



# MySQL架构与优化

---

金蝶云苍穹云开发平台部

2020-09

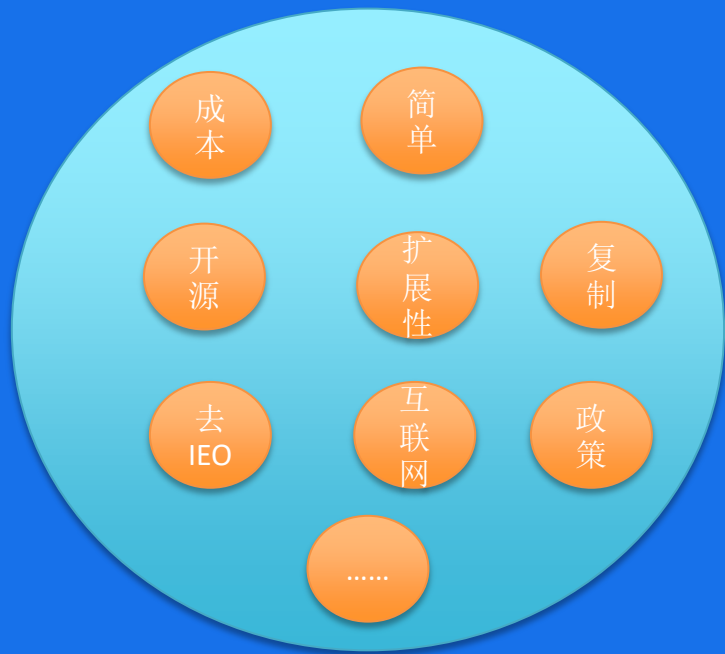


01 / MySQL体系架构

02 / MySQL架构设计

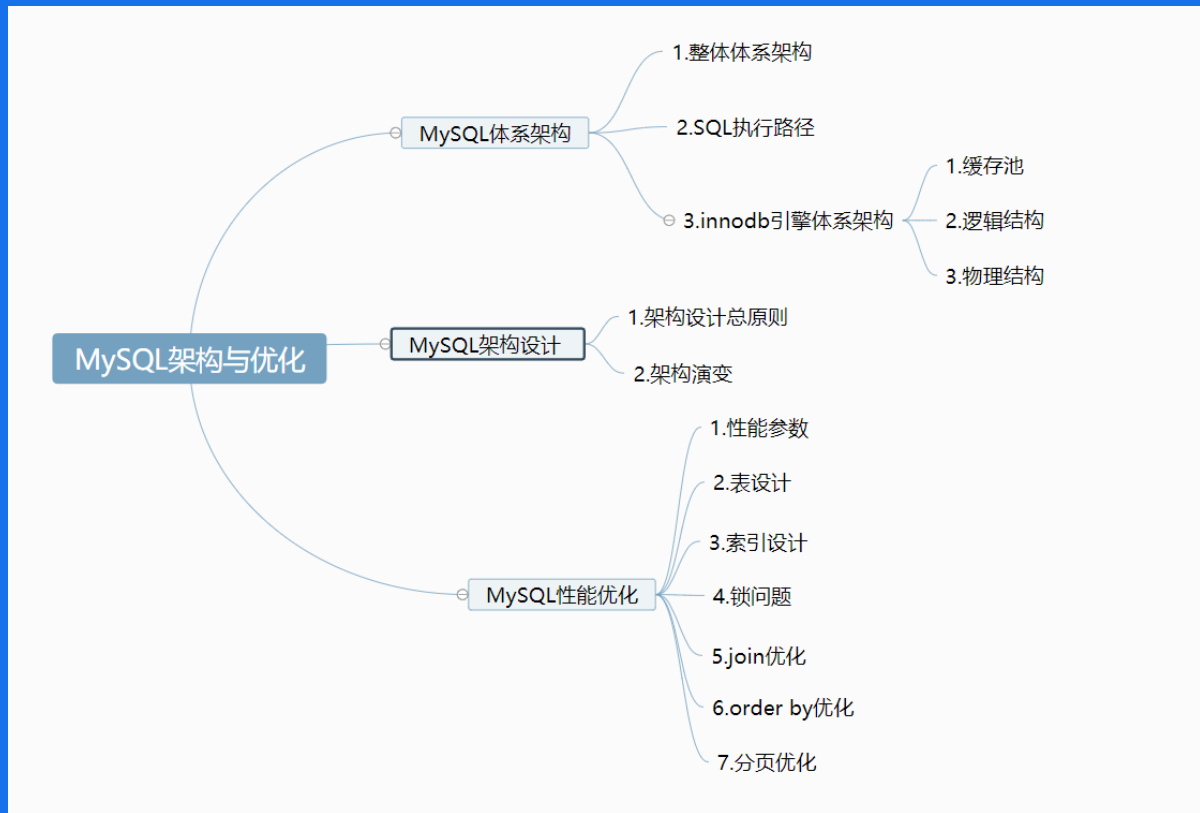
03 / MySQL性能优化

## 阿里为什么会把Oracle替换为MySQL?



1. Oracle RAC存储容量和性能无法横向扩展
2. 开源能够自己掌控，可以自己定制和优化
3. MySQL的binlog日志使其扩展性很强

## ■ 思维导图



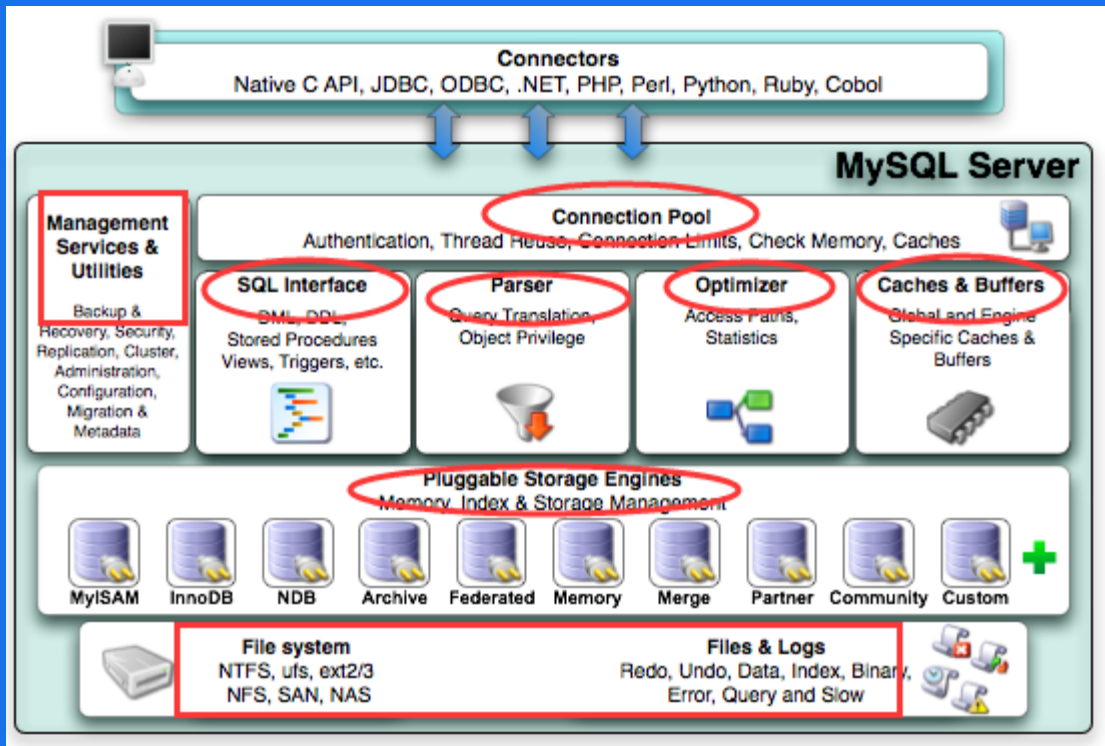
01 / MySQL体系架构

02 / MySQL架构设计

03 / MySQL性能优化

# MySQL体系架构

## 1.1 总体架构



连接层

Server层

引擎层

文件层

## ■ 1.1总体架构

### ➤ 连接池组件(Connection Pool)

负责监听对客户端向MySQL Server端的各种请求，接收请求，转发请求到目标模块。每个成功连接MySQL Server的客户请求都会被创建或分配一个线程，该线程负责客户端与MySQL Server端的通信，接收客户端发送的命令，传递服务端的结果信息等

### ➤ SQL接口组件(SQL Interface)

接收用户SQL命令，如DML,DDL和存储过程等，并将最终结果返回给用户

### ➤ 查询分析器组件(Parser)

首先分析SQL命令语法的合法性，并尝试将SQL命令分解成数据结构，若分解失败，则提示SQL语句不合理。

# MySQL体系架构

## ■ 1.1总体架构

### ➤ 优化器组件 (Optimizer)

对SQL命令按照标准流程进行优化分析

### ➤ 缓存主件 (Caches & Buffers)

缓存和缓冲组件

### ➤ 管理服务组件和工具组件(Management Service & Utilities)

提供对MySQL的集成管理，如备份(Backup),恢复(Recovery),安全管理(Security)等

### ➤ MySQL存储引擎

存储引擎说白了就是如何管理操作数据（存储数据、如何更新、查询数据等）的一种方法



# MySQL体系架构

## ■ 1.2一条SQL查询语句执行路径

select \* from T where ID=10;

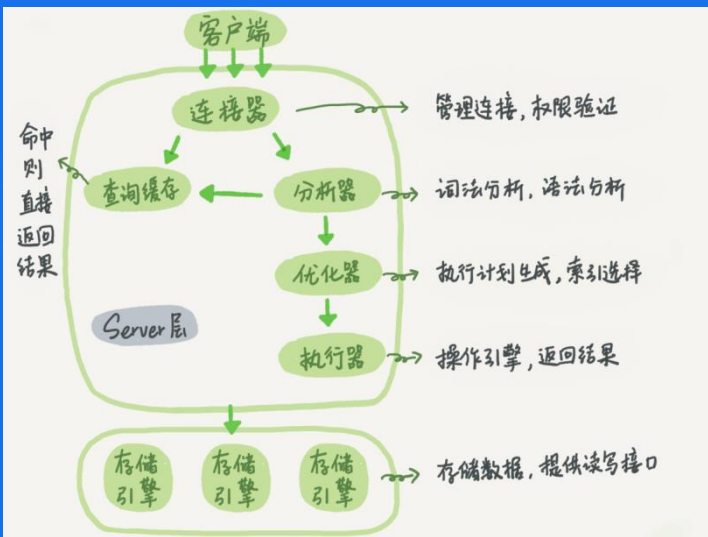
“Access denied for user” 是那一层返回的错误?

“You have an error in your SQL syntax” 是那一层返回的错误?

“SELECT command denied to user ‘b’@‘localhost’ for table ‘T’ 是那一层范围的错误?

“Unknown column ‘k’ in ‘where clause’” 是那一层返回的错误?

