tc   | NULL     | ALL | NULL        | NULL | NULL    | NULL |    3 |    33.33 | Using where; Using
join buffer (Block Nested Loop) |
| 1 | SIMPLE     | c    | NULL     | ALL | NULL        | NULL | NULL    | NULL |    4 |    25.00 | Using where; Using
join buffer (Block Nested Loop) |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-----------------------
------------------------+
```

- 查询教授SQL课程的老师描述

```
select tc.tedesc from teacher t, course c , teacherCard tc where t.tid = c.tid and t.tcid = tc.tcid and
c.cname = 'sql';
->转子查询：
select tc.tedesc from  teacherCard tc where
tc.tcid = (select t.tcid from teacher t where
t.tid = (select c.tid from course c where c.cname= 'sql'));
```

```
sql查询结果：
+--------+
| tedesc |
+--------+
| twdesc |
+--------+

explain执行计划：
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | PRIMARY     | tc    | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    3 |   33.33 | Using where |
|  2 | SUBQUERY    | t     | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    3 |   33.33 | Using where |
|  3 | SUBQUERY    | c     | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    4 |   25.00 | Using where |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
```

- 结果分析：
  - id值相同，从上往下 顺序执行 `t - tc - c`
    - 排序根据表数据量笛卡尔积决定：

      ```
      t表数据量： 3
      c表数据量：4
      tc表数据量：3

      此时笛卡尔积：3*4*3=36
      虽然最后结果是一样的，但是中间是不一样的：3*4=12*3=24 | 3*3=9*4=36
      会优先选择后者方案：3*3*4
      ```

    - 数据量小的表 优先查询
  - id值不同，id值越大优先查询
  - id值相同，又有不同：id值越大优先；id值相同，从上往下 顺序执行

4.1.4 select_type分类

1. PRIMARY：包含子查询SQL中的 主查询 （最外层）
2. SUBQUERY： 包含子查询SQL中的子查询 （非最外层）
3. simple： 简单查询（不包含子查询，union）
4. derived:衍生查询（使用到临时表）
   - 在from子查询中只有一张表
   - 在from子查询中，如果有table1 union table2，则table1就是derived
5. union
6. union result:告知开发人员，哪些表之间存在union查询

4.1.5 type详解

- type:索引类型
- 效率：system > const > eq_ref > ref > range > index > all ， 要对type进行优化的前提：有索引
  - 其中：system ，const只是理想情况；实际能达到ref ， range

1.system（忽略）

- 只有一条数据的系统表 | 衍生表只有一条数据的主查询

```
create table test01(
  tid int(3),
  tname varchar(20)
);

insert into test01 values(1,'a');
commit;
alter table test01 add constraint tid_pk primary key(tid);
explain select * from (select * from test01) t where tid=1;

+----+-------------+--------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
| id | select_type | table  | partitions | type  | possible_keys | key     | key_len | ref   | rows | filtered |
Extra |
+----+-------------+--------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | test01 | NULL       | const | PRIMARY       | PRIMARY | 4       | const |    1 |   100.00 | NULL  |
+----+-------------+--------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
```

2.const

- 仅仅能查到一条数据的SQL，用于Primary key 或者unique索引 (类型与索引类型有关)

```
exelect tid from test01 where tid=1;

+----+-------------+--------+------------+-------+---------------+---------+---------+-------+------+----------+-------------+
| id | select_type | table  | partitions | type  | possible_keys | key     | key_len | ref   | rows | filtered | Extra       |
+----+-------------+--------+------------+-------+---------------+---------+---------+-------+------+----------+-------------+
|  1 | SIMPLE      | test01 | NULL       | const | PRIMARY       | PRIMARY | 4       | const |    1 |   100.00 | Using index |
+----+-------------+--------+------------+-------+---------------+---------+---------+-------+------+----------+-------------+
```

3.eq_ref(唯一性索引)

- 对于每个索引键的查询，返回匹配唯一行数据（有且只有1个，不能多，不能0）
- 常见于唯一索引和主键索引

```
alter table teacherCard add constraint pk_tcid primary key(tcid);
alter table teacher add constraint uk_tcid unique index(tcid);

explain select t.tcid from teacher t,teacherCard tc where t.tcid = tc.tcid;

