咕泡学院 JAVA VIP 高级课程教案

性能优化之 MYSQL

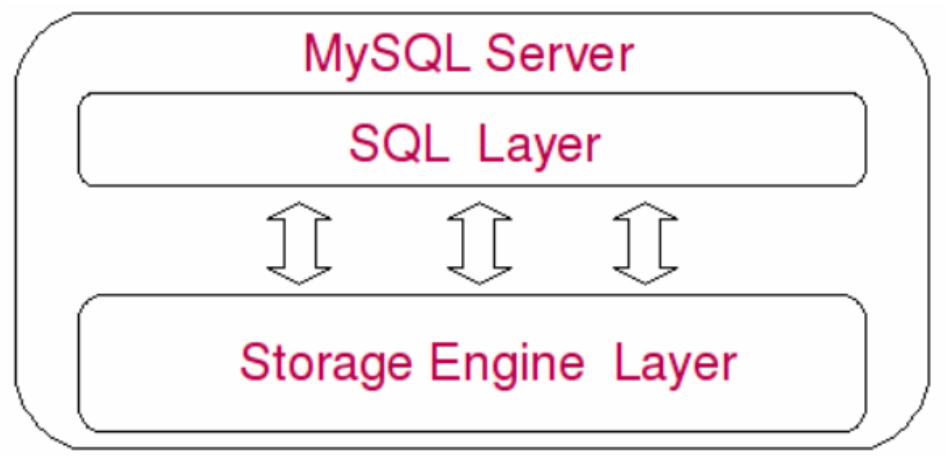
主讲	James QQ 2904270631
版本	V1.0
适用对象	咕泡学院 JAVA 高级 VIP 学员