- 1.连接器
- 2.分析器
- 3.执行器
- 4.分析器



# MySQL体系架构

## ■ 1.2一条SQL更新语句执行过程

update T set c=c+1 where ID=2;

图中浅色框表示是在 InnoDB 内部执行的，深色框表示是在执行器中执行的。

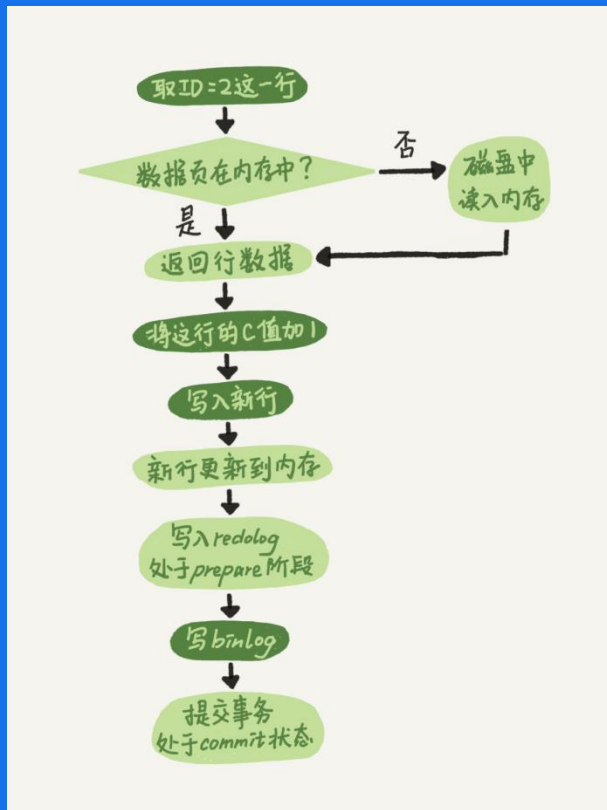
redolog+binlog

WAL(write-ahead logging)

redo log 的写入拆成了两个步骤：

prepare 和 commit，这就是"两阶段提交"

事务=更新内存+写redolog+写binlog



## ■ 1.2 SQL语句执行顺序

- ( 1 ) ( 8 ) SELECT
- ( 2 ) ( 9 ) DISTINCT
- ( 3 ) ( 1 ) FROM [ left\_table ]--[入口]
- ( 4 ) ( 3 ) < join\_type > JOIN < right\_table >
- ( 5 ) ( 2 ) ON < join\_condition >
- ( 6 ) ( 4 ) WHERE < where\_condition >
- ( 7 ) ( 5 ) GROUP BY < group\_by\_list >
- ( 8 ) ( 6 ) WITH < CUBE | RollUP >
- ( 9 ) ( 7 ) HAVING < having\_condition >
- ( 10 ) ( 10 ) ORDER BY < order\_by\_list >
- ( 11 ) ( 11 ) LIMIT M,N

# MySQL体系架构

## ■ 1.3MySQL存储引擎

```
mysql> show engines;
```

Engine	Support	Comment	Transactions	XA	Savepoints
InnoDB	DEFAULT	Supports transactions, row-level locking, and foreign keys	YES	YES	YES
CSV	YES	CSV storage engine	NO	NO	NO
MyISAM	YES	MyISAM storage engine	NO	NO	NO
BLACKHOLE	YES	/dev/null storage engine (anything you write to it disappears)	NO	NO	NO
PERFORMANCE_SCHEMA	YES	Performance schema	NO	NO	NO
MRG_MYISAM	YES	Collection of identical MyISAM tables	NO	NO	NO
ARCHIVE	YES	Archive storage engine	NO	NO	NO
MEMORY	YES	Hash based, stored in memory, useful for temporary tables	NO	NO	NO
FEDERATED	NO	Federated MySQL storage engine	NULL	NULL	NULL

0 rows in set (0.00 sec)

```
mysql> show create table test_innodb\G
***** 1. row *****
Table: test_innodb
Create Table: CREATE TABLE `test_innodb` (
  `FID` bigint(20) NOT NULL DEFAULT '0',
  `FNUMBER` varchar(60) NOT NULL DEFAULT '',
  PRIMARY KEY (`FID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
1 row in set (0.00 sec)

mysql> show create table test_myisam\G
***** 1. row *****
Table: test_myisam
Create Table: CREATE TABLE `test_myisam` (
  `FID` bigint(20) NOT NULL DEFAULT '0',
  `FNUMBER` varchar(60) NOT NULL DEFAULT '',
  PRIMARY KEY (`FID`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4
1 row in set (0.00 sec)

mysql> show create table test_memory\G
***** 1. row *****
Table: test_memory
Create Table: CREATE TABLE `test_memory` (
  `FID` bigint(20) NOT NULL DEFAULT '0',
  `FNUMBER` varchar(60) NOT NULL DEFAULT '',
  PRIMARY KEY (`FID`)
) ENGINE=MEMORY DEFAULT CHARSET=utf8mb4
1 row in set (0.00 sec)
```

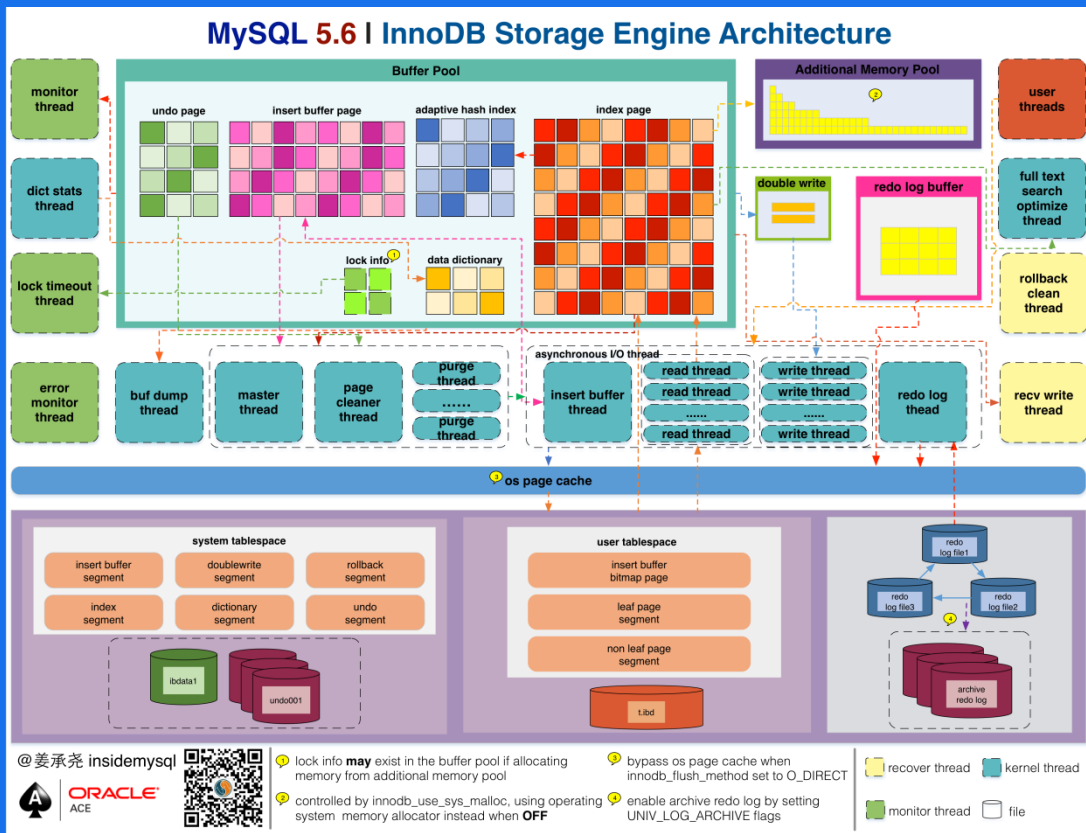
# MySQL体系架构

## ■ 1.3MySQL存储

特性	MyISAM	BDB	Memory	InnoDB	Archive	NDB
Storage Limits	NO	NO	YES	64TB	NO	YES
Transaction(commit rollback)		√		√		
Locking granularity	Table	Page	Table	Row	Row	Row
MVCC/Snapshot Read				√	√	√
Ceospacial Support	√					
B-Tree indexes	√	√	√	√		√
Hash Indexes			√	√		
Full Text search Index	√			√ (InnoDB1.2版本后支持)		
Clustered Index				√		
Data Caches			√	√		√
Index Caches	√		√	√		√
Compressed Data	√				√	
Encripted Data	√	√	√	√	√	√
Storage Cost	Low	Low	N/A	High	Very Low	Low
Memory Cost	Low	Low	Medium	Low	Very High	High
Bulk Insert Speed	High	High	High	Low	Very High	High
Cluster database Support				√		√
Replication Support	√	√	√	√	√	√
Foreign Key Support				√		
BackUp/Point in-time recovery	√	√	√	√	√	√
Update Statistic for data Dictionary	√	√	√	√	√	√

# MySQL体系架构

## 1.3 InnoDB引擎体系架构



# MySQL体系架构

## ■ 1.3InnoDB引擎体系架构

### 后台线程

#### ➤ Monitor thread

Lock timeout thread: 锁监控

Error monitor thread: 错误监控

#### ➤ Kernel thread

master thread: 负责刷新缓存数据到硬盘上

IO thread: 处理insert buffer, redolog日志, 读写请求等回调

purge thread: 回收undo页

page cleaner thread: 刷新脏页

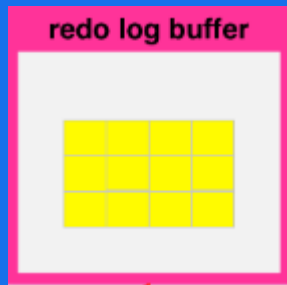
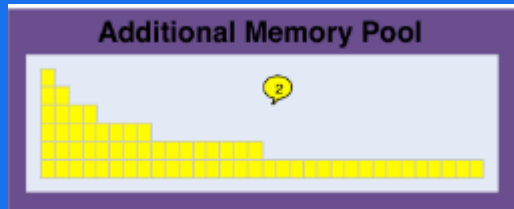
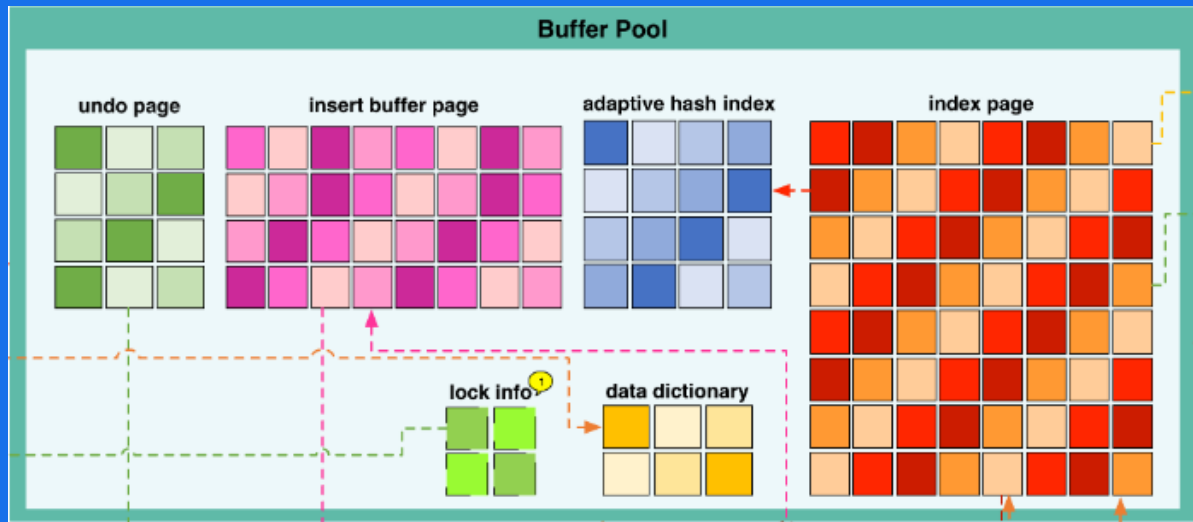
#### ➤ Recover thread

Rollback clean thread

Recv write thread

# MySQL体系架构

## ■ 1.3InnoDB引擎缓存结构





# MySQL体系架构

## ■ 1.3InnoDB引擎缓存结构

### ➤ 缓存池 (innodb\_buffer\_pool\_size参数控制)

缓冲池中缓存的数据页类型有：索引页、数据页、undo页、插入缓冲(insert buffer)、自适应哈希索引(adaptive hash index)、InnoDB存储的锁信息(lock info)和数据字典信息(data dictionary)