+----+-------------+-------+------------+--------+---------------+---------+---------+--------------------+------+----------+-------
-------------------+
| id | select_type | table | partitions | type   | possible_keys | key     | key_len | ref                | rows | filtered |
Extra                |
+----+-------------+-------+------------+--------+---------------+---------+---------+--------------------+------+----------+-------
-------------------+
|  1 | SIMPLE      | t     | NULL       | index  | uk_tcid       | uk_tcid | 5       | NULL               |    3 |   100.00 | Using where;
Using index |
|  1 | SIMPLE      | tc    | NULL       | eq_ref | PRIMARY       | PRIMARY | 4       | sqlOptimize.t.tcid |    1 |   100.00 |
Using index          |
```

```
+----+-------------+-------+------------+--------+---------------+---------+---------+--------------------+------+----------+-------
-----------------+
```

以上sql，用到的索引是t.tcid 即t表中的tcid字段；如果t表单数据个数 和连接查询的数据个数一致，则可能满足eq_ref级别

4.ref(非唯一索引)

- 对于每个索引键的查询，返回匹配的所有行（0，多）
- 准备数据：

```
insert into teacher values(4,'tz',4);

insert into teacherCard values(4,'tz2desc');

alter table teacher add index index_name(tname);
```

- 测试

```
explain select * from teacher where tname='tz';

+----+-------------+---------+------------+------+---------------+------------+---------+-------+------+----------+-------+
| id | select_type | table   | partitions | type | possible_keys | key        | key_len | ref   | rows | filtered | Extra |
+----+-------------+---------+------------+------+---------------+------------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | teacher | NULL       | ref  | index_name    | index_name | 63      | const |    2 |   100.00 | NULL  |
+----+-------------+---------+------------+------+---------------+------------+---------+-------+------+----------+-------+
```

5.range

- 检索指定范围的行，where后面是一个范围i查询（between ， in > < >=）
- 特殊：in 有时候会失效，从而转为 无索引 all
- 测试

```
alter table teacher add index tid_index(tid);

explain select * from teacher t where t.tid in (1,2);

+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | t     | NULL       | ALL  | tid_index     | NULL | NULL    | NULL |    4 |    50.00 | Using where |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+

explain select * from teacher t where t.tid >3;

+----+-------------+-------+------------+-------+---------------+-----------+---------+------+------+----------+-------------
----------+
```

```
| id | select_type | table | partitions | type  | possible_keys | key      | key_len | ref  | rows | filtered |
Extra           |
+----+-------------+-------+------------+-------+---------------+----------+---------+------+------+----------+-------------
----------+
|  1 | SIMPLE      | t     | NULL       | range | tid_index     | tid_index | 5       | NULL |    1 |   100.00 | Using index
condition |
+----+-------------+-------+------------+-------+---------------+----------+---------+------+------+----------+-------------
----------+
```

6.index

- 查询全部索引中所有数据
- 测试

```
show index from teacher;
+---------+------------+------------+--------------+-------------+-----------+-------------+----------+--------+------+------
------+---------+---------------+
| Table   | Non_unique | Key_name   | Seq_in_index | Column_name | Collation | Cardinality |
Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+---------+------------+------------+--------------+-------------+-----------+-------------+----------+--------+------+------
------+---------+---------------+
| teacher |          0 | uk_tcid    |            1 | tcid        | A         |           3 |     NULL | NULL   | YES  | BTREE      |
        |
| teacher |          1 | index_name |            1 | tname       | A         |           4 |     NULL | NULL   | YES  | BTREE      |
     |         |
| teacher |          1 | tid_index  |            1 | tid         | A         |           4 |     NULL | NULL   | YES  | BTREE      |
        |
+---------+------------+------------+--------------+-------------+-----------+-------------+----------+--------+------+------
------+---------+---------------+

explain select tid from teacher;

+----+-------------+---------+------------+-------+---------------+----------+---------+------+------+----------+-----------
--+
| id | select_type | table   | partitions | type  | possible_keys | key      | key_len | ref  | rows | filtered |
Extra       |
+----+-------------+---------+------------+-------+---------------+----------+---------+------+------+----------+-----------
--+
|  1 | SIMPLE      | teacher | NULL       | index | NULL          | tid_index | 5       | NULL |    4 |   100.00 | Using
index |
+----+-------------+---------+------------+-------+---------------+----------+---------+------+------+----------+-----------
--+
```

7.all

- 查询全部表中所有数据

4.1.6 key_len简介

- 如果索引字段为nul，则会适用1个字节用于标识。
- utf8：1个字符等于3个字节

4.1.7 ref 简介

- 指明当前表所参照的字段
- `select ... where a.c = b.x`；其中`b.x`可以是常量，const

4.1.8 Extra简介

- using filesort : 性能消耗大，需要额外的一次排序（查询）；一般出现在order by语句
  - 避免：where 和order by 按照复合索引的顺序的使用，不要跨列或无序使用
    - 错误示例.. where a1 = '' order by a2;
    - ... where a1 = '' order by a1;
- using temporary: 性能损耗大，用到了临时表。一般出现在groub by语句
  - 避免： 查询哪些列，就根据那些列分组
- using index :性能提升；索引覆盖。原因：不能读取原文件，只能从索引文件中获取文件（不用回表查询）
  - 只要使用到额度列，全部都在索引中，就是索引覆盖using index
  - select age from test where age=12;
- using where： 需要回表查询
  - select age,name from test age=12; (name需要回表查询)
- impossible where : where 子句永远为false
  - select .... where a1="1" and a1="2";

5.优化示例

5.1准备