课程安排

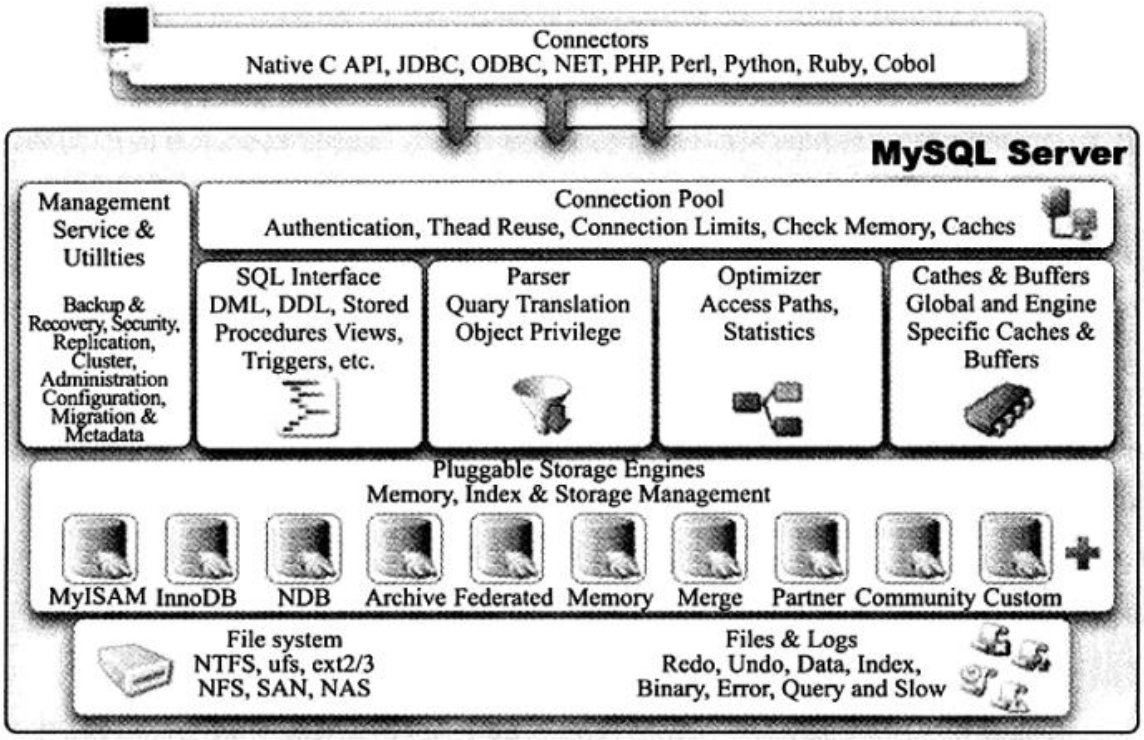
1. 认识 MYSQL

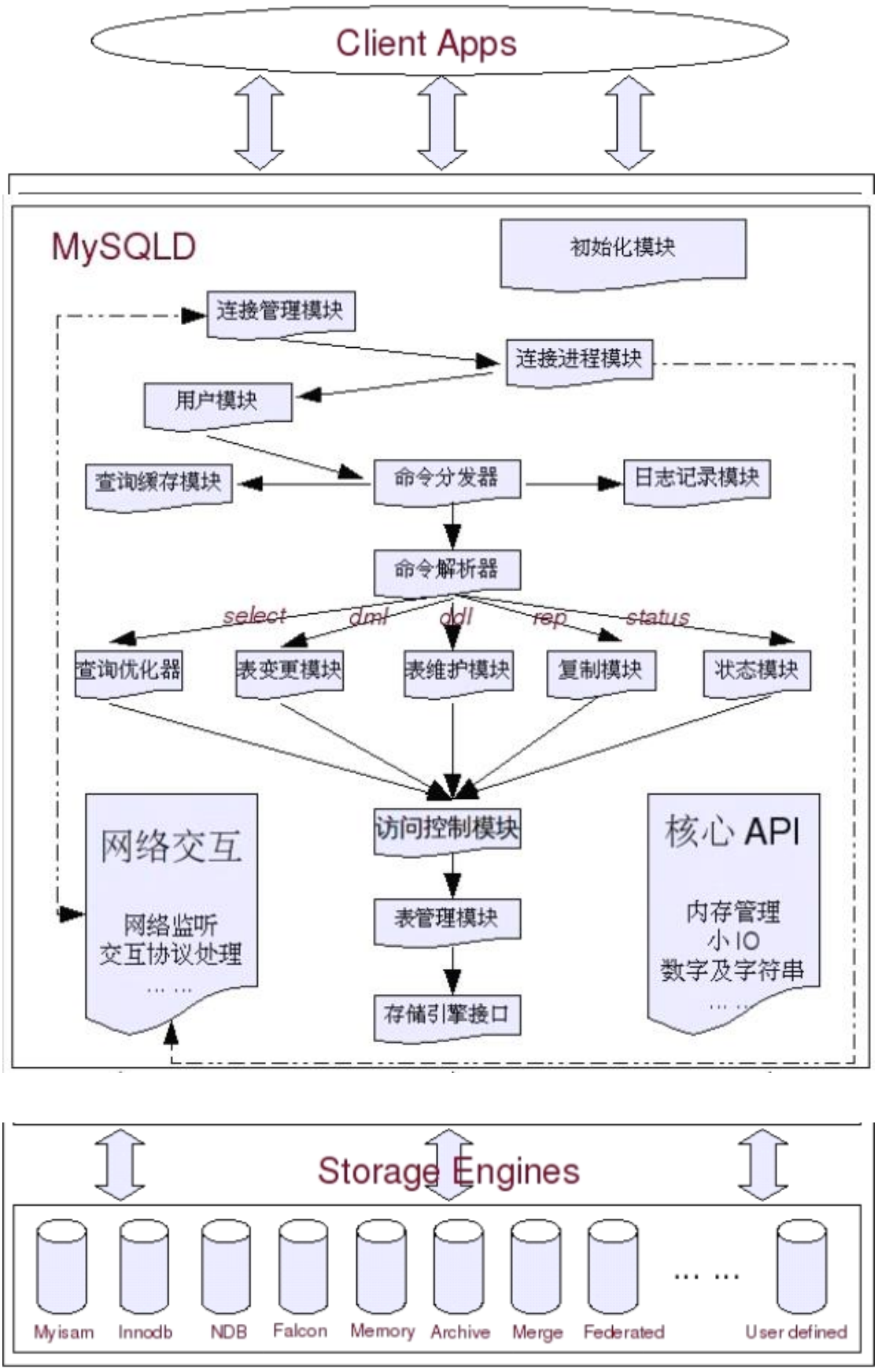
1985 年，瑞典的几位志同道合小伙子（以 David Axmark 为首） 成立了一家公司，这就是 MySQL AB 的前身。这个公司最初并不是为了开发数据库产品，而是在实现他们想法的过程中，需要一个数据库。他们希望能够使用开源的产品。但在当时并没有一个合适的选择，没办法，那就自己开发吧。

1.1 架构图



1.2 Detail


Connectors
Native C API, JDBC, ODBC, NET, PHP, Perl, Python, Ruby, Cobol
MySQL Server
Management Service & Utillties
Backup & Recovery, Security, Replication, Cluster, Administration Configuration, Migration & Metadata
Connection Pool
Authentication, Thead Reuse, Connection Limits, Check Memory, Caches
SQL Interface
DML, DDL, Stored Procedures Views, Triggers, etc.
Parser
Quary Translation
Object Privilege
Optimizer
Access Paths, Statistics
Cathes & Buffers
Global and Engine Specific Caches & Buffers
Pluggable Storage Engines
Memory, Index & Storage Management
MyISAM
InnoDB
NDB
Archive
Federated
Memory
Merge
Partner
Community
Custom
File system
NTFS, ufs, ext2/3
NFS, SAN, NAS
Files & Logs
Redo, Undo, Data, Index,
Binary, Error, Query and Slow


Client Apps
MySQLD
初始化模块
连接管理模块
连接进程模块
用户模块
查询缓存模块
命令分发器
日志记录模块
命令解析器
select
dml
ddl
rep
status
查询优化器
表变更模块
表维护模块
复制模块
状态模块
网络交互
网络监听
交互协议处理
… …
访问控制模块
表管理模块
存储引擎接口
核心 API
内存管理
小 IO
数字及字符串
… …
Storage Engines
Myisam
Innodb
NDB
Falcon
Memory
Archive
Merge
Federated
… …
User defined

引擎介绍

	Innodb	Myisam
存储文件	.frm 表定义文件 .ibd 数据文件	.frm 表定义文件 .myd 数据文件 .myi 索引文件
锁	表锁、行锁	表锁
事务	ACID	不支持
CRDU	读、写	读多
count	扫表	专门存储的地方
索引结构	B+ Tree	B+ Tree

