MySQL

[1. 联合查询 1](#_179)

[1.1. [inner] join 1](#_180)

[1.2. left join 1](#_181)

[1.3. right join 1](#_182)

[2. explain 1](#_174)

[2.1. select\_type 1](#_175)

[2.2. type 1](#_176)

[2.3. possible\_key 2](#_177)

[2.4. rows 2](#_178)

[3. 分库分表 2](#_161)

[3.1. 垂直拆分 2](#_162)

[3.2. 水平拆分 2](#_163)

[4. 日志 2](#_152)

[4.1. Error Log：记录运行过程中的Error等信息 2](#_153)

[4.2. General Query Log：记录mysql的日常日志，包括查询、修改、更新 2](#_154)

[4.3. Binary Log 3](#_155)

[4.4. Slow Query Log 3](#_156)

[4.5. undo/redo 3](#_157)

[5. 存储数据结构 3](#_101)

[5.1. B树 3](#_102)

[5.2. B+树 3](#_103)

[5.3. B+与B树的比较 4](#_105)

[6. 事务属性ACID 4](#_104)

[7. 隔离级别 4](#_106)

[7.1. 未提交读，会出现脏读 4](#_107)

[7.2. 提交读，会出现不可重复读 4](#_108)

[7.3. 可重复读（InnoDB默认） 4](#_109)

[7.3.1. 幻读 4](#_129)

[7.3.2. 实际实现上通过间隙锁来实现 4](#_130)

[7.4. 可串行化，并发度下降，性能最差 5](#_110)

[8. 存储引擎 5](#_136)

[8.1. InnoDB 5](#_137)

[8.2. MyISAM 6](#_138)

[9. MVCC 6](#_139)

[10. 优化语句 6](#_140)

[10.1. 索引选择 6](#_141)

[10.2. 表连接>(not) exists > (not) in 7](#_149)

[10.3. 索引相关 7](#_168)

[10.3.1. order by只能在一个索引下使用，并且具有唯一顺序 7](#_148)

[10.3.2. 尽量避免使用is null或is not null，因为不会使用索引 7](#_146)

[10.3.3. 字段加函数不会使用索引 7](#_143)

[10.3.4. 尽量将or替换为union all 7](#_142)

[10.3.5. 使用组合索引时必须要包括最左索引，最左前缀匹配：尽量把=放前面 8](#_145)

[10.3.6. 字段加运算符不会使用索引，尽量把运算放在数值上 8](#_144)

[10.3.7. !=不会使用索引 8](#_147)

[11. 锁使用 8](#_111)

[11.1. 锁类型 8](#_118)

[11.1.1. 锁粒度 8](#_119)

[11.1.2. 封锁类型 8](#_122)

[11.2. 两阶段锁协议(innoDB) 8](#_117)

[11.3. 加锁 8](#_123)

[11.3.1. 隐式 8](#_124)

[11.3.2. 显式（必须要在事务内才有效） 9](#_125)

[11.4. innoDB加锁处理 9](#_183)

[11.4.1. 快照读 9](#_184)

[11.4.2. 当前读 9](#_185)

[12. 常用语句 9](#_126)

[12.1. 显示建表时的详细信息 10](#_127)

[12.2. 显示详细字段信息 10](#_128)

[13. 索引 10](#_131)

[13.1. 索引分类 10](#_171)

[13.1.1. 普通索引 10](#_132)

[13.1.2. 唯一索引 10](#_133)

[13.1.3. 联合索引（最左匹配） 10](#_135)

[13.1.4. 全文索引 10](#_134)

[13.2. 创建索引 10](#_150)

[13.3. 查看索引 11](#_151)

[13.4. Change Buffer 11](#_167)

[14. 主从同步 11](#_158)

[14.1. 流程 11](#_159)

[14.2. 原理 11](#_160)

[15. Trick 12](#_165)

[15.1. 分段查询 12](#_166)

[15.2. 远程连接 12](#_173)

[16. 权限 12](#_172)

[17. 问题 12](#_186)

[17.1. truncate、drop和delete 12](#_187)

[17.2. DDL和DML 13](#_188)

[17.3. 插入多条数据 13](#_189)