```
create table test02(
  a1 int(4) not null,
  a2 int(4) not null,
  a3 int(4) not null,
  a4 int(4) not null
);
alter table test02 add index index_a1_a2_a3_a4(a1,a2,a3,a4);
```

5.2测试

```
--1
explain select a1,a2,a3,a4 from test02 where a1=1 and a2=2 and a3=3 and a4=4;

+----+-------------+--------+------------+------+------------------+------------------+---------+-------------------------+------+----------+-------------+
| id | select_type | table  | partitions | type | possible_keys    | key              | key_len | ref                     | rows | filtered | Extra       |
+----+-------------+--------+------------+------+------------------+------------------+---------+-------------------------+------+----------+-------------+
|  1 | SIMPLE      | test02 | NULL       | ref  | index_a1_a2_a3_a4 | index_a1_a2_a3_a4 | 16      | const,const,const,const |    1 |   100.00 | Using index |
+----+-------------+--------+------------+------+------------------+------------------+---------+-------------------------+------+----------+-------------+
```

```
--推荐写法，因为 索引的使用顺序（where后面的顺序） 和复合索引的顺序一致

--2
explain select a1,a2,a3,a4 from test02 where a4=1 and a3=2 and a2=3 and a1=4;

+----+-------------+--------+------------+------+------------------+------------------+---------+-----------------------+------+----------+-------------+
| id | select_type | table  | partitions | type | possible_keys    | key              | key_len | ref                   | rows | filtered | Extra       |
+----+-------------+--------+------------+------+------------------+------------------+---------+-----------------------+------+----------+-------------+
|  1 | SIMPLE      | test02 | NULL       | ref  | index_a1_a2_a3_a4 | index_a1_a2_a3_a4 | 16      | const,const,const,const |    1 |   100.00 | Using index |
+----+-------------+--------+------------+------+------------------+------------------+---------+-----------------------+------+----------+-------------+

--虽然编写的顺序和索引的顺序不一致，但是经过SQL优化器会调整，结果会与上条一致

--3
explain select a1,a2,a3,a4 from test02 where a1=1 and a2=2  and a4=4 order by a3;

+----+-------------+--------+------------+------+------------------+------------------+---------+-------------+------+----------+-----------------------+
| id | select_type | table  | partitions | type | possible_keys    | key              | key_len | ref         | rows | filtered | Extra                 |
+----+-------------+--------+------------+------+------------------+------------------+---------+-------------+------+----------+-----------------------+
|  1 | SIMPLE      | test02 | NULL       | ref  | index_a1_a2_a3_a4 | index_a1_a2_a3_a4 | 8       | const,const |    1 |   100.00 | Using where; Using index |
+----+-------------+--------+------------+------+------------------+------------------+---------+-------------+------+----------+-----------------------+

--where后面的顺序 a1 , a2在索引的顺序，using index | 但是a4跨列，需要回表查询，所以索引无效，using where

--4
explain select a1,a2,a3,a4 from test02 where a1=1   and a4=4 order by a3;

+----+-------------+--------+------------+------+------------------+------------------+---------+-------+------+----------+----------------------------------+
| id | select_type | table  | partitions | type | possible_keys    | key              | key_len | ref   | rows | filtered | Extra                            |
+----+-------------+--------+------------+------+------------------+------------------+---------+-------+------+----------+----------------------------------+
|  1 | SIMPLE      | test02 | NULL       | ref  | index_a1_a2_a3_a4 | index_a1_a2_a3_a4 | 4       | const |    1 |   100.00 | Using where; Using index; Using filesort |
+----+-------------+--------+------------+------+------------------+------------------+---------+-------+------+----------+----------------------------------+