2. 性能

2.1 影响性能的因素

2.1.1 人为因素-需求

2.1.1.1 count(*) 论坛 实时、准实时、有误差

2.1.2 程序员因素 – 面向对象

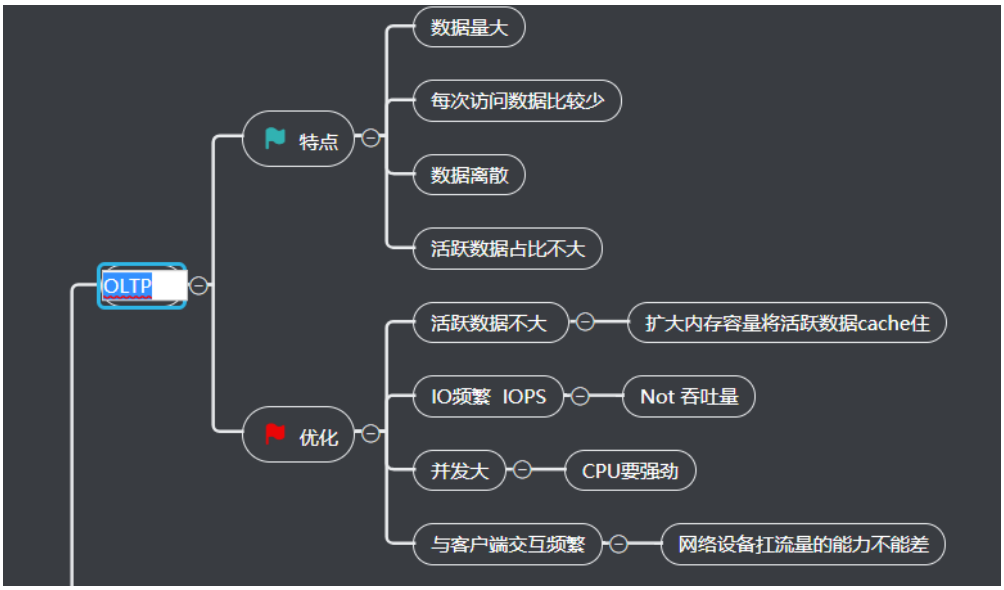
2.1.3 Cache

2.1.4 对可扩展过度追求

2.1.5 表范式

2.1.6 应用场景

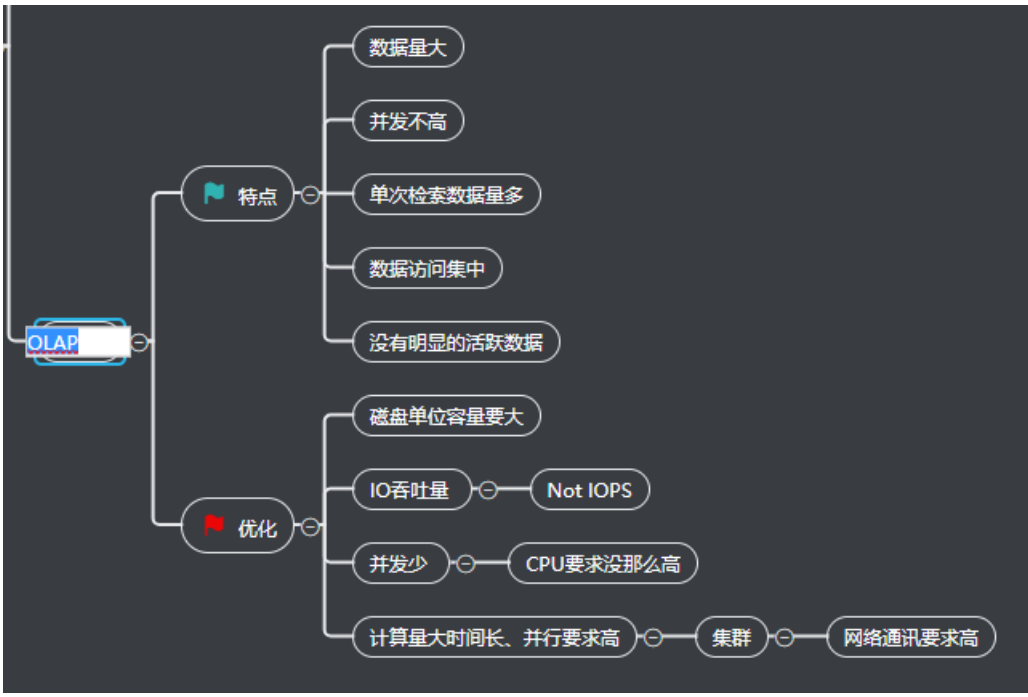
2.1.6.1 OLTP On-Line Transaction Processioning