# 联合查询

## [inner] join

select student.studentId, score.score from student join score using(studentId);

select student.studentId, score.score from student join score on student.studentId = score.studentId;

## left join

## right join

# explain

## select\_type

select\_type表示查询的类型，常见取值有：

SIMPLE：表示此查询不包含union查询或子查询。最常见。

PRIMARY：表示此查询是最外层的查询

UNION：表示此查询是UNION的第二或随后的查询

DEPENDENT UNION：UNION中的第二个或后面的查询语句，取决于外面查询。

UNION RESULT：UNION的结果

SUBQUERY：子查询中第一个select

DEPENDENT SUBQUERY：子查询中第一个select，取决于外面的查询。

## type

type提供了判断查询是否高效的重要依据。可以通过它判断是全表扫描还是索引扫描。

system：表中只有一条数据

const：针对主键或唯一索引的等值查询扫描。

eq\_ref:多出现在多表的join查询中，表示对于前表的每一个结果都只能匹配到后表的一行结果。

ref：多出现在多表的join查询，针对于非唯一或非主键索引，或者是使用了最左前缀规则索引的查询

range：表示使用索引范围查询。

index：表示全索引扫描

all：表示全表扫描。性能最差的查询。

性能比较：

all < index < range < ref < eq\_ref < const < system

## possible\_key

可能会被用到的索引

## rows

MySQL查询优化器根据统计信息，估算SQL要查找到结果集需要扫描读取的数据行数等。

这个值非常直观显示SQL的效率好坏，原则上rows越少越好。

# 分库分表

## 垂直拆分

垂直分表：“大表拆小表”，基于列字段进行的。一般是表中字段较多，将不常用的，数据较大，长度较大（比如text字段）拆分至“扩展表”。

垂直分库：针对系统中的不同业务进行拆分，比如用户User一个库，商品Product一个库，订单Order一个库。切分后，要放在多个服务器，而不是一个服务器上。

## 水平拆分

水平分表：针对数据量巨大的单张表（比如订单表），按照某种规则（RANGE,HASH取模等），切分到多张表。

水平分库分表：将单张表的数据切分到多个服务器上，每个服务器具有相应的库与表，只是表中数据集合不同。

水平分库分表规则：

RANGE：从0-10000一个表，10001到20000一个表。

HASH取模：比如对用户ID哈希后取模。

地理区域

时间：将6个月前甚至一年前的数据且出去放到另外一张表，因为随着时间流逝，这些表的数据被查询的概率变小，所以没必要和“热数据”放在一起，这个也是“冷热数据分离”。

# 日志

## Error Log：记录运行过程中的Error等信息

## General Query Log：记录mysql的日常日志，包括查询、修改、更新

## Binary Log

主要用于备份恢复、回滚，有三种模式：

statment：每一行会修改数据的sql都会记录到master Log中，再交给slave进行执行

row：记录成每一行数据数据被修改的形式。

优点：不记录执行的SQL语句上下文

缺点：日志会非常大

mixed：根据执行的每一条SQL语句去选择row模式还是statement模式

## Slow Query Log

可记录查询时间超过某个设定值的SQL

1. 打开慢查询日志

slow\_query\_log=1

2. 慢查询日志存储路径

slow\_query\_log\_file=/var/log/mysql/log-slow-queries.log

3. SQL执行时间大于3秒，则记录日志

long\_query\_time=3

## undo/redo

# 存储数据结构

## B树

M阶B树：

任意非叶子结点最多有M个孩子，M>2

根节点的孩子数为[2, M]