### ➤ 重做日志缓存 (innodb\_log\_buffer\_size参数控制)

Redo log 日志缓存，InnoDB存储引擎会首先将重做日志信息先放入重做日志缓冲中，然后再按照一定频率将其刷新到重做日志文件

### ➤ 额外内存缓存 (innodb\_additional\_mem\_pool\_size参数控制，5.7.4去除)

每个缓冲池中的帧缓冲 (frame buffer) 还有对应的缓冲控制对象 (buffer control block)，这些对象记录了诸如LRU、锁、等待等方面的信息，而这个对象的内存需要从额外内存池中申请

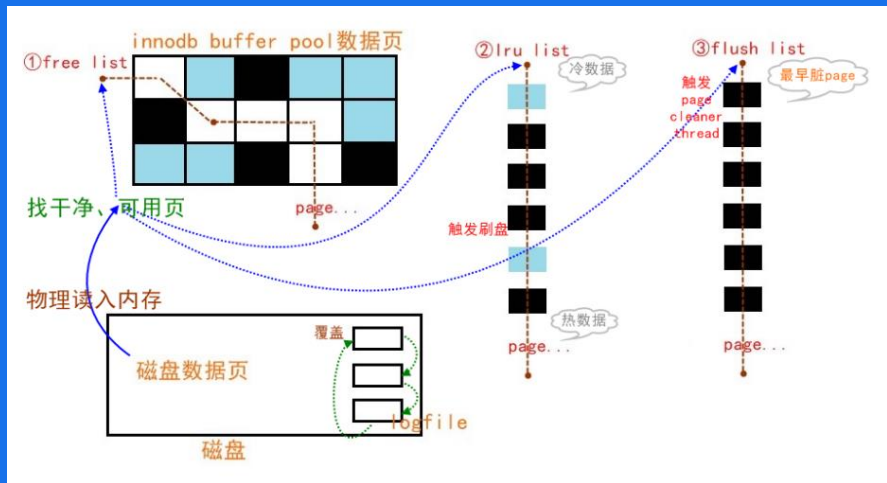
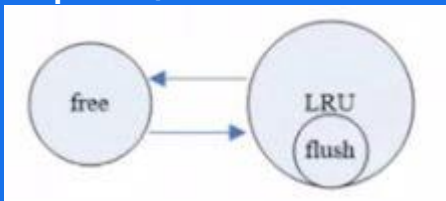
# MySQL体系架构

## ■ 1.3InnoDB引擎缓存池(buffer pool)内存结构

Free list: 空闲链表

Lru list: 最近最少使用链表

Flush list: 刷新链表



```
-----
BUFFER POOL AND MEMORY
Total large memory allocated 54971596800
Dictionary memory allocated 28991407
Buffer pool size 3276400
Free buffers 41894
Database pages 3222605
Old database pages 1189430
Modified db pages 0
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 1489849, not young 3372085
0.00 young/s, 0.00 non-young/s
Pages read 718679, created 7714668, written 11443111
0.00 reads/s, 0.00 creates/s, 0.55 writes/s
Buffer pool hit rate 1000 / 1000, young-making rate 0 / 1000 not 0 / 1000
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 3222605, unzip_LRU len: 0
I/o sum[1336]:cur[0], unzip sum[0]:cur[0]
-----
```

free list

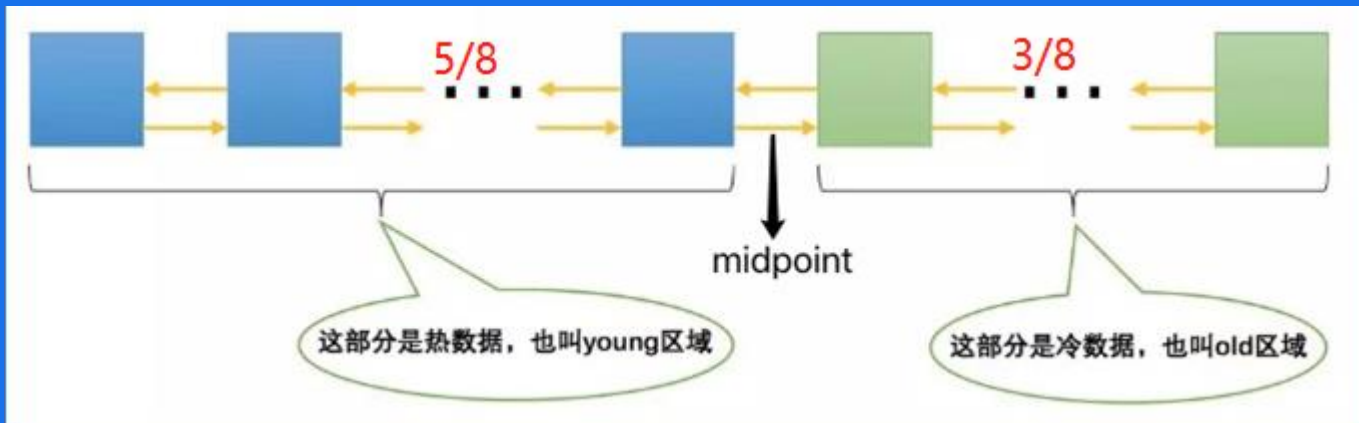
LRU list

缓存命中率

# MySQL体系架构

## ■ 1.3InnoDB引擎缓存池

### LRU链表

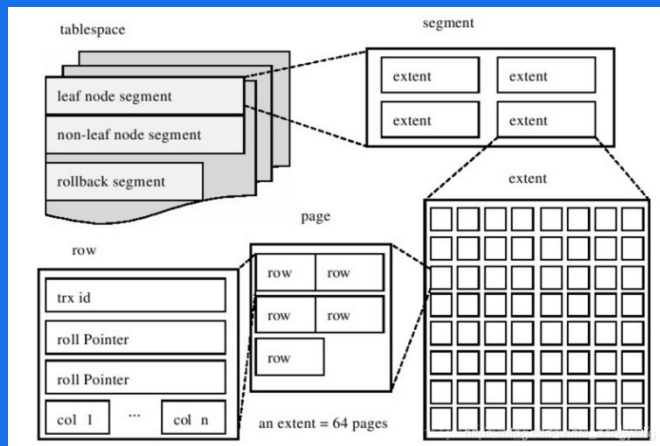


### 全表扫描缓存污染问题

`innodb_old_blocks_time=1000`

# MySQL体系架构

## ■ 1.3InnoDB引擎逻辑结构



表空间：InnoDB有共享表空间ibdata1,每张表有单独的表空间ibd，独立表空间存放数据和索引，共享表空间存undo信息，事务信息，二次写缓存等。

段：数据段，索引段，回滚段。索引段即为B+树的非索引节点。

区：连续页组成的空间，任何情况下大小都为1Mb，为保证连续性，InnoDB存储引擎一次申请4-5个区。一个区由64个连续的页组成。

页：磁盘管理的最小单位，默认16K。类型有：数据页，undo页，系统页① 绝密信息 严禁泄露 20

# MySQL体系架构

## 1.3 InnoDB引擎物理结构

```
root@centos7-65:~# ls -l /var/lib/mysql/
-rw-r--r-- 1 mysql mysql 56 Mar 12 2019 auto.cnf
-rw-r--r-- 1 mysql mysql 907224134 Sep 8 16:59 centos7-65.log
-rw-r--r-- 1 mysql mysql 63202459 Sep 7 16:17 centos7-65.log.bak
-rw-r--r-- 1 mysql mysql 367712343 Sep 8 10:06 centos7-65.log.bak0908
-rw-r--r-- 1 mysql mysql 226355973 Sep 8 11:23 centos7-65.log.bak2
-rw-r--r-- 1 mysql mysql 166926106 Sep 8 11:27 centos7-65.log.bak3
-rw-r--r-- 1 mysql mysql 39182 Aug 19 10:55 centos7-65.log.toddb
-rw-r--r-- 1 mysql mysql 51901 Aug 19 11:11 centos7-65.log.todfulltext1
drwxr-x-- 2 mysql mysql 53 Mar 17 08:38 cosmic_perf_stat
-rw-r--r-- 1 root root 28239 Sep 8 10:08 gen1.log
-rw-r--r-- 1 root root 23803 Sep 8 11:23 gen2.sql
-rw-r--r-- 1 root root 25216 Sep 8 12:04 gen3.sql
-rw-r--r-- 1 root root 29116 Sep 8 17:02 gen4.sql
-rw-r--r-- 1 root root 28125 Sep 7 16:11 gen.log
-rw-r--r-- 1 mysql mysql 10852280 Sep 10 20:03 ib_buffer_pool
-rw-r--r-- 1 mysql mysql 1849688064 Sep 11 09:46 ibdata1
-rw-r--r-- 1 mysql mysql 536870912 Sep 11 09:46 ib_logfile0
-rw-r--r-- 1 mysql mysql 536870912 Sep 11 09:46 ib_logfile1
-rw-r--r-- 1 mysql mysql 536870912 Sep 10 20:37 ib_logfile2
-rw-r--r-- 1 mysql mysql 79691776 Sep 11 09:46 ibtmp1
drwxr-x-- 2 mysql mysql 4096 Apr 8 2019 mysql
-rw-r--r-- 1 mysql mysql 7 Sep 10 20:04 mysql3307.pr
drwxr-x-- 2 mysql mysql 8192 Mar 12 2019 performance_schema
drwxr-x-- 2 mysql mysql 75 Sep 9 10:37 ptest_biz_baseline_aidb
drwxr-x-- 2 mysql mysql 4096 Sep 9 10:37 ptest_biz_baseline_bda
drwxr-x-- 2 mysql mysql 24576 Sep 9 10:37 ptest_biz_baseline_cr
drwxr-x-- 2 mysql mysql 8192 Sep 9 10:37 ptest_biz_baseline_de
drwxr-x-- 2 mysql mysql 57344 Sep 9 16:04 ptest_biz_baseline_drp
drwxr-x-- 2 mysql mysql 93 Oct 9 2019 ptest_biz_baseline_ec
drwxr-x-- 2 mysql mysql 32768 Sep 9 11:41 ptest_biz_baseline_eip
drwxr-x-- 2 mysql mysql 20480 Sep 9 10:37 ptest_biz_baseline_epm
drwxr-x-- 2 mysql mysql 245760 Sep 10 21:21 ptest_biz_baseline_fi
drwxr-x-- 2 mysql mysql 196608 Sep 9 11:39 ptest_biz_baseline_hr
drwxr-x-- 2 mysql mysql 4096 Sep 9 10:37 ptest_biz_baseline_hssp
drwxr-x-- 2 mysql mysql 19 Nov 26 2019 ptest_biz_baseline_hwsc
drwxr-x-- 2 mysql mysql 4096 Sep 9 11:05 ptest_biz_baseline_log
drwxr-x-- 2 mysql mysql 19 Sep 9 10:37 ptest_biz_baseline_pmc
drwxr-x-- 2 mysql mysql 192512 Sep 9 11:27 ptest_biz_baseline_scm
drwxr-x-- 2 mysql mysql 4096 Sep 9 10:38 ptest_biz_baseline_secd
drwxr-x-- 2 mysql mysql 225280 Sep 10 21:03 ptest_biz_baseline_sys
drwxr-x-- 2 mysql mysql 24576 Sep 9 11:48 ptest_biz_baseline_tmc
drwxr-x-- 2 mysql mysql 28672 Sep 9 11:03 ptest_biz_baseline_wfs
drwxr-x-- 2 mysql mysql 8192 Mar 12 2019 sys
drwxr-x-- 2 mysql mysql 4096 May 21 18:37 tpcc
```