2.1.6.2

2.1.6.3 IO per second

2.1.6.4 OLAP On-Line Analysis Processing



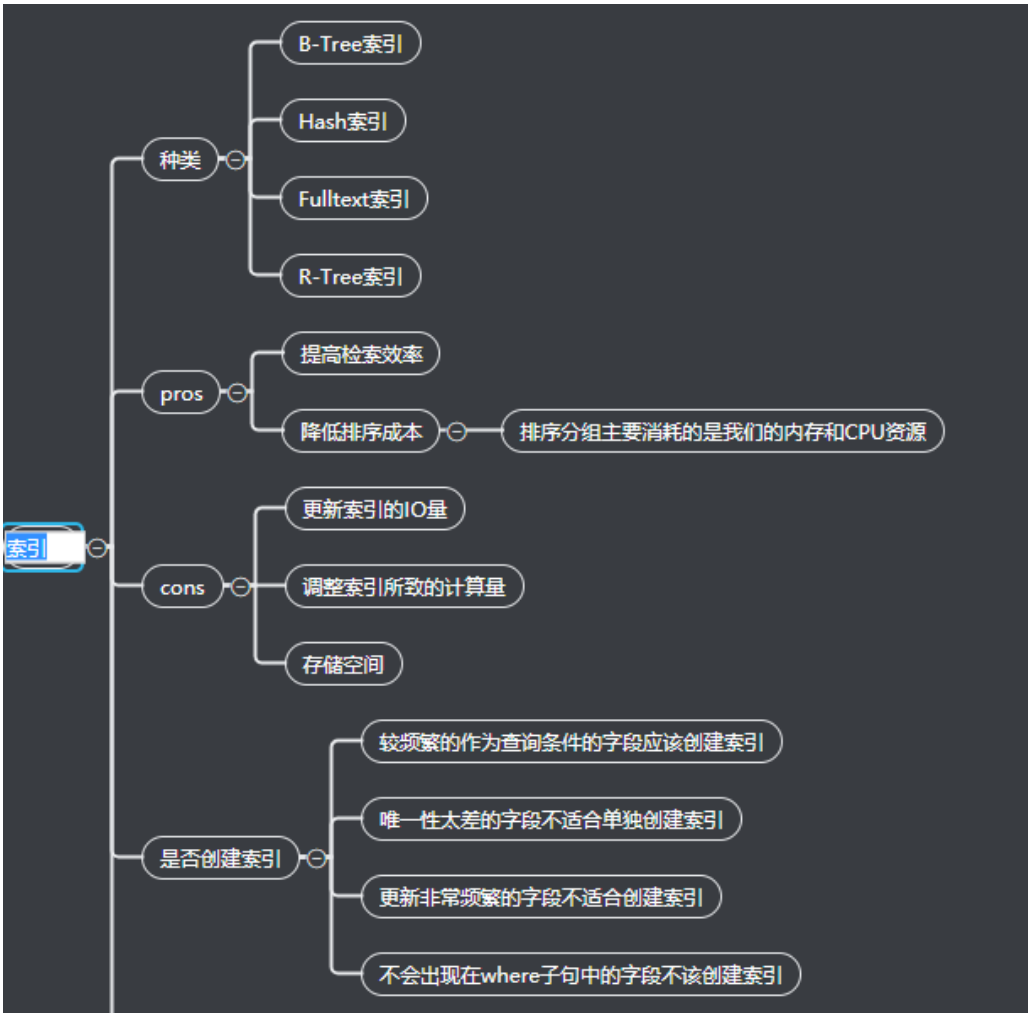
2.1.6.5

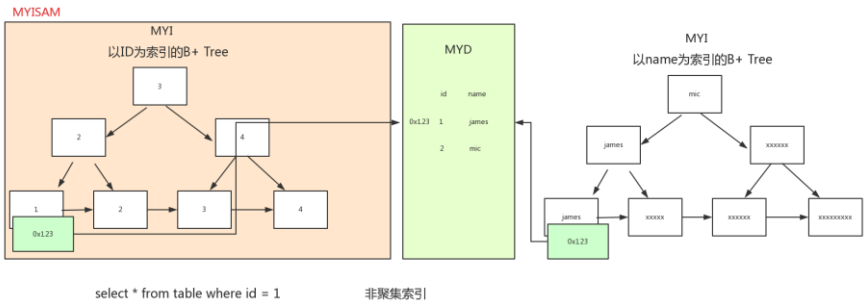
2.2 提高性能

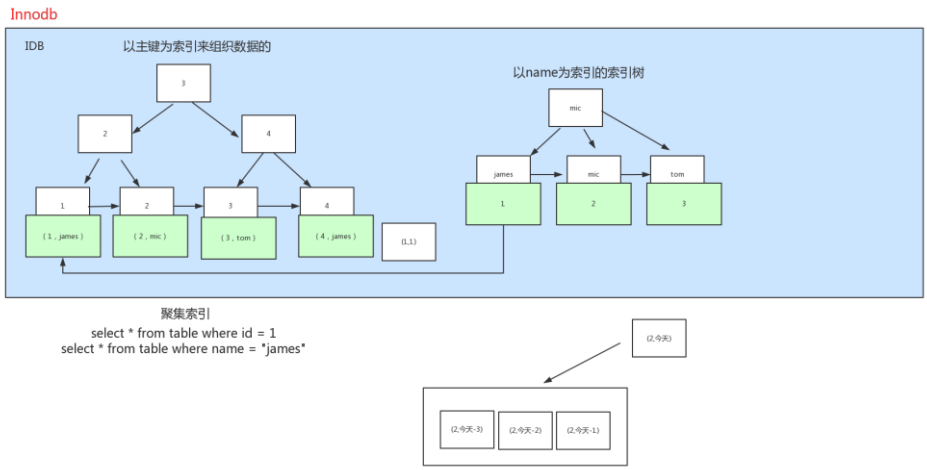
2.2.1 索引

2.2.1.1 索引结构

2.2.1.2







2.2.1.3

2.2.1.4 https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

```
评论表
id  user_id  created_time
1     2
2
3
4
5
6
7
(user_id,created_time)
select * from table where  user_id = 2 order by created_time ;
```

2.2.1.5

2.2.2 锁

行锁


pros
粒度小
cons
获取、释放所做的工作更多
容易发生死锁
row-level
共享锁
排他锁
通过在指向数据记录的第一个索引键之前和最后一个索引键之后的空域空间上标记锁定信息来实现的
间隙锁
实现 Innodb
尽可能让所有的数据检索都通过索引来完成
合理设计索引
锁优化
减少基于范围的数据是检索过滤条件
尽量控制事务的大小
业务允许的情况下，尽量使用较低级别的事务隔离