--a1  using index | a4 using where | a3 using filesort  a3跨列使用
--避免：where a1 and a2 and a4 order by a3 : 此时索引顺序： a1 a2 a3 | a4索引失效
--    或者where a1 and a4 order by a2 , a3
```

5.3总结

1. 如果（a,b,c,d）复合索引 和使用的顺序全部一致（且不垮列使用），则复合索引全部使用，如果部分一致，则使用部分索引
2. `where` 和 `order by` 拼起来 索引不垮列使用

6.优化案例

6.1单表优化

6.1.1准备

```
create table book(
   bid int(4) primary key,
   name varchar(20) not null,
   authorid int(4) not null,
   publicid int (4) not null,
   typeid int(4)not null
);

insert into book values(1,'tjava',1,1,2);
insert into book values(2,'tc',2,1,2);
insert into book values(3,'wx',3,2,1);
insert into book values(4,'math',4,2,3);
commit;
```

6.1.2 优化

6.1.2.1 未优化

```
--查询authorid = 1 且typeid为2或3的bid
explain select bid from book where typeid in (2,3) and authorid = 1 order by typeid desc;
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-----------------------------
---+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra
      |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-----------------------------
---+
|  1 | SIMPLE      | book  | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    4 |    25.00 | Using where; Using
filesort |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-----------------------------
---+
```

- 优化前type级别为：all

6.1.2.2 优化：加索引

```
--优化：加索引
alter table book add index index_bta(bid,typeid,authorid);

--执行上SQL
+----+-------------+-------+------------+-------+---------------+-----------+---------+------+------+----------+-----------------------
----------------------+
| id | select_type | table | partitions | type  | possible_keys | key       | key_len | ref  | rows | filtered | Extra
                      |
+----+-------------+-------+------------+-------+---------------+-----------+---------+------+------+----------+-----------------------
----------------------+
|  1 | SIMPLE      | book  | NULL       | index | NULL          | index_bta | 12      | NULL |    4 |   25.00 | Using where;
Using index; Using filesort |
+----+-------------+-------+------------+-------+---------------+-----------+---------+------+------+----------+-----------------------
----------------------+
```

- 优化后type级别为：index

6.1.2.3 优化：索引顺序调整

```
--根据SQL实际解析的顺序，调整索引的顺序
alter table book add index index_tab(typeid,authorid,bid); --虽然可以回表查询bid，但是将bid反倒索引表中 可以提升using index

--索引一旦进行升级 需要将之前废弃的索引删掉 防止干扰
drop index index_bta on book;

--执行上SQL
+----+-------------+-------+------------+-------+---------------+-----------+---------+------+------+----------+-----------------------
------+
| id | select_type | table | partitions | type  | possible_keys | key       | key_len | ref  | rows | filtered | Extra
      |
+----+-------------+-------+------------+-------+---------------+-----------+---------+------+------+----------+-----------------------
------+
|  1 | SIMPLE      | book  | NULL       | range | index_tab     | index_tab | 8       | NULL |    2 |   100.00 | Using where;
Using index |
+----+-------------+-------+------------+-------+---------------+-----------+---------+------+------+----------+-----------------------
------+
```

- 优化后type级别为：range

6.1.2.4 优化：范围查询优化

- 思路：因为范围查询in有时会索引失效，因此交换 索引的顺序
    1. 将之前的索引删掉 放干扰
    2. 添加索引atb

```
drop index index_tab on book;
alter table book add index index_atb(authorid,typeid,bid);

--执行以下SQL
explain select bid from book where  authorid = 1 and typeid in (2,3) order by typeid desc;
+----+-------------+-------+------------+-------+---------------+-----------+---------+------+------+----------+-------------
-------------+
| id | select_type | table | partitions | type  | possible_keys | key       | key_len | ref  | rows | filtered |
Extra                |
```

```
+----+-------------+-------+------------+-------+---------------+-----------+---------+------+------+----------+-------------
-------------+
| 1 | SIMPLE    | book | NULL     | range | index_atb    | index_atb | 8     | NULL |   2 |  100.00 | Using
where; Using index |
+----+-------------+-------+------------+-------+---------------+-----------+---------+------+------+----------+-------------
-------------+