共享表空间

redo log

```
root@centos7-65:~# ls -l /var/lib/mysql/ptest_biz_baseline_fi/
-rw-r--r-- 1 mysql mysql 8878 Sep 9 10:45 t_gl_voucherabstract.frm
-rw-r--r-- 1 mysql mysql 114688 Sep 9 10:45 t_gl_voucherabstract.ibd
-rw-r--r-- 1 mysql mysql 8670 Sep 9 10:45 t_gl_voucherabstract.l.frm
-rw-r--r-- 1 mysql mysql 114688 Sep 9 10:45 t_gl_voucherabstract.l.ibd
-rw-r--r-- 1 mysql mysql 9840 Sep 9 11:17 t_gl_voucheramortscheme.frm
-rw-r--r-- 1 mysql mysql 114688 Sep 9 10:45 t_gl_voucheramortscheme.ibd
-rw-r--r-- 1 mysql mysql 8662 Sep 9 10:45 t_gl_voucheramortscheme.l.frm
-rw-r--r-- 1 mysql mysql 114688 Sep 9 10:45 t_gl_voucheramortscheme.l.ibd
-rw-r--r-- 1 mysql mysql 8978 Sep 9 11:17 t_gl_voucherargs.frm
-rw-r--r-- 1 mysql mysql 114688 Sep 9 11:18 t_gl_voucherargs.ibd
-rw-r--r-- 1 mysql mysql 9410 Sep 9 10:45 t_gl_voucherbreakpoint.frm
-rw-r--r-- 1 mysql mysql 131072 Sep 9 10:45 t_gl_voucherbreakpoint.ibd
-rw-r--r-- 1 mysql mysql 9678 Sep 9 10:45 t_gl_voucherentry.frm
-rw-r--r-- 1 mysql mysql 142606336 Sep 10 21:59 t_gl_voucherentry.ibd
-rw-r--r-- 1 mysql mysql 10070 Sep 9 11:17 t_gl_voucher.frm
-rw-r--r-- 1 mysql mysql 54525952 Sep 10 22:04 t_gl_voucher.ibd
-rw-r--r-- 1 mysql mysql 8802 Sep 9 10:46 t_gl_voucherrelation.frm
-rw-r--r-- 1 mysql mysql 131072 Sep 9 10:46 t_gl_voucherrelation.ibd
-rw-r--r-- 1 mysql mysql 8652 Sep 9 10:46 t_gl_vouchertypeexc.frm
-rw-r--r-- 1 mysql mysql 114688 Sep 9 10:46 t_gl_vouchertypeexc.ibd
-rw-r--r-- 1 mysql mysql 9178 Sep 9 10:46 t_gl_vouchertype.frm
-rw-r--r-- 1 mysql mysql 114688 Sep 9 10:46 t_gl_vouchertype.ibd
-rw-r--r-- 1 mysql mysql 8662 Sep 9 10:46 t_gl_vouchertype.l.frm
-rw-r--r-- 1 mysql mysql 114688 Sep 9 10:46 t_gl_vouchertype.l.ibd
-rw-r--r-- 1 mysql mysql 8600 Sep 9 10:46 t_gl_vouchertype_r2.frm
-rw-r--r-- 1 mysql mysql 98304 Sep 9 10:46 t_gl_vouchertype_r2.ibd
-rw-r--r-- 1 mysql mysql 8600 Sep 9 10:46 t_gl_vouchertype_r3.frm
-rw-r--r-- 1 mysql mysql 98304 Sep 9 10:46 t_gl_vouchertype_r3.ibd
-rw-r--r-- 1 mysql mysql 8652 Sep 9 10:46 t_gl_vouchertype_u.frm
-rw-r--r-- 1 mysql mysql 114688 Sep 10 09:39 t_gl_vouchertype_u.ibd
-rw-r--r-- 1 mysql mysql 8830 Sep 9 10:46 t_gl_vouchertypeusereg.frm
-rw-r--r-- 1 mysql mysql 131072 Sep 9 10:46 t_gl_vouchertypeusereg.ibd
```

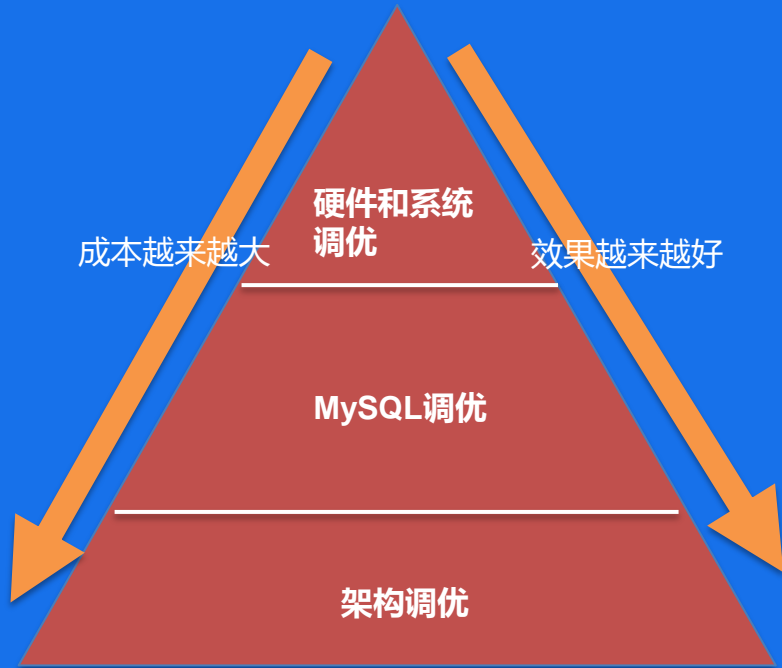
独立表空间

01 / MySQL体系架构

02 / MySQL架构设计

03 / MySQL性能优化

## ■ 2.1 金字塔法则



## ■ 2.1 金字塔法则

- 硬件优化：CPU，内存，硬盘
- 系统优化：系统参数调优
- MySQL优化：参数优化，索引优化，SQL优化
- MySQL架构设计：读写分离，垂直拆分，水平拆分，冷热分离，中间件，异地双活，两地三中心



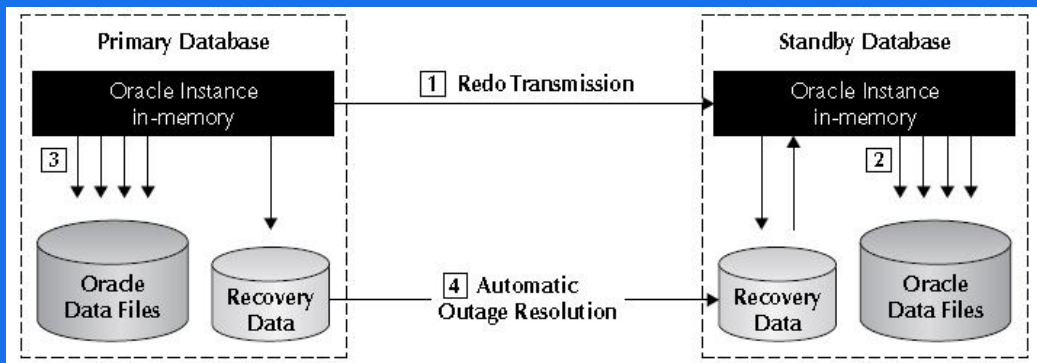
## ■ 2.2MySQL架构设计要素

- 简单可控：满足业务需求越简单越好，能够很好的驾驭
- 演变式：根据业务发展情况，逐步迭代演变
- 可用性：业务高可用，数据库挂掉自动切换
- 一致性：主从延迟，半同步，MRG，异地同步等
- 扩展性：不停机情况下水平扩容读写请求

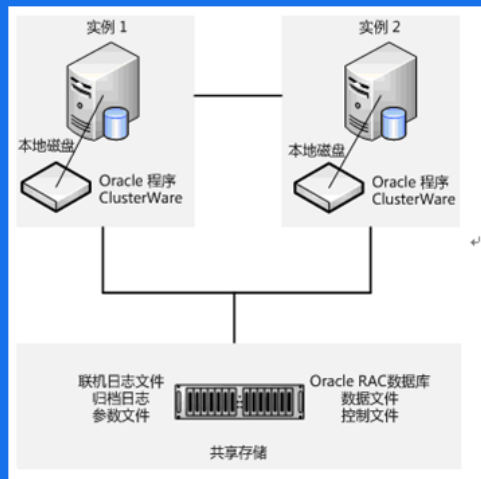
# MySQL架构设计

## ■ 2.3MySQL架构演变-Oracle架构

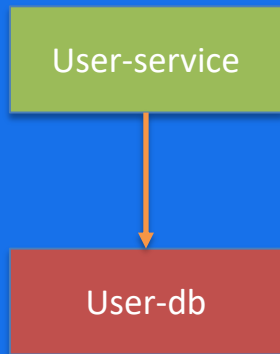
### Dataguard



### Oracle-RAC



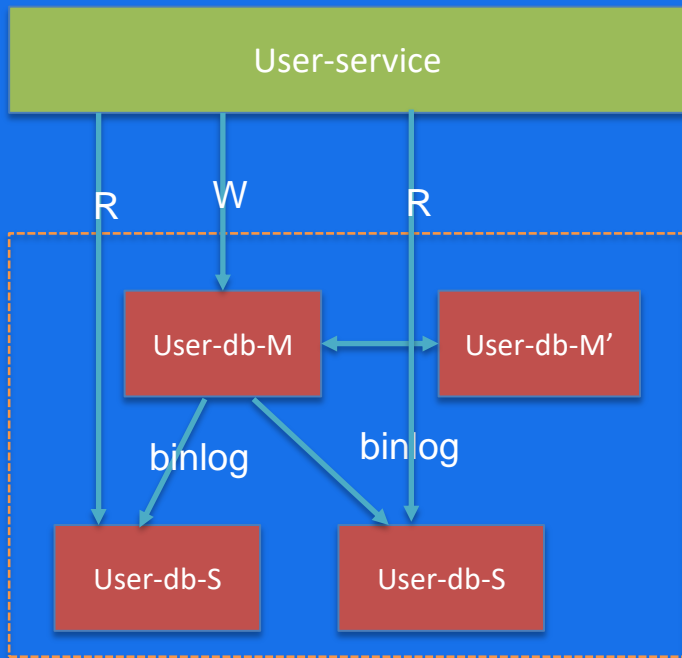
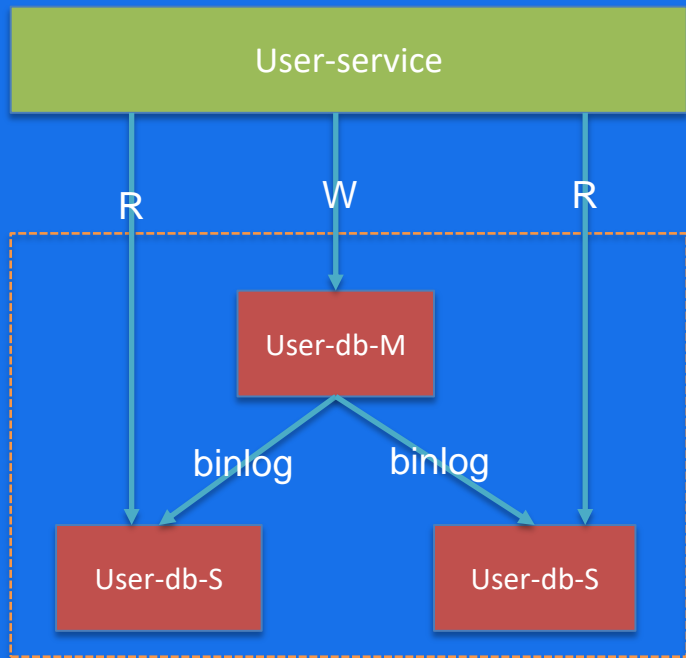
## ■ 2.3MySQL架构演变-单库架构