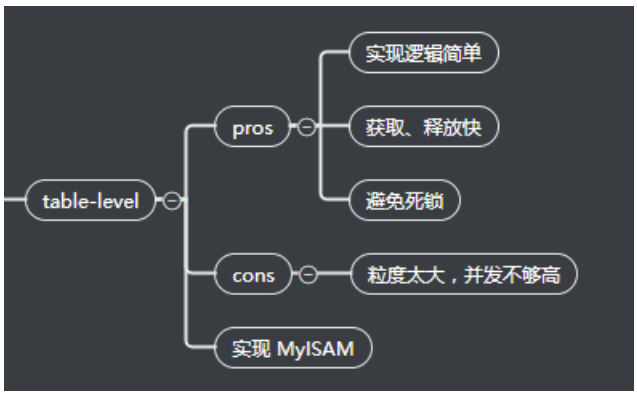
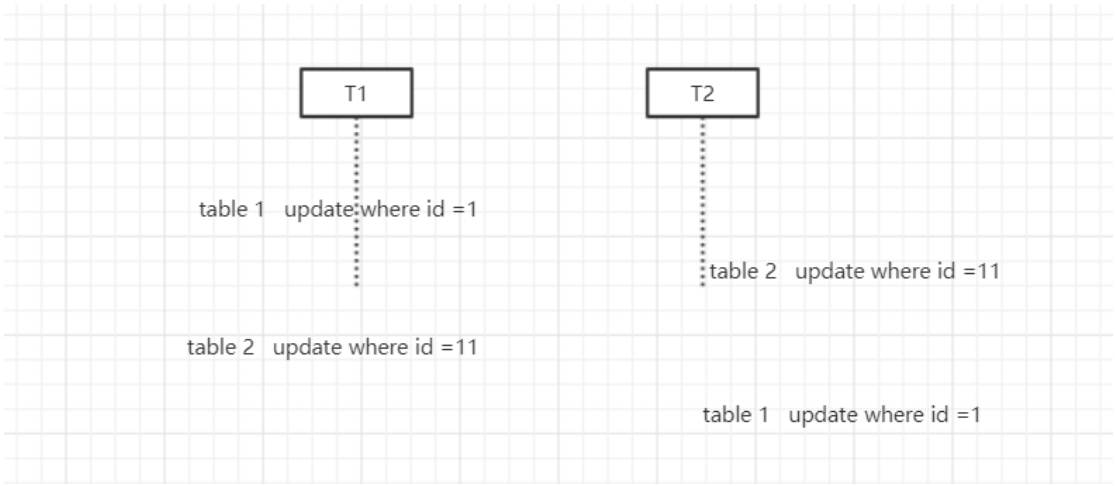


table-level
pros
实现逻辑简单
获取、释放快
避免死锁
cons
粒度太大，并发不够高
实现 MyISAM


page-level
介于 row和table之间
实现 BerkeleyDB


T1
T2
table 1 update where id =1
table 2 update where id =11
table 2 update where id =11
table 1 update where id =1

-- 表级锁的争用状态变量

show status like 'table%';

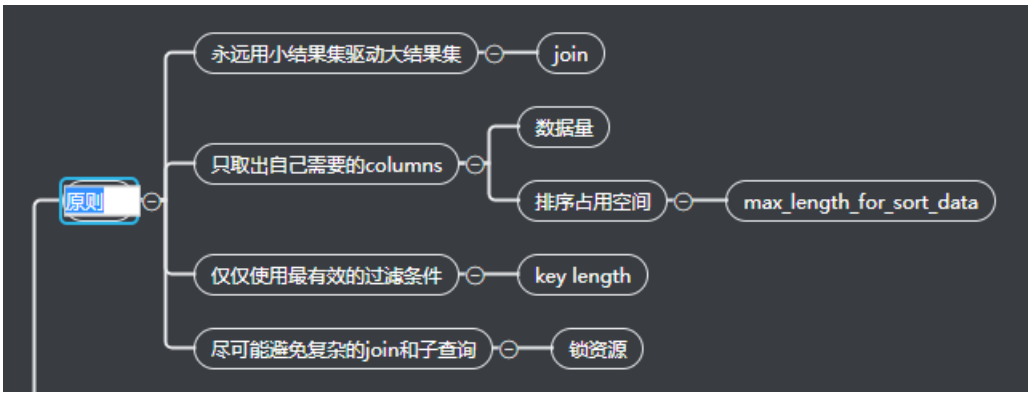
-- 行级锁争用状态变量

show status like 'innodb_row_lock%';

	共享锁（S）	排他锁（X）	意向共享锁（IS）	意向排他锁（IX）
共享锁（S）	兼容	冲突	兼容	冲突
排他锁（X）	冲突	冲突	冲突	冲突
意向共享锁（IS）	兼容	冲突	兼容	兼容
意向排他锁（IX）	冲突	冲突	兼容	兼容

2.2.3 优化

2.2.3.1



2.2.3.2 QEP Query Execution Plan

2.2.3.2.1 使用 加上 explain

我们先看一下在 MySQL Explain 功能中给我们展示的各种信息的解释：

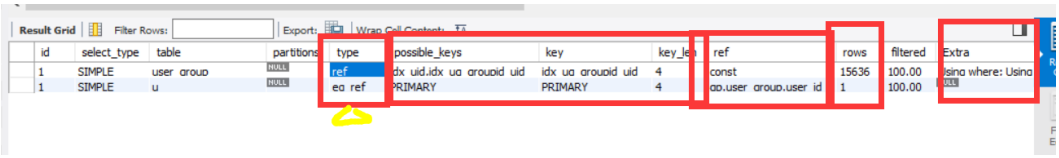
- ID：Query Optimizer 所选定的执行计划中查询的序列号；
- Select_type：所使用的查询类型，主要有以下这几种查询类型
  - DEPENDENT SUBQUERY：子查询中内层的第一个 SELECT，依赖于外部查询的结果集；
  - DEPENDENT UNION：子查询中的 UNION，且为 UNION 中从第二个 SELECT 开始的后面所有

SELECT，同样依赖于外部查询的结果集；

- PRIMARY：子查询中的最外层查询，注意并不是主键查询；
- SIMPLE：除子查询或者 UNION 之外的其他查询；
- SUBQUERY：子查询内层查询的第一个 SELECT，结果不依赖于外部查询结果集；
- UNCACHEABLE SUBQUERY：结果集无法缓存的子查询；
- UNION：UNION 语句中第二个 SELECT 开始的后面所有 SELECT，第一个 SELECT 为 PRIMARY
- UNION RESULT：UNION 中的合并结果；