除根节点以外的非叶子结点的孩子数为[M/2, M]，向上取整。

非叶子结点的关键字个数=孩子数-1

所有叶子结点位于同一层。

k个关键字把节点拆成k+1段，分别指向k+1个孩子。

## B+树

**M阶B+树**：

n棵子树的非叶子结点中含有n个关键字，这些关键字不保存数据（b树保存数据）

所有的叶子结点包含了全部关键字的信息，及指向含有这些关键字记录的指针，且叶子结点本身依关键字顺序链接。

所有非叶子结点可看成索引部分，仅含其子树中最大（或最小）关键字

通常在b+树上有两个头指针，一个指向根节点，另一个指向关键字最小的叶子结点。

在插入时先找到叶子结点，然后根据关键字个数判断是否分裂。

## B+与B树的比较

优点：

B+树的中间节点不保存数据，所以磁盘可容纳更多节点元素，因此树高较小。

B+数查询必须查找到叶子结点，因此B+数查找更加稳定。

对于范围查找，B+树只需要遍历叶子节点链表即可。

缺点：

主键不是有序递增的，会导致每次插入数据产生大量的数据迁移和空间碎片。

即使主键是有序递增的，大量写请求的分布仍是随机的。

适用：MySql（B+），MongoDB（B）

# 事务属性ACID

原子性：一起成功，一起失败

一致性：事务执行前后都保持一致性状态

隔离性：事务的修改在提交前不可见

持久性：提交后会永远保存在数据库中

# 隔离级别

## 未提交读，会出现脏读

## 提交读，会出现不可重复读

## 可重复读（InnoDB默认）

### 幻读

### 实际实现上通过间隙锁来实现

间隙锁：对于原始数据形成一个个间隙（gap），锁住这些间隙。

前提：必须在RR级别上，检索条件必须有索引。

作用：防止幻读，防止间隙内有新数据被插入，防止已存在的数据更新成间隙内的数据

锁定区域：根据检索条件向左寻找最靠近检索条件的记录值A，向右寻找靠近检索条件的记录值B，区间为(A, B)，是开区间。

注意：

普通索引上，不管什么查询，只要加锁，都会产生间隙锁。

如果二级索引的值都一样，则会考虑主键索引的先后顺序。

分成两大类考虑

**唯一索引**

1. 等值查询：select \* from gap where id = 3 for update;

记录存在，则只会产生记录锁；记录不存在，往左往右查找到一个开区间，如(1, 5)进行锁定。

2. 更新：update gap set content = "gogo" where id = 3;

记录存在，仅仅锁住单条记录（记录锁）；记录不存在，往左往右查找到一个开区间，如(1, 5)进行锁定。

3. 范围查询：select \* from gap where id between 4 and 5 for update;

若score = 4、5的记录都不存在，则会从4往左找到第一个记录A，往右找到第一个记录B，形成(A, B]的间隙锁

若score = 4、5的记录存在，则锁住[4, 5]，(5, B]的间隙

**非唯一索引**

1. 等值查询：select \* from gap where id = 3 for update;

不管记录是否存在，都生成间隙锁，锁住(A, B)，A是小于3的第一个id，B是大于3的第一个id。

2. 使用范围查询：select \* from gap where id between 4 and 5 for update;

若score = 4、5的记录都不存在，则会从4往左找到第一个记录A，往右找到第一个记录B，形成(A, B]的间隙锁

若score = 4、5的记录存在，则锁住[4, 5]，(5, B]的间隙

2. 使用范围查询：select \* from gap where id > 3 and id < 5 for update;

按照该范围找一个左开右闭的间隙锁。

## 可串行化，并发度下降，性能最差

# 存储引擎

## InnoDB

**存储：**数据都存储在一个文件中（也可能是多个文件，或者是独立的表空间）

**聚集索引：**会依次根据主键、非空的唯一索引、隐式创建一个自增的列来作为聚集索引。 无特殊需求下Innodb建议使用与业务无关的自增ID作为主键

聚集索引既存储了**索引**，又存储了**元组**。

innoDB数据是存在**索引的叶子页**上。而myISAM索引的叶子页只是存了偏移。

**辅助索引**：辅助索引的叶子节点中存放的是主键的键值，查到主键后，再使用主键值在聚集索引中进行第二次搜索。

**锁支持**：表锁和行锁

## MyISAM

存储结构：分为三个文件存储，.frm存储表定义，.MYD存储数据文件，.MYI存储索引文件。

加锁与并发：表锁，读时加共享锁，写时加排他锁。MyISAM可以实现并发插入，但是需要配合全局变量concurrent\_insert（NEVER，AUTO（中间无空洞时可插）, ALWAYS）只有后面两个才支持。

索引特性：支持全文索引。

压缩表：如果表在创建并导入数据以后，不会再进行修改操作，那么适合压缩表。压缩时表中的记录是独立压缩的，所以读取单行的时候不需要去解压整个表（甚至也不解压行所在的整个页面）