➤ 开发环境，测试环境，线上环境不允许出现这种架构

# MySQL架构设计

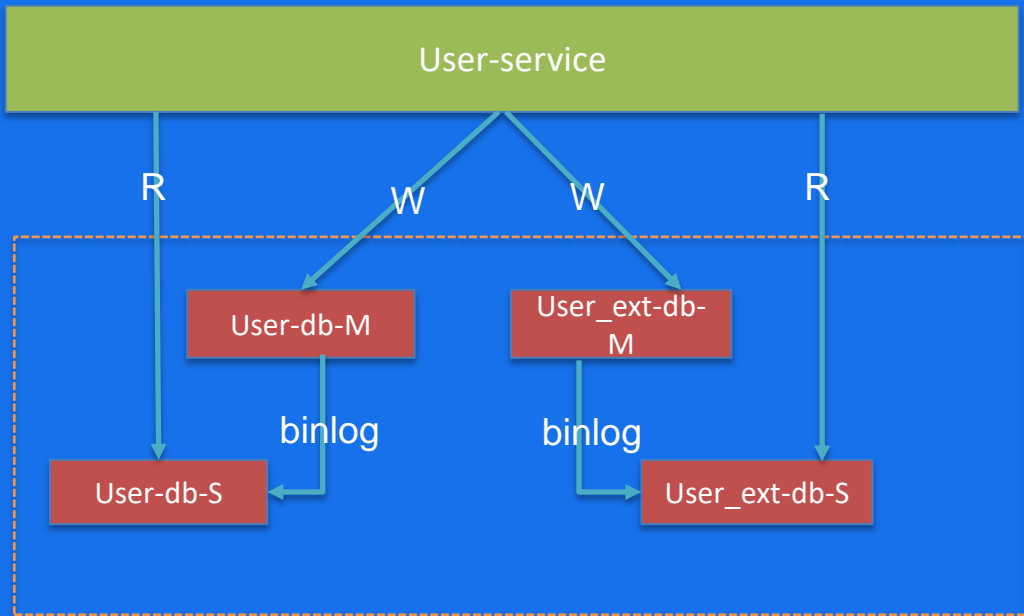
## ■ 2.3MySQL架构演变-分组架构



- 80%公司业务这种架构就够了
- 主要主从延迟问题，读从库数据要对数据的实时要求不高

# MySQL架构设计

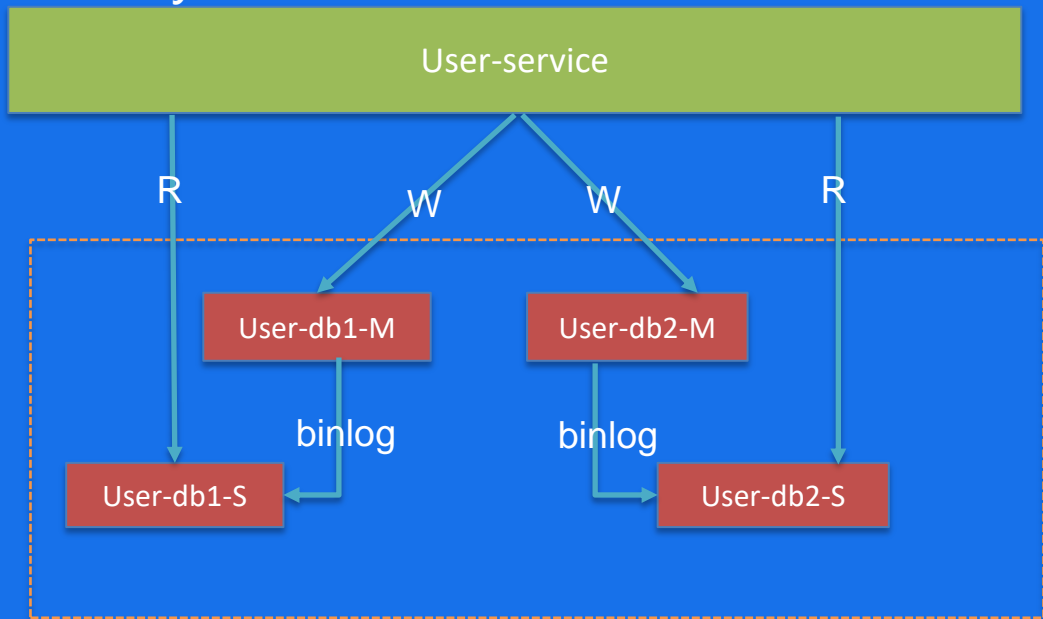
## ■ 2.3MySQL架构演变-分组+垂直切换



- 主库先按功能模块拆分(fi,scm)，同一模块中按照冷热垂直拆分
- 如果是不在同实例或机器不能跨库关联，业务要解耦

# MySQL架构设计

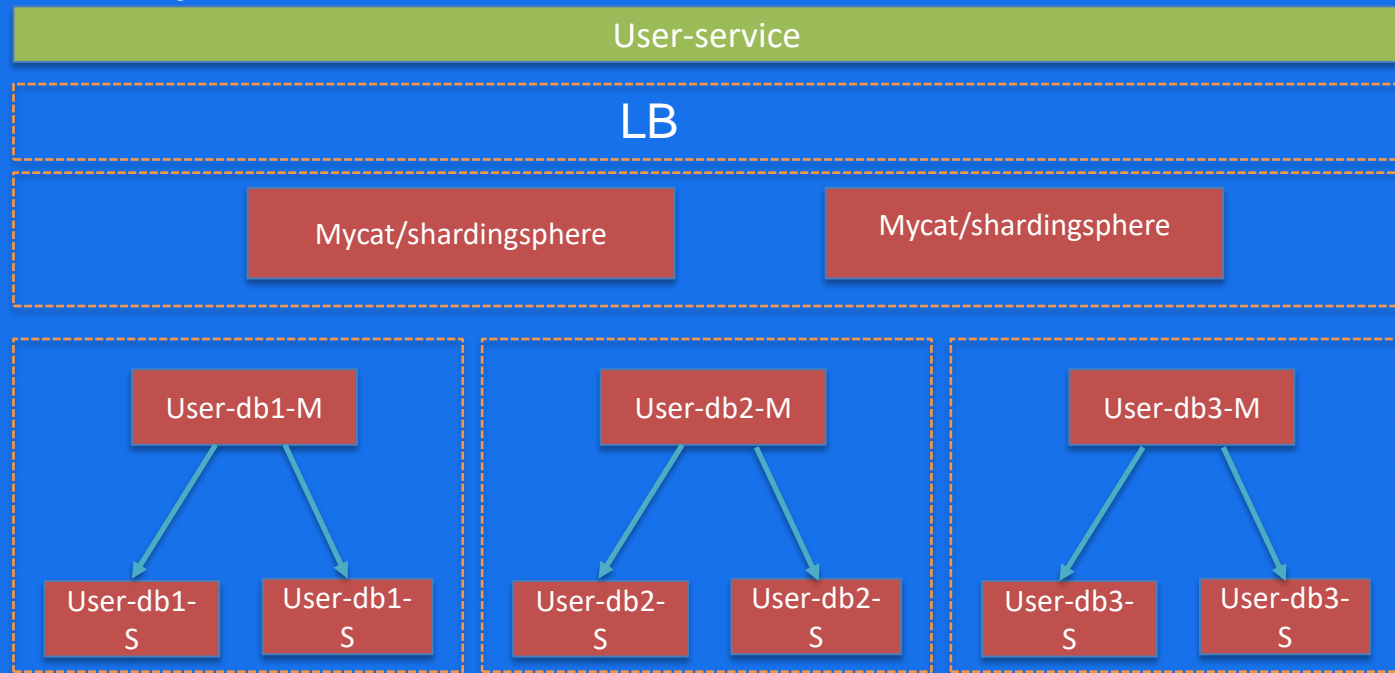
## ■ 2.3MySQL架构演变-分组+水平切分



- 选择合适分片策略mode, hash, range, date
- SQL语句中要带分片键, 避免扫描所有分片表, 性能急剧下降
- 最好分片键中自带路由信息, 识别对应的分片表

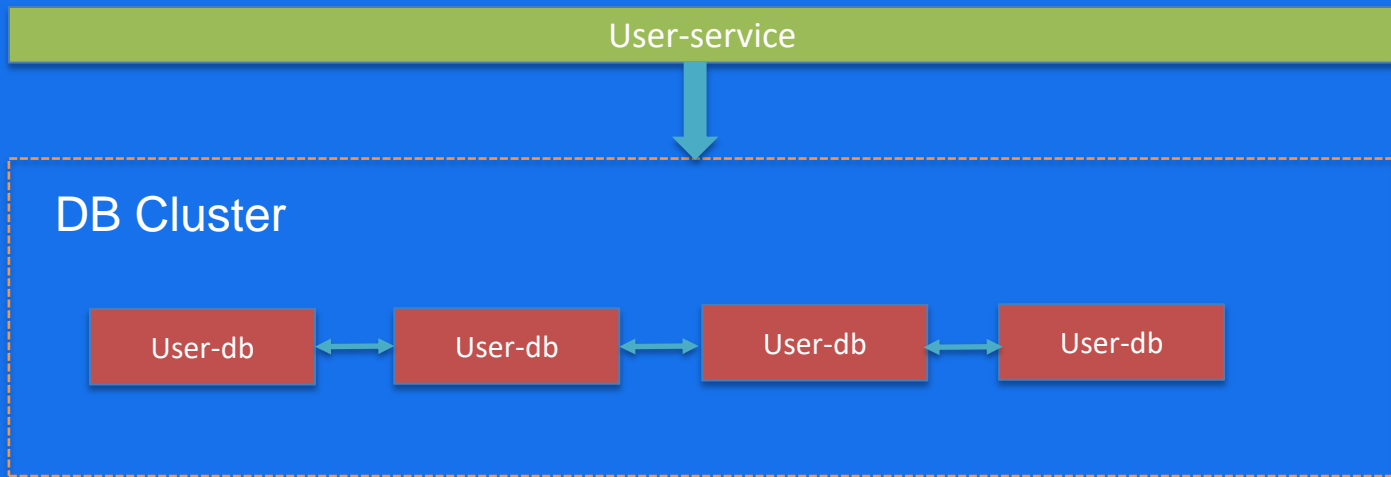
# MySQL架构设计

## ■ 2.3MySQL架构演变-中间件



- 选择合适的中间件, drds,mycat,sharding-jdbc,sharding-proxy,dble 【水平分表】  
altas,kingshard,dbproxy,proxysql,maxscale 【读写分离】
- 需要有二次开发中间件能力或者自研中间件能力

## ■ 2.3MySQL架构演变-分布式数据库



- 分布式存储+分布式计算：TiDB, OceanBase
- 改写MySQL分布式数据库：TDSQL, Aurora, PolarDB
- 海量数据, 高吞吐量, 水平扩展, 高可用, OLAP和OLTP结合
- 全套打包, 对开发友好, 但是成品不成熟, 需要很强的厂商支持



01 / MySQL体系架构

02 / MySQL架构设计

03 / MySQL性能优化

## ■ 3.1 MySQL性能相关参数

参数	参数解释	建议值
<b>Innodb buffer相关</b>		
innodb_buffer_pool_size	数据缓存大小	物理内存50-80%
innodb_max_dirty_pages_pct	控制buffer pool中脏页比例	默认75% 50% 90%
innodb_flush_method	innodb数据文件及redo log的打开、刷写模式	O_DIRECT
innodb_io_capacity	每次刷新脏页的数据量	SSD设置5000
innodb_thread_concurrency	Innodb层并发线程数	等于cpu核数或者默认值0
<b>Redolog相关</b>		
innodb_log_buffer_size	Redo log 缓存大小	根据buffer pool大小调整，64G内可以设置（8M-64M）
innodb_flush_log_trx_commit	Redo log刷新方式	1, 2
innodb_log_file_size	Redo log 文件大小	根据buffer pool大小调整，128M-512M
innodb_log_files_in_group	Redo log文件个数	2-3个