- Table：显示这一步所访问的数据库中的表的名称；
- Type：告诉我们对表所使用的访问方式，主要包含如下集中类型；
  - all：全表扫描
  - const：读常量，且最多只会有一条记录匹配，由于是常量，所以实际上只需要读一次；
  - eq_ref：最多只会有一条匹配结果，一般是通过主键或者唯一键索引来访问；
  - fulltext：
  - index：全索引扫描；
  - index_merge：查询中同时使用两个（或更多）索引，然后对索引结果进行 merge 之后再读取表数据；
  - index_subquery：子查询中的返回结果字段组合是一个索引（或索引组合），但不是一个主键或者唯一索引；
  - rang：索引范围扫描；
  - ref：Join 语句中被驱动表索引引用查询；
  - ref_or_null：与 ref 的唯一区别就是在使用索引引用查询之外再增加一个空值的查询；
  - system：系统表，表中只有一行数据；
  - unique_subquery：子查询中的返回结果字段组合是主键或者唯一约束；
  -

依次从好到差：system，const，eq_ref，ref，fulltext，ref_or_null，unique_subquery，index_subquery，range，index_merge，index，ALL



- Possible_keys：该查询可以利用的索引. 如果没有任何索引可以使用，就会显示成null，这一项内容对于优化时候索引的调整非常重要；
- Key：MySQL Query Optimizer 从 possible_keys 中所选择使用的索引；
- Key_len：被选中使用索引的索引键长度；
- Ref：列出是通过常量（const），还是某个表的某个字段（如果是 join）来过滤（通过 key）的；
- Rows：MySQL Query Optimizer 通过系统收集到的统计信息估算出来的结果集记录条数；

- Extra：查询中每一步实现的额外细节信息，主要可能会是以下内容：
  - Distinct：查找 distinct 值，所以当 mysql 找到了第一条匹配的结果后，将停止该值的查询而转为后面其他值的查询；
  - Full scan on NULL key：子查询中的一种优化方式，主要在遇到无法通过索引访问 null 值的使用使用；
  - Impossible WHERE noticed after reading const tables：MySQL Query Optimizer 通过收集到的统计信息判断出不可能存在结果；
  - No tables：Query 语句中使用 FROM DUAL 或者不包含任何 FROM 子句；
  - Not exists：在某些左连接中 MySQL Query Optimizer 所通过改变原有 Query 的组成而使用的优化方法，可以部分减少数据访问次数；

- Range checked for each record (index map: N)：通过 MySQL 官方手册的描述，当 MySQL Query Optimizer 没有发现好的可以使用的索引的时候，如果发现如果来自前面的表的列值已知，可能部分索引可以使用。对前面的表的每个行组合，MySQL 检查是否可以使用 range 或 index_merge 访问方法来索取行。
- Select tables optimized away：当我们使用某些聚合函数来访问存在索引的某个字段的时候，MySQL Query Optimizer 会通过索引而直接一次定位到所需的数据行完成整个查询。当然，前提是在 Query 中不能有 GROUP BY 操作。如使用 MIN()或者 MAX（）的时候；
- Using filesort：当我们的 Query 中包含 ORDER BY 操作，而且无法利用索引完成排序操作的时候，MySQL Query Optimizer 不得不选择相应的排序算法来实现。

- ◇ Using index：所需要的数据只需要在 Index 即可全部获得而不需要再到表中取数据；
- ◇ Using index for group-by：数据访问和 Using index 一样，所需数据只需要读取索引即可，而当 Query 中使用了 GROUP BY 或者 DISTINCT 子句的时候，如果分组字段也在索引中，Extra 中的信息就会是 Using index for group-by；
- ◇ Using temporary：当 MySQL 在某些操作中必须使用临时表的时候，在 Extra 信息中就会出现 Using temporary 。主要常见于 GROUP BY 和 ORDER BY 等操作中。
- ◇ Using where：如果我们不是读取表的所有数据，或者不是仅仅通过索引就可以获取所有需要的数据，则会出现 Using where 信息；
- ◇ Using where with pushed condition：这是一个仅仅在 NDBCluster 存储引擎中才会出现的信息，而且还需要通过打开 Condition Pushdown 优化功能才可能会被使用。控制参数为 engine_condition_pushdown 。

2.2.3.3 Profiling (有个概念)

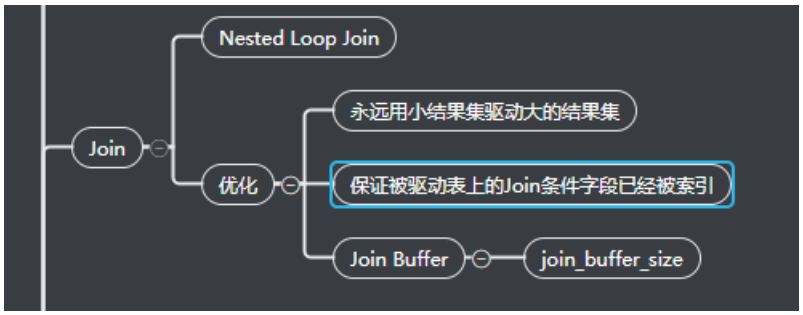
set profiling=1;

select nick_name ,count(*) from user group by nick_name;show profiles;