--视频教程优化到ref级别的

--本例中：
--    同时出现了using index (不需要回原表）using where（回原表）
--    解释：where  authorid = 1 and typeid in (2,3) 中authorid在索引中，因此不需要回原表，而typeid
虽然在索引中，但是含in的范围查询让typeid索引失效，所以需要回原表
--证明：将typeid in （2，3） 改为 typeid=2
explain select bid from book where  authorid = 1 and typeid =2 order by typeid desc;
+----+-------------+-------+------------+------+---------------+-----------+---------+-------------+------+----------+-------
------+
| id | select_type | table | partitions | type | possible_keys | key      | key_len | ref        | rows | filtered |
Extra     |
+----+-------------+-------+------------+------+---------------+-----------+---------+-------------+------+----------+-------
------+
| 1 | SIMPLE    | book | NULL     | ref | index_atb    | index_atb | 8     | const,const |   1 |  100.00 |
Using index |
+----+-------------+-------+------------+------+---------------+-----------+---------+-------------+------+----------+-------
------+

--ref级别
```

6.1.2.5总结

1. 最佳左前缀，保持索引的定义和使用的顺序一致性
2. 索引需要逐步优化
3. 将含In的范围查询 放到where条件的最后，防止失效

6.2 多表优化

6.2.1准备

```
create table teacher2(
  tid int(4)primary key,
  cid int(4)not null
);

insert into teacher2 values(1,2);
insert into teacher2 values(2,1);
insert into teacher2 values(3,3);

create table course2(
  cid int(4),
  cname varchar(20)
);

insert into course2 values(1,'java');
insert into course2 values(2,'web');
insert into course2 values(3,'c++');
```

- 左连接：

```
select * from teacher2 t left  outer join course2 c on t.cid=c.cid where c.cname='java';

+-----+-----+------+-------+
| tid | cid | cid  | cname |
+-----+-----+------+-------+
|  2  |  1  |   1  | java  |
+-----+-----+------+-------+
```

6.2.2 优化前考虑

- 索引往哪儿加
  - 小表驱动大表 -> 小表.id = 大表.id
  - 索引建立在经常使用字段上 -> 本例中 t.cid = c.cid 可知，t.cid字段使用频繁，因此该给字段加索引
  - 一般情况下左外连接，给左表加索引； 右外连接，给右表加索引

6.2.3 优化前

```
explain select * from teacher2 t left  outer join course2 c on t.cid=c.cid where c.cname='java';

+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-----------------------------------------------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra                                               |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-----------------------------------------------------+
|  1 | SIMPLE      | c     | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    3 |    33.33 | Using where                                         |
|  1 | SIMPLE      | t     | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    3 |    33.33 | Using where; Using join buffer (Block Nested Loop) |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-----------------------------------------------------+
```

6.2.4优化：加索引

```
--1
alter table teacher2 add index index_t_cid(cid);

explain select * from teacher2 t left  outer join course2 c on t.cid=c.cid where c.cname='java';
+----+-------------+-------+------------+------+---------------+-------------+---------+------------------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key         | key_len | ref              | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+-------------+---------+------------------+------+----------+-------------+
|  1 | SIMPLE      | c     | NULL       | ALL  | NULL          | NULL        | NULL    | NULL             |    3 |    33.33 | Using where |
|  1 | SIMPLE      | t     | NULL       | ref  | index_t_cid   | index_t_cid | 4       | sqlOptimize.c.cid |    1 |   100.00 | Using index |
```

```
+----+-------------+-------+------------+------+---------------+-------------+---------+------------------+------+----------+-------
-------+

--2
alter table course2 add index index_c_cname(cname);

explain select * from teacher2 t left  outer join course2 c on t.cid=c.cid where c.cname='java';