## ■ 3.1MySQL性能相关参数

参数	参数解释	建议值
<b>binlog相关</b>		
binlog_cache_size	Binlog日志缓存，基于会话级	256k
sync_binlog	多少个事务提交缓存后刷新binlog到磁盘	1或者100
<b>Session级buffer</b>		
sort_buffer_size	排序缓存，决定是全字段排序还是rowid排序	64G内存，8M
join_buffer_size	Join是驱动表存放空间，BAK	64G内存，8M
read_buffer_size	全表扫描的MyISAM数据表线程指定缓存	64G内存，4M
read_rnd_buffer_size	优化回表，行指针排序后再去取数据	随机IO转为顺序IO 4M
thread_cache_size	线程缓存，可以重复利用线程，不要反复创建	128
tmp_table_size	临时表大小	128M

## ■ 3.1MySQL性能相关参数

参数	参数解释	建议值
<b>Timeout相关</b>		
connect_timeout	获取数据库连接响应超时，获取连接阶段（authenticate）	默认值10s
net_read_timeout	服务端等待客户端读数据超时，连接繁忙阶段（query）	设置5min
net_write_timeout	服务端等待客户端写数据超时，连接繁忙阶段（query）	设置5min
wait_timeout	连接空闲超时，非交互是连接，连接空闲阶段（sleep）	8小时
interactive_timeout	连接空闲超时，交换式连接，连接空闲阶段（sleep）	8小时
innodb_lock_wait_timeout	Innodb锁等待超时	120s

性能关键参数：缓存+并发+redolog+binlog+session+timeout

## ■ 3.2MySQL表设计

- 表中要有主键，主键为整型，有序，推荐类型为 INT 或者 BIGINT 类型
- 表的存储引擎选择innodb，字符集选择utf8或utf8mb4
- 表的字段数不要超过50个，字段很多时垂直拆分表
- 同一表中，所有 varchar 字段的长度加起来，不能大于 65535，最好不要超过 8K，保证每页能够存2行数据
- 控制单表的数据量在5KW以内，超过就要考虑水平分表了
- 需要多表 join 的字段，名称和数据类型保持绝对一致
- 平衡范式和冗余，效率，一致性，扩展性

## ■ 3.3MySQL索引设计

### 索引类型

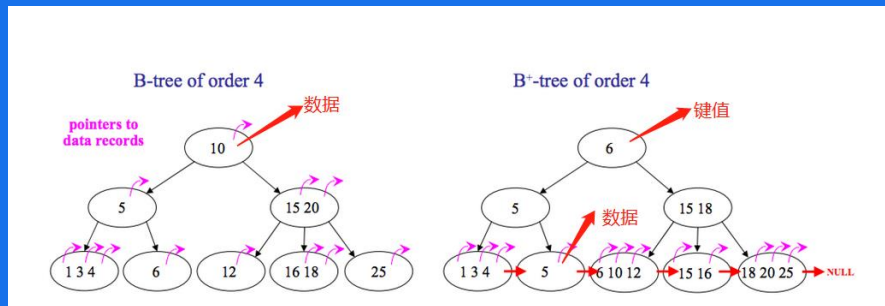
B+Tree索引：大部分都是，像主键，唯一索引，普通索引

Hash索引：mysql自适应hash索引，自动生成维护，无法人为干预

全文索引，myisam，mysql5.6以上

### 相对于B树B+Tree索引特点

- 非叶子节点不存储数据，仅存储键值
- 所有键值都在叶子节点出现
- 叶子节点数据按照顺序存储，有双向链表



行大小1k，高度为3的树能够存储21902400行数据，树的高度一般为2-3

# MySQL优化

## ■ 3.3MySQL索引设计

```
mysql> create table T(  
id int primary key,  
k int not null,  
name varchar(16),  
index (k))  
engine=InnoDB;
```

表中 R1~R5 的 (ID,k) 值分别为

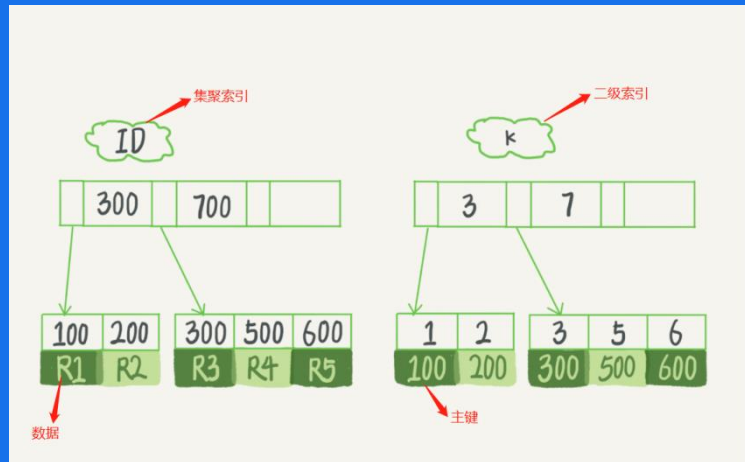
(100,1)

(200,2)

(300,3)

(500,5)

(600,6)



# MySQL优化

## ■ 3.3MySQL索引设计 覆盖索引

```
CREATE TABLE `tuser` (  
  `id` int(11) NOT NULL,  
  `id_card` varchar(32) DEFAULT NULL,  
  `name` varchar(32) DEFAULT NULL,  
  `age` int(11) DEFAULT NULL,  
  `ismale` tinyint(1) DEFAULT NULL,  
  PRIMARY KEY (`id`),  
  KEY `id_card` (`id_card`,`name`),  
  KEY `name_age` (`name`,`age`)  
) ENGINE=InnoDB
```

Database changed

```
mysql> explain select fid,FPHONE from t_sec_user where fphone='13000069915';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	t_sec_user	NULL	ref	IDX_T_SEC_USER_PHONE	IDX_T_SEC_USER_PHONE	110	const	1	100.00	Using index

1 row in set, 1 warning (0.01 sec)

```
mysql> explain select fid,FNUMBER from t_sec_user where fphone='13000069915';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	t_sec_user	NULL	ref	IDX_T_SEC_USER_PHONE	IDX_T_SEC_USER_PHONE	110	const	1	100.00	NULL

1 row in set, 1 warning (0.00 sec)

```
select id_card,name from tuser where id_card=xxxx  
select age from from tuser where id_card=xxxx  
select age from from tuser where name=xxxx
```

通过索引树查询直接可以获得数据，不需要回表，Select 后面的字段和 where条件后面的字段都在复合索引中。



## ■ 3.3MySQL索引设计

### 最左前缀原则

联合索引 (name, age)

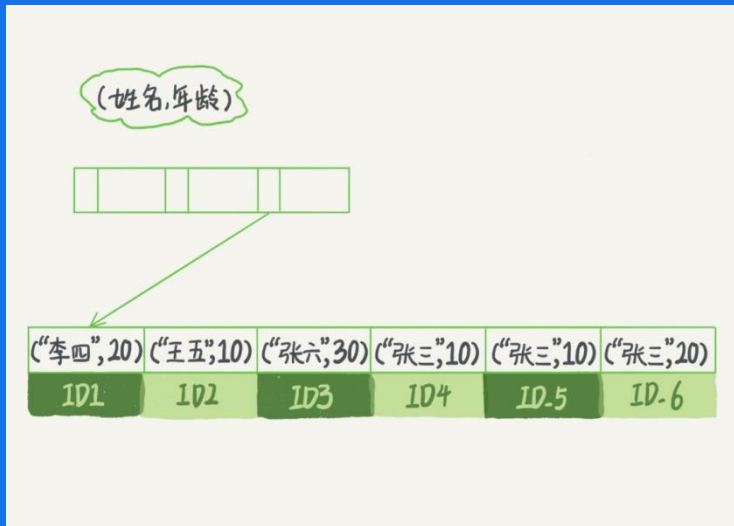
Select \* from t\_user where name like '张三%'

Select \* from t\_user where name like '张%'

Select \* from t\_user where age=10 and name='张三'

Select \* from t\_user where age like '10%'

最左前缀原则：索引复用高，选择性高放最左边



## ■ 3.3MySQL索引设计

### 索引下推

ICP (index condition pushdown) 是MySQL利用索引(二级索引)元组和筛字段在索引中的where条件从表中提取数据记录的一种优化操作, ICP(优化器)尽可能的把index condition的处理从server层下推到storage engine层。storage engine使用索引过滤不相关的数据, 仅返回符合index condition条件的数据给server层

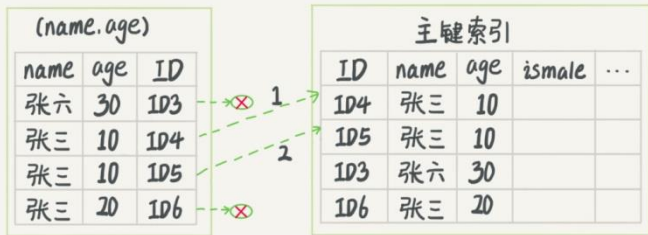
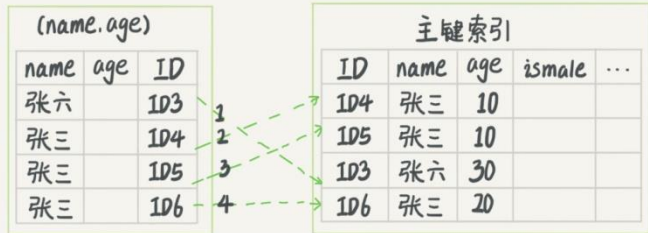
# MySQL优化

## ■ 3.3MySQL索引设计

### 索引下推

索引(name,age)

mysql> select \* from tuser where name like '张%' and age=10 and ismale=1;



# MySQL优化

## ■ 3.3MySQL索引设计

### 设计原则

- 索引是双刃剑，过多的索引会影响插入和更新的速度且影响整体性能，一般索引数量不要超过5个
- 在选择性高的字段是建索引，注意组合索引的顺利，利用索引的最左原则，
- 最常使用的字段放前面，区分度最大的字段放在前面
- 尽量使用复合索引，而不是添加新的索引
- 尽可能建普通索引而不是唯一索引
- 避免冗余索引(a,b,c)，(a)，(a,b)

## ■ 3.4MySQL锁

锁保证数据并发访问的一致性、有效性；锁冲突也是影响数据库并发访问性能的一个重要因素。锁是MySQL在服务器层和存储引擎层的并发控制

根据加锁的范围，MySQL 里面的锁大致可以分成

全局锁：Flush tables with read lock (FTWRL), set global read\_only=1

表级锁：一种是表锁，一种是元数据锁(meta data lock, MDL)

行锁：共享锁，排他锁，间隙锁，自增锁等

## ■ 3.4MySQL锁-表锁

表锁的语法是 lock tables ... read/write

session1	session2
lock tables t1 read, t2 write;	
只能执行读 t1、读写 t2 的操作	写 t1、读写 t2 的语句都会被阻塞
Unlock tables	