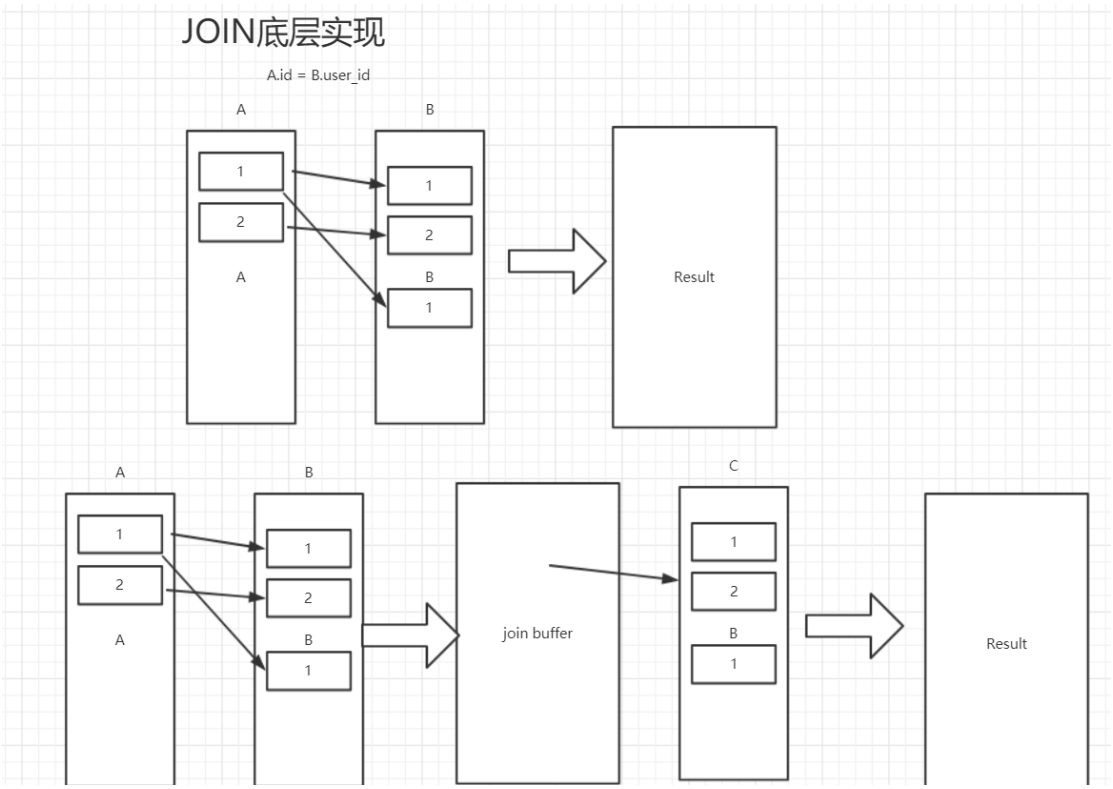
show profile cpu,block io for query 75;

2.2.4 join \ order by \group by 解释

2.2.4.1 Join

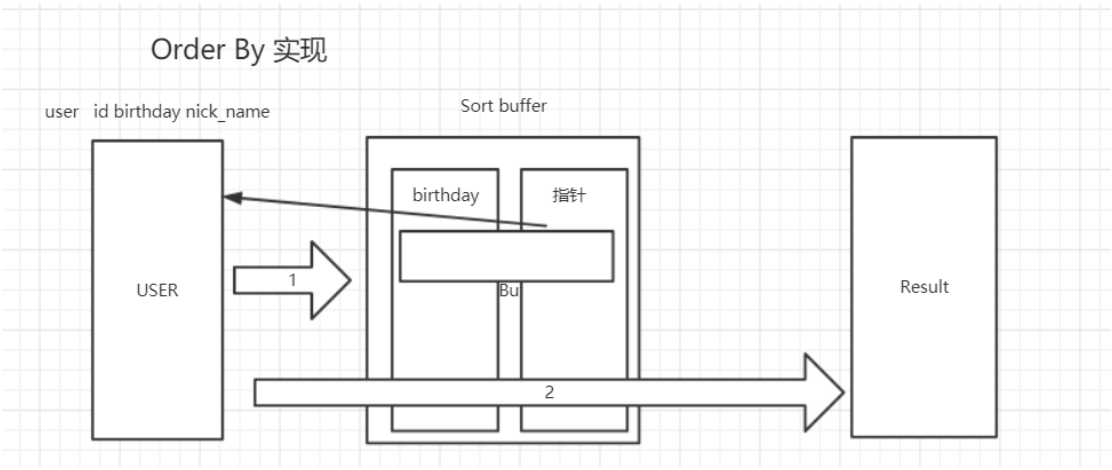


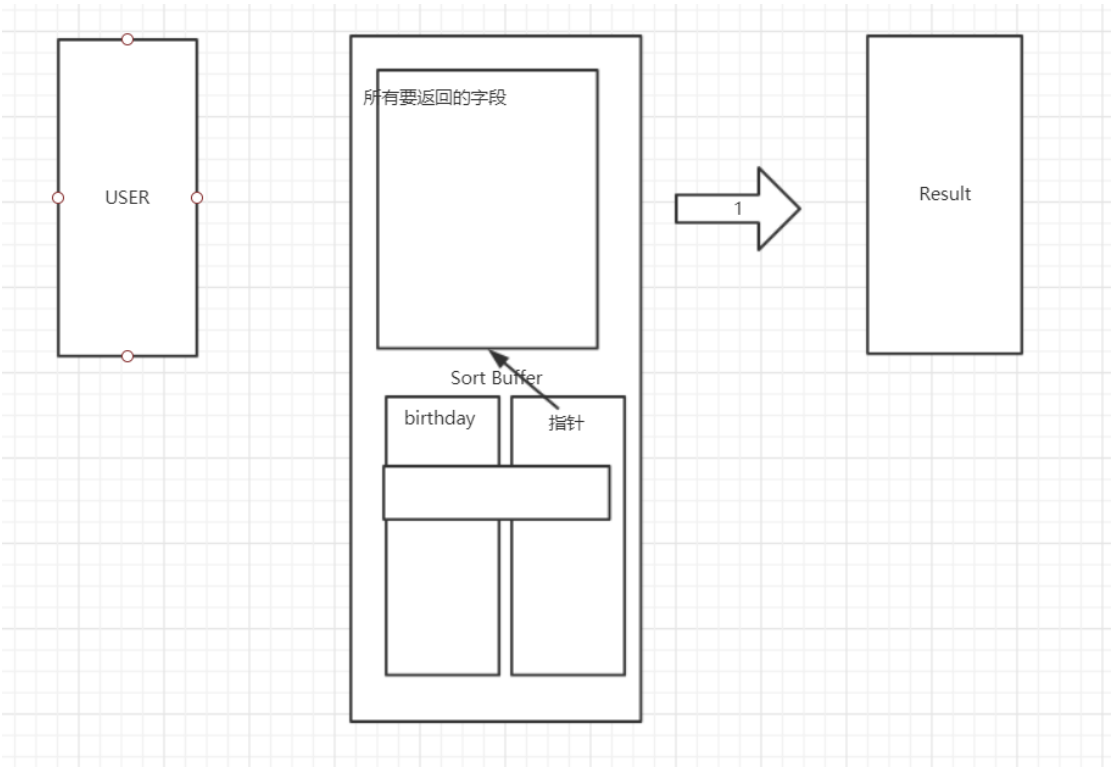
show variables like 'join_%';