性能：由于数据紧密存储，所以在某些场景下性能很好，适合读多写少的场景。

# MVCC

适用于提交读和可重复读。

mysql中innoDB的MVCC，是通过在每行记录后面保存两个隐藏的列来实现的。这两个列，一个保存了行的创建时间，一个保存了行的过期时间。这里的时间并不是具体的时间，而是系统版本号。

查询时只查找到早于当前事务版本的数据行。小于删除版本号

删除/插入则为新插入的行保留当前事务的版本号，前者保留在过期时间，后者保留在创建时间。

更新：插入一行新记录，保存当前事务版本，同时更新保存当前系统的版本号到旧行的删除时间。

# 优化语句

## 索引选择

选择区分度最高的列作为索引：

selet count(distinct col)/count(1) from xxx;

如果是对字符串计算前缀索引的话：

select count(1) as cnt, LEFT(city, 7) as pref

from table group by pref order by cnt desc limit 10;

## 表连接>(not) exists > (not) in

## 索引相关

### order by只能在一个索引下使用，并且具有唯一顺序

**前提：**

alter table t1 add index(a, b);

alter table2 add index(c);

**结果：**

不使用索引

select \* from t1 order by a,c; 属于不同的索引

select \* from t1 order by b; 不是完整的索引

使用索引

select \* from t1 order by a, b;

select \* from t1 order by c desc;

### 尽量避免使用is null或is not null，因为不会使用索引

### 字段加函数不会使用索引

### 尽量将or替换为union all

老版本

select \* from student where studentId = 1 union all (select \* from student where studentname = "乐乐");

MySQL5.0开始引入了索引合并的策略。能够使用or的两个索引进行扫描，并对结果进行合并。

### 使用组合索引时必须要包括最左索引，最左前缀匹配：尽量把=放前面

### 字段加运算符不会使用索引，尽量把运算放在数值上

select \* from student where age + 10 < 50;

select \* from studetn where age < 40; 效率高于前者

### !=不会使用索引

# 锁使用

## 锁类型

### 锁粒度

#### 行锁（myISAM不支持)

#### 页锁（innoDB不支持)

#### 表锁

### 封锁类型

#### 读写锁

#### 意向锁

##### X/S锁

##### IX/IS锁（任意IX/IS锁之间都是兼容的）

## 两阶段锁协议(innoDB)

增长阶段获取锁，缩减阶段释放锁。

## 加锁

### 隐式

### 显式（必须要在事务内才有效）

select \* from t1 lock in share mode 共享锁

select \* from t1 for update 排他锁

以上根据是否索引决定是行锁还是表锁

## innoDB加锁处理

### 快照读

### 当前读

delete from table where id = 10;

根据上面这条语句分情况进行分析。

组合1：id列是主键，RC隔离级别

将主键id=10的记录加上X锁。

组合2：id列是二级唯一索引，RC隔离级别

先将id=10这个二级索引加X锁，然后找到对应主键所在的记录并加X锁。

组合3：id列是二级非唯一索引，RC隔离级别

将所有id=10的二级索引都加X锁，然后所有对应id=10的主键记录加X锁。

组合4：id列上没有索引，RC隔离级别

由于没有索引，所以是全表扫描，并加表级X锁。

组合5：id列是主键，RR隔离级别

将id=10这一条记录加上X锁。

组合6：id列是二级唯一索引，RR隔离级别

先将id=10这个二级索引加X锁，然后找到对应主键所在的记录并加X锁。

组合7：id列是二级非唯一索引，RR隔离级别

相比于组合3多了一个GAP锁

组合8：id列上没有索引，RR隔离级别

表X锁。

# 常用语句

## 显示建表时的详细信息

show create table xxx:

## 显示详细字段信息

show full columns from xxx;

# 索引

## 索引分类

### 普通索引

### 唯一索引

#### 插入速度比普通索引慢

因为唯一索引需要保证唯一性，需要将数据页加载进内存才能判断是否违反唯一性约束。但是既然数据页都加载到内存了，倒不如直接更新内存中的数据页，没必要使用Change Buffer。