+----+-------------+-------+------------+------+---------------+---------------+---------+------------------+------+----------+----
---------+
| id | select_type | table | partitions | type | possible_keys | key           | key_len | ref              | rows | filtered |
Extra      |
+----+-------------+-------+------------+------+---------------+---------------+---------+------------------+------+----------+----
---------+
|  1 | SIMPLE      | c     | NULL       | ref  | index_c_cname | index_c_cname | 63      | const            |    1 |   100.00 |
Using where |
|  1 | SIMPLE      | t     | NULL       | ref  | index_t_cid   | index_t_cid   | 4       | sqlOptimize.c.cid |    1 |   100.00 |
Using index |
+----+-------------+-------+------------+------+---------------+---------------+---------+------------------+------+----------+----
---------+
```

6.2.5总结

1. 小表驱动大表
2. 索引建立在经常查询的字段上

6.3避免索引失效的原则

1. 复合索引
    1. 不要跨列或无序使用（最佳左前缀）
    2. 尽量使用全索引匹配
    3. 如果左侧索引失效，右边索引全部失效
    4. 不能使用不等于（!= <>）或is null(is not null)，否则自身以及右侧所有全部失效
2. 不要在索引上进行任何操作（计算，函数，类型转换），否则索引失效
    - 计算例子：select ... where A.x*3 = ... (A.x为索引)
    - 类型转换例子：| `select * from teacher where tname = 'abc'` | `select * from teacher where tname = 123` ->程序低层将123 ->'123' 即类型转换
3. 我们学习索引优化，是一个大部分情况适用的结论，但由于SQL优化器等原因，该结论不是100%正确
4. 一般情况，范围查询（> < in ）之后的索引失效
5. 补救：尽量使用索引覆盖（using index）
    - `select a, b ,c .... where a=.. and b =.. ;`
6. like 尽量以'常量'开头 ，不要以'%'开头，否则索引失效
    - selece * from teacher where name like '%x%'; -- name索引失效
    - 可以使用索引覆盖进行补救 `select name from teacher where name like '%x%'`
7. 尽量不使用or 否则索引失效

7.其他的优化方法

7.1 Exist和In

- 语法：`select ... from table where exist/in (子查询)`
- 选择:
    1. 如果主查询的数据集大，则使用In
    2. 如果子查询的数据集大，则使用Exist
- exist语法：将主查询的结果，放到子查询结果中进行条件校验（看子查询 是否有数据，如果有数据，则校验成功），如果符合校验，则保留数据；

7.2 Order by优化

- using filesort: 有两种算法：双路排序，单路排序（根据IO的次数）
    - 双路排序：扫描2次磁盘（1：从磁盘读取排序字段，对排序字段进行排序；2：扫描其他字段） --IO较消耗性能
    - 单路排序：只读取一次（全部字段），会有一定的隐患（不一定只进行一次IO ）
- 注意：单路比双路会占用更多的俄buffer
- 单路排序在使用时，如果数据大，可以考虑调大buffer的容量大小：`set max_length_for_sort_data = 1024;`
    - max_length_for_sort_data 的值太低，则mysql会自动从单路转换到双路（需要排序的列 > max_length_for_sort_data ）
- 提高order by查询的策略：
    1. 选择使用单路，双路 ；调整buffer的容量大小
    2. 避免使用 select *...
    3. 复合索引不要跨列使用，避免using filesort
    4. 保证全部的排序字段 排序的一致性（都是升序或者降序）

8.SQL排查-慢查询日志

8.1概述

- MySql提供的一种日志记录，用于记录mysql响应时间超过阈值的SQL语句
- 慢查询日志默认是关闭的
- 建议：开发调优是打开的，而最终部署是关闭的
- 检查是否开启了慢查询日志：`show variables like '%slow_query_log%'`
- 慢查询开启：
    - 临时开启：`set global slow_query_log = 1;`
    - 永久开启：在`/etc/my.cnf`中追加配置：

```
[mysqld]
slow_query_log=1
slow_query_log_file=/var/lib/mysql/localhost-slow.log
```

- 慢查询阈值：`show variables like '%long_query_time%';`
    - 临时设置阈值：`set global long_query_time = 5`
    - 永久设置：在`/etc/my.cnf`中追加配置：

```
[mysqld]
long_query_time = 5
```

- 查询超过阈值的SQL：
  - `show global status loke '%slow_queries%';`
  - `cat /var/lib/mysql/localhost-slow.log`
  - 通过mysqldumpslow工具查看慢SQL `mysqldumpslow -s r -t 3 /var/lib/mysql/localhost-slow.log`
    - -s：排序方式
    - -r：逆序
    - -l：锁定时间
    - -g：正则匹配模式

9.海量数据分析

9.1准备

```
--创建表 部门表 员工表
create table dept(
   dno int(5) primary key default 0,
   dname varchar(20) not null default '',
   loc varchar(30) default ''
)engine = innodb default charset=utf8;

create table emp(
   eno int(5) primary key ,
   ename varchar(20) not null default '',
   job varchar(20) default '',
   deptno int(5) not null default 0
)engine = innodb default charset=utf8;
```

9.2插入海量数据

9.2.1创建存储函数

- 随机字符：