使用表锁的情况

1.事务需要更新大部分或全部数据，表又比较大，如果使用默认的行锁，不仅这个事务执行效率低，而且可能造成其他事务长时间锁等待和锁冲突

2.事务涉及多个表，比较复杂，很可能引起死锁，造成大量事务回滚。

这种情况也可以考虑一次性锁定事务涉及的表，从而避免死锁、减少数据库因事务回滚带来的开销。

# MySQL优化

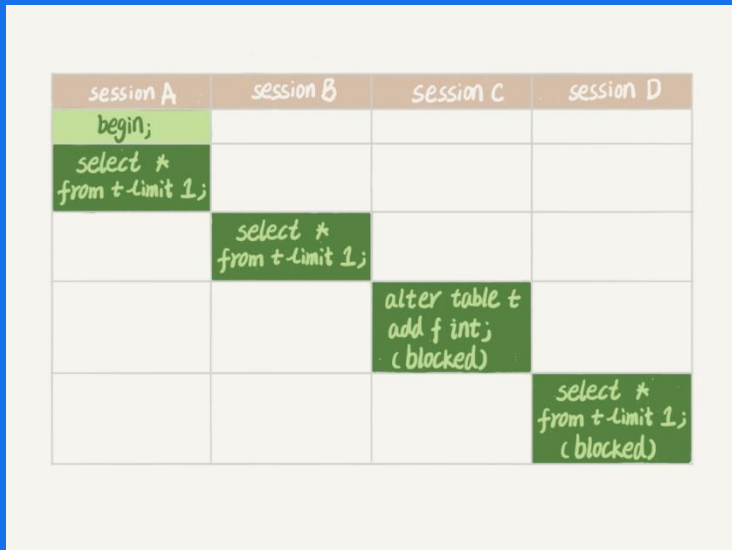
## ■ 3.4MySQL锁-元数据锁

用于解决或者保证DDL操作与DML操作之间的一致性。当对一个表做增删改查操作的时候，加 MDL 读锁；当要对表做结构变更操作的时候，加 MDL 写锁。

读锁之间不互斥，因此你可以有多个线程同时对一张表增删改查。

读写锁之间、写锁之间是互斥的，用来保证变更表结构操作的安全性

锁要等到事务提交才会释放，不是SQL执行完成



SER	HOST	DB	COMMAND	TIME	STATE	INFO
c_prod_110.200.23.69	51670	prod_tgit_ci	Query	1288	Waiting for tal/	"NO_SHARDING"/DROP TABLE IF EXISTS TEMP_A983260966177211392
c_prod_110.200.23.69	51672	prod_tgit_ci	Query	1379	Sending data	SELECT A.fid AS "F1", A.fperiodendactualcost AS "F2", A.fcostaccountid AS "F3", A.fcalorgid AS "F4", B.fgroupid AS "F5"
c_prod_110.200.23.186	4248	prod_tgit_ci	Query	505	Waiting for tal	select * from TEMP_A983260966177211392
ant_110.200.0.201	63748	newmc	Query	0	executing	select * from information schema.PROCESSLIST where command <> 'Sleep'

2.等待获取元数据写锁

3.阻塞查询

1.事务没有提交持有元数据锁

# MySQL优化

## ■ 3.4MySQL锁-行锁

行锁的劣势：开销大；加锁慢；会出现死锁

行锁的优势：锁的粒度小，发生锁冲突的概率低；处理并发的能力强

加锁的方式：自动加锁。对于UPDATE、DELETE和INSERT语句，InnoDB会自动给涉及数据集加排他锁；对于普通SELECT语句，InnoDB不会加任何锁，是**快照读**；当然我们也可以显示的加锁：

共享锁：select \* from tableName where ... +  
lock in share mode

排他锁：select \* from tableName where ... +  
for update





# MySQL优化

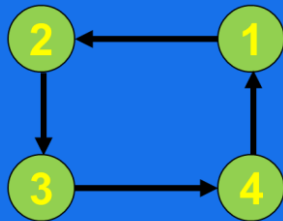
## ■ 3.4MySQL锁-死锁

当并发系统中不同线程出现循环资源依赖，涉及的线程都在等待别的线程释放资源时，就会导致这几个线程都进入无限等待的状态，称为死锁

当出现死锁以后，有两种策略

一种策略是，直接进入等待，直到超时。这个超时时间可以通过参数 `innodb_lock_wait_timeout` 来设置。

另一种策略是，发起死锁检测，发现死锁后，主动回滚死锁链条中的某一个事务，让其他事务得以继续执行。将参数 `innodb_deadlock_detect` 设置为 `on`，表示开启这个逻辑。



AB-BA死锁

事务A	事务B
begin;	begin;
update t set k=k+1 where id=1;	
	update t set k=k+1 where id=2;
update t set k=k+1 where id=2;	
	update t set k=k+1 where id=1;

## ■ 3.4MySQL锁-死锁

### S-lock 升级为 X-lock 不能直接继承

session1	session2	备注
begin	begin	
SELECT * FROM t WHERE i = 1 LOCK IN SHARE MODE;		获取S-lock
	DELETE FROM t WHERE i = 1;	想要获取X-lock，但是被session1的S-lock 卡住，目前处于waiting lock阶段
DELETE FROM t WHERE i = 1;		想要获取X-lock，session1本身拥有S-lock，但是由于session 2 获取X-lock在前，所以session1不能够从S-lock 提升到 X-lock，需要等待session2 释放才可以获取，所以造成死锁

## ■ 3.4MySQL锁-死锁

唯一索引死锁，S2/S3需要唯一键冲突检测，需要加S锁

session1	session2	session3	备注
begin	begin	begin	
delete from uk where a=1;			获取X-lock锁
	insert into uk values(1);		wait lock(想要加S-lock，却被session1的X-lock卡住)
		insert into uk values(1);	wait lock（想要加S-lock，却被session1的X-lock卡住）
commit;			session2和session3 都获得了S-lock，然后都想要去给记录1 加上X-lock，却互相被对方的S-lock卡住，死锁产生

# MySQL优化

## ■ 3.4MySQL锁-快速发现锁等待

### Innodb相关锁

```
select b.trx_mysql_thread_id as '被阻塞线程'  
,b.trx_query as '被阻塞SQL'  
,c.trx_mysql_thread_id as '阻塞线程'  
,c.trx_query as '阻塞SQL'  
,(UNIX_TIMESTAMP() - UNIX_TIMESTAMP(c.trx_started)) as '阻塞时间'  
from  
information_schema.innodb_lock_waits a  
join  
information_schema.innodb_trx b on a.requesting_trx_id=b.trx_id  
join  
information_schema.innodb_trx c on a.blocking_trx_id=c.trx_id  
where  
(UNIX_TIMESTAMP() - UNIX_TIMESTAMP(c.trx_started))>10;  
或  
select * from sys.innodb_lock_waits\G
```

```
mysql> select * from sys.innodb_lock_waits\G  
***** 1. row *****  
wait_started: 2020-09-17 14:50:48  
wait_age: 00:00:22  
wait_age_secs: 22  
locked_table: 'ptest_biz_baseline_fi'. 't_gl_voucher'  
locked_index: PRIMARY  
locked_type: RECORD  
waiting_trx_id: 838801319  
waiting_trx_started: 2020-09-17 14:50:48  
waiting_trx_age: 00:00:22  
waiting_trx_rows_locked: 1  
waiting_trx_rows_modified: 0  
waiting_pid: 213647  
waiting_query: update t_gl_voucher set FNUMBE ... ' where fid=717855977235311616  
waiting_lock_id: 838801319:2510314:5:5  
blocking_trx_id: 838801316  
blocking_pid: 213278  
blocking_query: NULL  
blocking_lock_id: 838801316:2510314:5:5  
blocking_lock_mode: X  
blocking_trx_started: 2020-09-17 14:50:45  
blocking_trx_age: 00:00:25  
blocking_trx_rows_locked: 1  
blocking_trx_rows_modified: 1  
sql_kill_blocking_query: KILL QUERY 213278  
sql_kill_blocking_connection: KILL 213278  
1 row in set, 3 warnings (0.01 sec)
```

锁的表和索引

被阻塞的SQL

阻塞SQL

持有锁thread ID

## ■ 3.4MySQL锁-快速发现锁等待

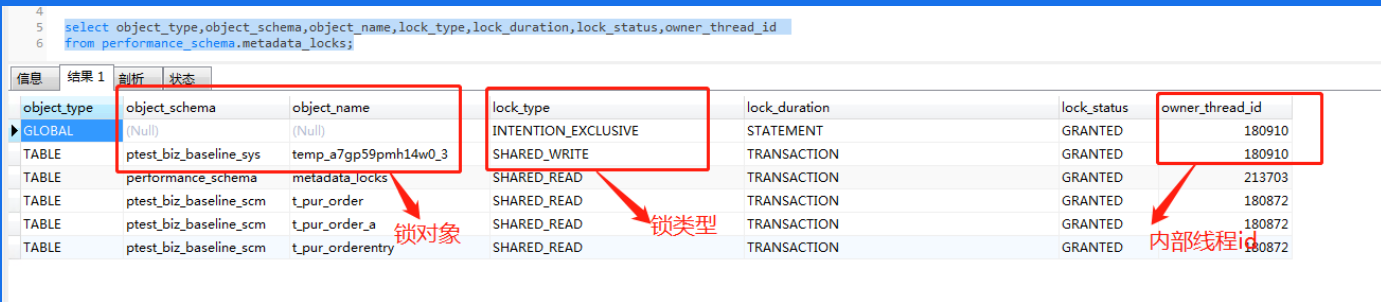
### 元数据锁

#### performance\_schema=on

```
UPDATE performance_schema.setup_instruments SET ENABLED = 'YES', TIMED = 'YES'  
WHERE NAME = 'wait/lock/metadata/sql/mdl';
```

```
select object_type,object_schema,object_name,lock_type,lock_duration,lock_status,owner_thread_id  
from performance_schema.metadata_locks;
```

```
select * from performance_schema.threads where thread_id in (980365,978840)
```



The screenshot shows the results of a SQL query in a MySQL client. The query is: `select object_type,object_schema,object_name,lock_type,lock_duration,lock_status,owner_thread_id from performance_schema.metadata_locks;`. The results are displayed in a table with 7 columns. Red boxes highlight specific columns: 'object\_name', 'lock\_type', and 'owner\_thread\_id'. Red arrows point from the text '锁对象' (lock object) to the 'object\_name' column, '锁类型' (lock type) to the 'lock\_type' column, and '内部线程id' (internal thread ID) to the 'owner\_thread\_id' column.

object_type	object_schema	object_name	lock_type	lock_duration	lock_status	owner_thread_id
GLOBAL	(Null)	(Null)	INTENTION_EXCLUSIVE	STATEMENT	GRANTED	180910
TABLE	ptest_biz_baseline_sys	temp_a7gp59pmh14w0_3	SHARED_WRITE	TRANSACTION	GRANTED	180910
TABLE	performance_schema	metadata_locks	SHARED_READ	TRANSACTION	GRANTED	213703
TABLE	ptest_biz_baseline_scm	t_pur_order	SHARED_READ	TRANSACTION	GRANTED	180872
TABLE	ptest_biz_baseline_scm	t_pur_order_a	SHARED_READ	TRANSACTION	GRANTED	180872
TABLE	ptest_biz_baseline_scm	t_pur_orderentry	SHARED_READ	TRANSACTION	GRANTED	180872

# MySQL优化

## ■ 3.4MySQL锁-死锁

Show engine innodb status\G

或者innodb\_print\_all\_deadlocks=ON 打印到错误日志

```
-----
LATEST DETECTED DEADLOCK
-----
2020-09-04 15:45:03 0x7f58e373b700
*** (1) TRANSACTION:
TRANSACTION 286051926, ACTIVE 32 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 3 lock struct(s), heap size 1136, 2 row lock(s), undo log entries 1
MySQL thread id 2136489, OS thread handle 140018825451264, query id 1297990621 192.168.63.63 crp_user updating
update t1 set k=k+1 where id=2
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 64758 page no 3 n bits 72 index PRIMARY of table `xtest`.`t1` trx id 286051926 lock_mode X locks rec but not gap waiting
Record lock, heap no 3 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
 0: len 4; hex 80000002; asc  ;;
 1: len 6; hex 0000110cce5b; asc  [;;
 2: len 7; hex 40000022f61c2e; asc @  .;;
 3: len 4; hex 80000003; asc  ;;

*** (2) TRANSACTION:
TRANSACTION 286051931, ACTIVE 15 sec starting index read, thread declared inside InnoDB 5000
mysql tables in use 1, locked 1
3 lock struct(s), heap size 1136, 2 row lock(s), undo log entries 1
MySQL thread id 2136876, OS thread handle 140019749861120, query id 1297990622 192.168.63.63 crp_user updating
update t1 set k=k+1 where id=1
*** (2) HOLDS THE LOCK(S):
RECORD LOCKS space id 64758 page no 3 n bits 72 index PRIMARY of table `xtest`.`t1` trx id 286051931 lock_mode X locks rec but not gap
Record lock, heap no 3 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
 0: len 4; hex 80000002; asc  ;;
 1: len 6; hex 0000110cce5b; asc  [;;
 2: len 7; hex 40000022f61c2e; asc @  .;;
 3: len 4; hex 80000003; asc  ;;

*** (2) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 64758 page no 3 n bits 72 index PRIMARY of table `xtest`.`t1` trx id 286051931 lock_mode X locks rec but not gap waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
 0: len 4; hex 80000001; asc  ;;
 1: len 6; hex 0000110cce56; asc  V;;
 2: len 7; hex 3d000022fc20bd; asc =  ";;
 3: len 4; hex 80000002; asc  ;;

*** WE ROLL BACK TRANSACTION (2)
-----
```