所以在插入速度上，唯一索引慢于普通索引（能利用Change Buffer）。

#### 搜索速度比普通索引快

普通索引在找到满足条件的第一条记录后，还需要判断下一条记录，直到第一个不满足条件的记录出现。

唯一索引在找到满足条件的第一条记录后，直接返回，不用判断下一条记录了。

### 联合索引（最左匹配）

key test\_col1\_col2\_col3 on test(col1, col2, col3)

上面的语句实际建立了(col1)、(co1,col2)和(col1,col2,col3)三个索引

### 全文索引

## 创建索引

第一种方法

create index indexname on mytable(fieldname(length))

如果是char，varchar，length可以小于实际长度

如果是Blob和Text，必须指定length

第二种方法

alter table mytable add index indexName(columnName)

## 查看索引

show index from tableName \G;

## Change Buffer

对表执行insert、update和delete操作时，索引列的值（尤其是二级索引的值）通常未按顺序排序，需要大量的I/O才能使二级索引更新。Change Buffer会缓存这个更新，当相关页面不在Buffer Pool时，磁盘上的相关页面不会立即被读避免了昂贵的I/O操作。

当页面加载到缓冲池中，将合并缓冲的更改，稍后将更新的页面刷新到磁盘。

该InnoDB主线程在服务器几乎空闲时以及在慢速关闭期间合并缓冲的更改。

# 主从同步

## 流程

主库：

1. 创建binlog，用于后续同步。

2. 进行授权grant replication slave on \*.\* to 'slave'@'172.16.1.5' identified by '123456';

3. 确定权限生效，flush privileges。

从库：

1. 执行之前的备份文件

2. change master to master\_host="ip", master\_port="port", master\_user="user", master\_password="123456" master\_log\_file="mybinlog.00001", master\_log\_pos=475;

## 原理

主库：

需要开启binlog，并完成用户授权（享有replication slave权限），并创建log dump线程，用于发送binlog中的内容。在读取binlog中的操作时，此线程会对主节点上的binlog加锁，当读取完成，甚至在发动给从节点之前，锁会被释放。

从库，有两个主要线程：

IO进程： 负责从主服务器上读取二进制日志，并写入从服务器的中继日志（Relay log）

SQL进程： 负责读取中继日志，解析出主服务器已经执行的数据更改并在从服务器中重放（Replay）

# Trick

## 分段查询

select c.Cno,c.Cname,

count(case when s.score between 85 and 100 then 1 end) as '[100-85分]',

count(case when s.score between 70 and 85 then 1 end) as '[85-70分]',

count(case when s.score between 60 and 70 then 1 end) as '[70-60分]',

count(case when s.score<60 then 1 end) as '[60分以下]' from Course c, SC s where c.Cno = s.Cno

group by c.Cno,c.Cname

## 远程连接

1. MySQL的配置文件（14.14）是/etc/mysql/mysql.conf.d/mysqld.cnf，需要将bind-address=127.0.0.1注释。

2. 授予某个远程连接用户权限，并刷新

grant all on \*.\* to 'twd'@'%' identified by '123456';

flush privileges;

# 权限

grant all on \*.\* to 'twd'@'%' identified by '123456' with grant option

授予twd用户所有的权限，并且使其也具有授权这一功能（with grant option）。通常该语句用来远程访问MySQL。

# 问题

## truncate、drop和delete

drop：drop table table\_name;

丢弃表，直接将表删除掉。

truncate：truncate table table\_name;

删除表中的数据，再插入数据时自增长id又从1开始，在清空表中的数据使用

delete：delete from table where column = val;

删除某一行的数据，如果不加where，和truncate类似。

注意drop和truncate属于DDL（数据定义语言），而delete属于DML（数据库操作语言）

执行速度上：drop > truncate > delete

## DDL和DML

**DDL**：数据定义语句（Date Definition Language），操作立即生效，原数据不放到rollback segment中，不能回滚，不能触发trigger。

**DML**：数据库操作语言（Data Manipulation Language）的缩写，是指对数据库中表记录的操作，主要包括记录的插入insert、更新update、删除delete和查询select。

DML只是对表内数据的操作，而不涉及到表的定义、结构的修改，更不会涉及到其他对象。DDL语句更多的被数据库管理员所使用，一般开发人员很少使用。

## 插入多条数据

速度比较：

单条数据插入 < 事务内使用多条插入语句 < 合并数据为单条插入