```
delimiter $
create function randstring(n int) returns varchar(255)
begin
   declare all_str varchar(100) default
'zxcvbnmasdfghjklqwertyuiopZXCVBNMLKJHGFDSAQWERTYUIOP';
   declare return_str varchar(255) default '';
   declare i int default 0;
    while i < n
      do
          set return_str = concat(return_str , substring(all_str ， FLOOR(1+rand()*52) ， 1 ));
          set i=i+1;
     end while;
     return return_str;
end $
```

- 随机整数：

```
delimiter $
create function randnum() returns int(5)
begin
   declare i int default 0;
   set i = floor(rand()*100);
    return i;
end $
```

- 插入海量数据函数：emp表

```
delimiter $
create procedure insert_emp(in eid_start int(10),in data_times int(10))
begin
 declare i int default 0;
 set autocommit = 0;
 repeat
    insert into emp values(eid_start+i,randstring(5),'other',randnum());
    set i=i+1;
    until i=data_times
 end repeat;
commit;
end $
```

- 插入海量数据函数：dept表

```
delimiter $
create procedure insert_dept(in dno_start int(10)， in data_times int(10))
begin
   declare i int default 0;
   set autocommit = 0;
   repeat
    insert into dept values(dno_start+i,randstring(6),randstring(8));
    set i=i+1;
    until i=data_times
   end repeat;
   commit ;
end $
```

9.2.2 执行存储函数

```
delimiter ;
call insert_emp(1000,800000);
call insert_dept(10,30);
```

9.2.3分析海量数据

1.profiles分析 (默认关闭)

- 作用：记录所有profiling打开之后的全部SQL执行语句花费时间；缺点：只能看到总共消费时间
- 开启： `set profiling = on;`
- 查看： `show profiles;`

2.精确分析：sql诊断

- 精确分析消费时间：
  - 显示所有花费的时间： `show profile all for query --Query_Id;`
  - 显示cpu和io花费的时间： `show profile cpu,block io for query --Query_Id;`

3.全局查询日志

- 记录开启之后，全部SQL语句（仅仅在调优，开发过程中打开即可，在最终的部署一定关闭）
- 查询状态： `show variables like '%genneral_log%';`
- 临时打开 ：
  - `set global general_log = 1;` --开启全局日志
    - `set global log_output='table';` --日志记录在table中
    - `set global log_output='file';` --日志记录在文件
    - `set global genral_log_file ='/tmp/genral_log'` --设置存储日志文件

10.锁机制

10.1分类

- 按操作类型分：
  - 读锁（共享锁）：对同一个数据，多个读操作可以同时进行，互不干扰
  - 写锁（互斥锁）：如果当前写操作没有完毕，则无法进行其他的读操作，写操作
- 按操作范围分：
  - 表锁：一次性对一张表整体加锁；如MyISAM存储引擎，开销小，加锁快，无死锁，但是锁的范围大，容易发生锁冲突，并发度低
  - 行锁：一次性对一条数据加锁，如InnoDB，开销大，加锁慢，容易出现死锁，不易发生锁冲突，并发度高
  - 页锁

10.2 表锁加锁

- 指令格式： `lock table 表 read/write`
- 释放锁： `unlock tables;`
- 准备：

```
create table tablelock(
   id int(5) primary key auto_increment,
   name varchar(20)
)engine=myisam;

insert into tablelock(name) values('a1');
insert into tablelock(name) values('a2');
insert into tablelock(name) values('a3');
insert into tablelock(name) values('a4');
insert into tablelock(name) values('a5');
```

10.2.1查看加锁的表

```
show open tables;

+-------------------+----------------------------------------------------+--------+-------------+
| Database          | Table                                              | In_use | Name_locked |
+-------------------+----------------------------------------------------+--------+-------------+

--0：代表没加锁  1:加锁
```

10.2.2加读锁

```
lock table tablelock read;
```

10.2.3 读锁测试

- 会话1加锁后：

```
select * from tablelock;
+----+------+
| id | name |
+----+------+
|  1 | a1   |
|  2 | a2   |
|  3 | a3   |
|  4 | a4   |
|  5 | a5   |
+----+------+
--读可以

delete from tablelock where id= 1;
---ERROR 1099 (HY000): Table 'tablelock' was locked with a READ lock and can't be updated;
--写不可以

select * from dept;
--ERROR 1100 (HY000): Table 'dept' was not locked with LOCK TABLES

delete from emp where eno=10;
--ERROR 1100 (HY000): Table 'emp' was not locked with LOCK TABLES
--对其他的表操作也不可以
```

- 会话1加锁后的会话2