JOIN底层实现
A.id = B.user_id
A
B
1
2
A
1
2
B
1
Result
A
B
C
1
2
A
1
2
B
1
join buffer
1
2
B
1
Result

order by


order by
实现
有序
排序字段和指针在Sort buffer排序 --》然后用指针去取数据
无序
排序字段和所有数据全部取出 --》排序字段+指针 Sort Buffer排序(其它数据放到内存中) --》指针到内存里去取数据然后返回
节省IO
耗内存
空间换时间
优化
索引顺序一致的话不需要再排序
加大 max_length_for_sort_data 从而使用第二种排序方法（排序只针对需要排序的字段）
内存不充足时 去掉不必要的返回字段
增大 sort_buffer_size
减少在排序过程中对需要排序的数据进行分段





Group by

DISTINCT

LIMIT

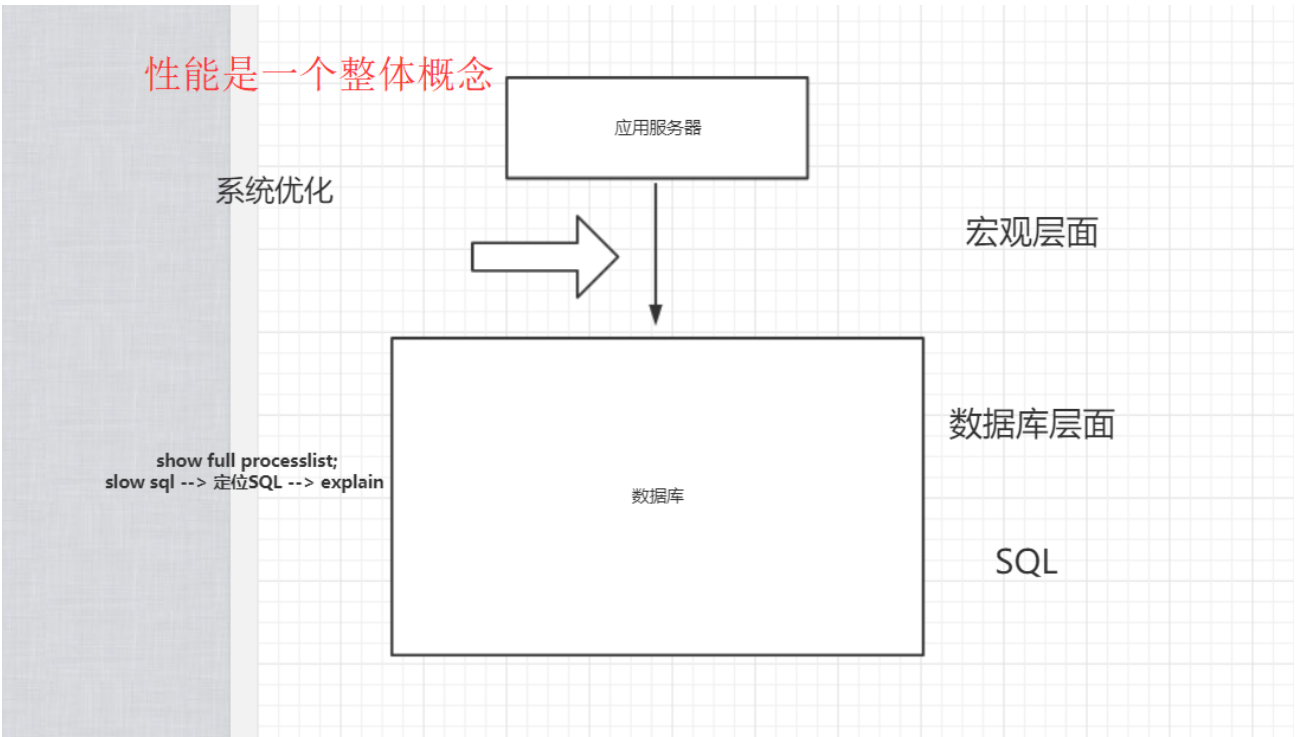
SELECT * FROM user limit 10000,10;

取 10010 条数据

SELECT * FROM user where id> 10000 limit 10;

Slow Sql 配置

```
[mysqld]
slow_query_log=1
slow_query_log_file=/path/to/file
long_query_time=0.2
log_output=FILE
```



有问题提到 http://git.gupaoedu.com/vip/nice/issues/18