```
select * from tablelock;
+----+------+
| id | name |
+----+------+
| 1 | a1  |
| 2 | a2  |
| 3 | a3  |
| 4 | a4  |
| 5 | a5  |
+----+------+
--可以对加锁表进行读

delete from tablelock where id= 1;
--会等待会话1将表锁释放

select * from dept;
delete from emp where eno=10;
--Query OK, 0 rows affected (0.00 sec)
--对其他的表正常操作
```

- 总结
  - 会话1对A表加了read锁
    - 则会话1只可以对A表进行读操作，不能进行写操作，也不能对其他表进行读写操作
    - 其他会话可以对A表读操作，但是写操作需要等待会话1将表锁释放才能进行下一步操作
    - 其他会话对除了A表以外的表可以正常读写操作

10.2.4加写锁

```
lock table tablelock write;
```

- 会话1对A表加了写锁
  - 会话1可以对A表进行任何操作（增删改查），但是不能操作其他表
  - 其他会话对A表进行增删改查操作前提是会话1对A表释放写锁

10.2.5表锁分析

- 分析表锁定的严重程度：
  - ```
    show status like '%table%';

    --Table_locks_immediate   可能获取到的锁数
    --Table_locks_waited      需要等待的表锁数（如果值越大，说明存在越大的锁竞争）
    ```
  - 一般建议 `Table_locks_immediat / Table_locks_waited > 5000` ,建议采用InnoDB引擎，否则MyISAM引擎

10.3 mysql表级锁的锁模式
    ```

MySQL表级锁的锁模式
MyISAM在执行查询语句（SELECT）前，会自动给涉及的所有表加读锁，
在执行更新操作（DML）前，会自动给涉及的表加写锁。
所以对MyISAM表进行操作，会有以下情况：
a、对MyISAM表的读操作（加读锁），不会阻塞其他进程（会话）对同一表的读请求，
但会阻塞对同一表的写请求。只有当读锁释放后，才会执行其它进程的写操作。
b、对MyISAM表的写操作（加写锁），会阻塞其他进程（会话）对同一表的读和写操作，
只有当写锁释放后，才会执行其它进程的读写操作。

10.4 行锁

10.4.1准备

```
create table linelock(
   id int(5) primary key auto_increment,
   name varchar(20)
);
insert into linelock(name) values('1');
insert into linelock(name) values('2');
insert into linelock(name) values('3');
insert into linelock(name) values('4');
insert into linelock(name) values('5');

--为了研究行锁，暂时关闭自动提交
set autocommit = 0;
```

10.4.1测试

```
--会话1
update linelock set name='a1' where id= 1;

--会话2
upadte linelock set name='1' where id= 1;
--会话2会等待会话1结束事务（commit/rollback）,才能对数据id=1进行操作
```

10.4.2总结

- 会话1对表linelock的数据id=1进行更新（InnoDB自动加行锁），如果此时会话1还未结束事务，其他会话对该行数据的写操纵就要等待会话1的事务结束事务操作(commit/rollback)，才能进行操作
- 表锁是通过unlock tables释放锁；行锁通过事务（commit/rollback）解锁

10.4.3行锁注意事项

- 如果没有索引，则行锁会转换为表锁

```
--会话1
update linelock set name='a1' where name='3';
--会话2
update linelock set name='a1' where name='4' ;
--操作不同行数据，互不干扰

------------------------------------------------------------

--会话1
```

```
update linelock set name='a1' where name=3;
--会话2
update linelock set name='a1' where name=4 ;
--此时会话2的操作会处于等待状态，说明数据阻塞（加锁）
--原因：发生类型转换，索引失效，因此转为表锁
```

- 行锁的一种特殊情况：间隙锁：值在范围内，但不存在

  ```
  update linelock set name = 'x' where id > 1 and id < 9 ; --没有id=7的数据
  ```

  mysql会自动给间隙（2-8行） 加锁，即本题会自动给id=7的数据加间隙锁

10.4.4 行锁优缺点

- 缺点：比表锁性能损耗大
- 优点：并发能力强，效率高

10.4.5 行锁分析

```
show status like '%innodb_row_lock%';

--| Innodb_row_lock_current_waits 当前正在等待锁的数量
--| Innodb_row_lock_time          等待总时长
--| Innodb_row_lock_time_avg      平均等待时长
--| Innodb_row_lock_time_max      最大等待时间
--| Innodb_row_lock_waits         等待的次数
```

11 其他

11.1关闭自动提交

1. ```
   set autocommit=0;
   ```
2. ```
   start transaction ;
   ```
3. ```
   begin ;
   ```
   ```