# MySQL优化

## ■ 3.4MySQL锁-死锁

锁是一种正常机制，我们要做的是避免有长时间的锁等待，不能有死锁

锁等待：SQL执行慢，大量SQL堆积，连接数爆掉

死锁：CPU使用率高，QPS急剧下降，回滚请求失败

解决死锁的办法

1. 降低隔离级别，修改 RR -> RC，无间隙锁(gap)
2. 调整业务逻辑和SQL，让其都按照顺序执行操作，把可能发生造成锁冲突、最可能影响并发度的锁的申请时机尽量往后放。
3. 减少unique索引，大部分死锁的场景都是由于unique索引导致
4. 为表添加合理的索引，如果不走索引将会为表的每一行记录加锁，死锁的概率就会大大增大
5. 避免大事务，尽量将大事务拆成多个小事务来处理

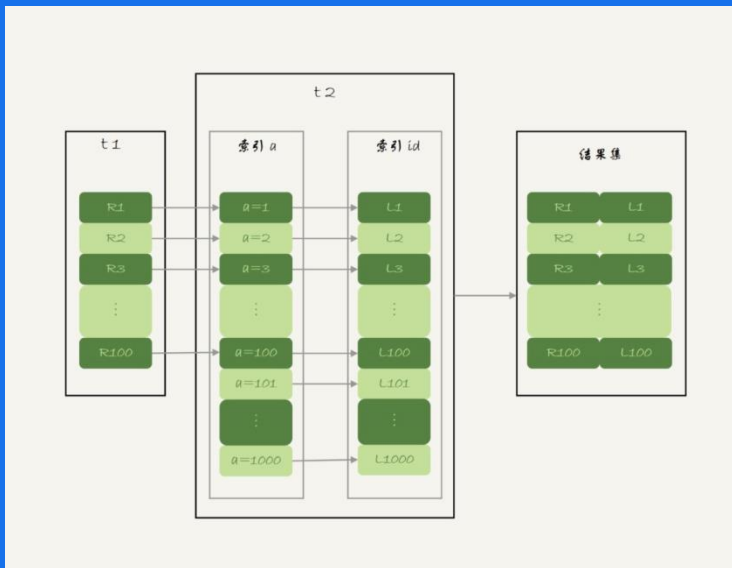
# MySQL优化

## ■ 3.5MySQL中join

- 当使用left join时，左表是驱动表，右表是被驱动表
- 当使用right join时，右表是驱动表，左表是被驱动表
- 当使用join时，MySQL会选择数据量比较小的表作为驱动表，大表作为被驱动表

Index Nested-Loop Join

t2表有索引

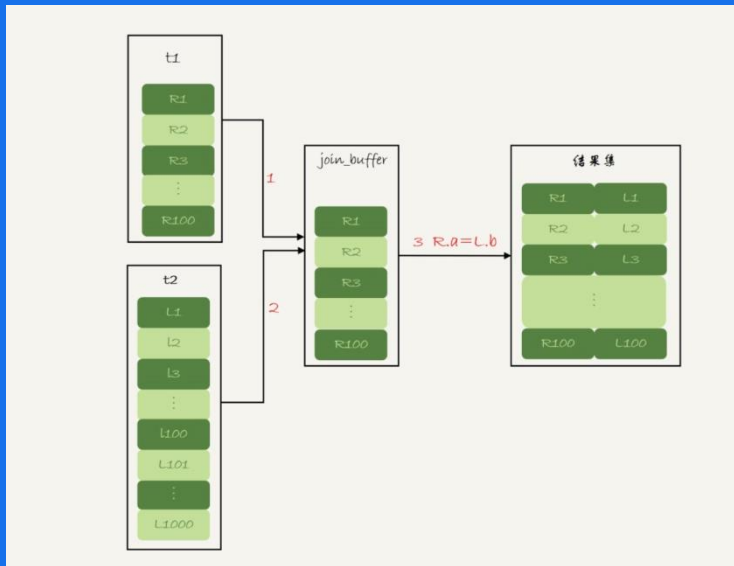




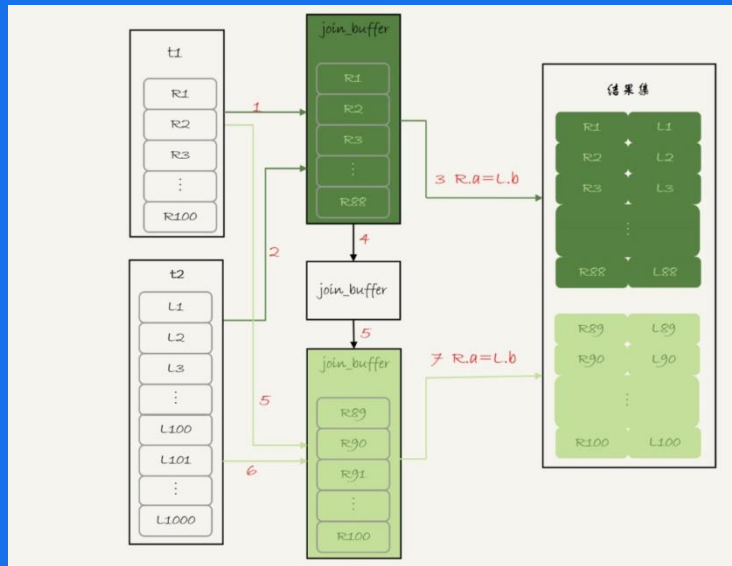
## ■ 3.5MySQL中join

### Block Nested-Loop Join

t2 无索引 AND  $t1 < \text{join\_buffer\_size}$



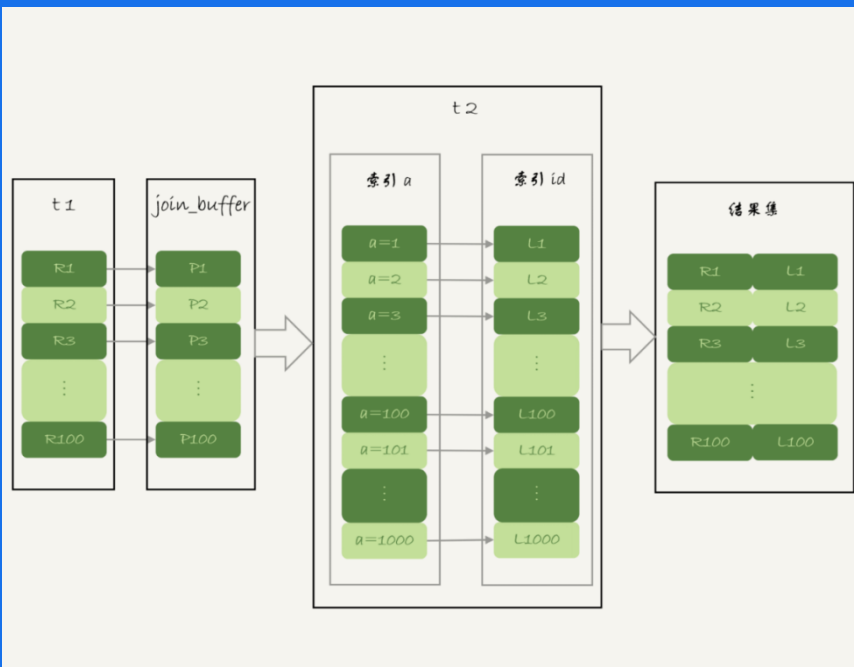
t2 无索引 AND  $t1 > \text{join\_buffer\_size}$



## ■ 3.5MySQL中join

### Batched Key Access

set optimizer\_switch='mrr=on,mrr\_cost\_based=off,batched\_key\_access=on';



## ■ 3.5MySQL中join

### JOIN优化原则

- 尽量使用inner join, 会自动选择小表去驱动表
  - 选择小表为驱动表(left/right join), 包含where条件筛选后大小
  - 被驱动表上要建立索引, 尽量走Index Nested-Loop Join
- join\_buffer\_size大小决定驱动表是否可以全部放在内存中
- 如果被驱动表上不合适建索引的可以通过临时表中转然后join

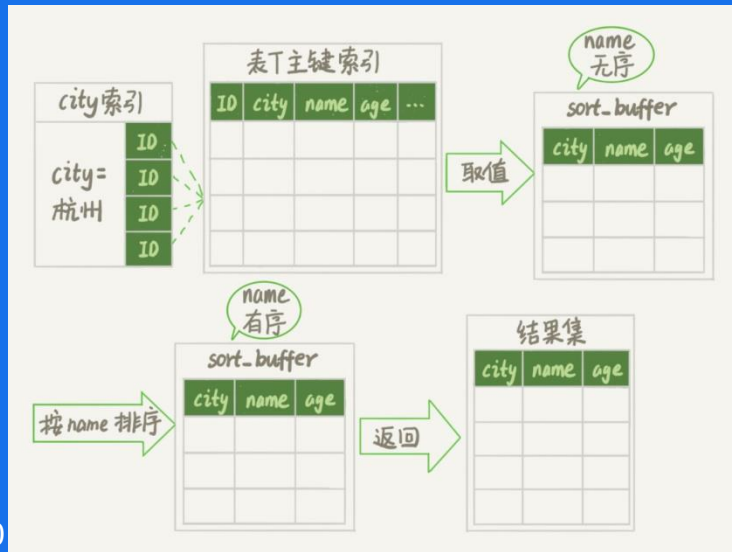
# MySQL优化

## ■ 3.6MySQL中order by

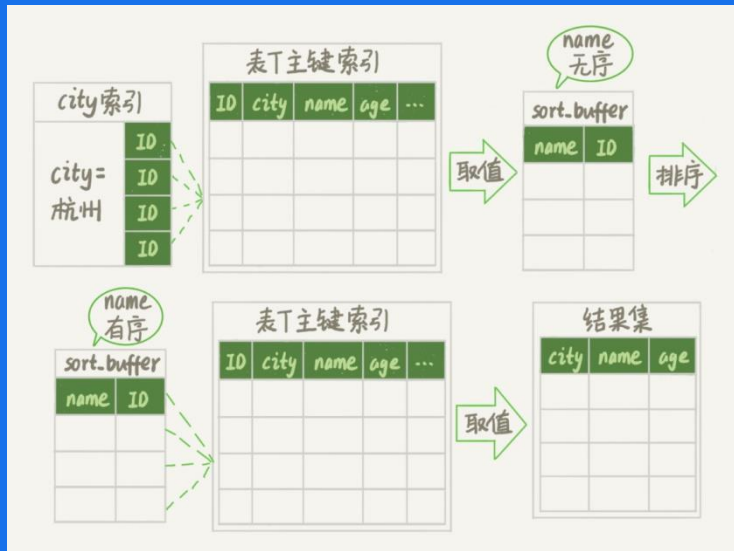
### 全字段排序

```
CREATE TABLE `t` (  
  `id` int(11) NOT NULL,  
  `city` varchar(16) NOT NULL,  
  `name` varchar(16) NOT NULL,  
  `age` int(11) NOT NULL,  
  `addr` varchar(128) DEFAULT NULL,  
  PRIMARY KEY (`id`),  
  KEY `city` (`city`)  
) ENGINE=InnoDB;
```

```
select city,name,age from t where city='杭州' order by name limit 1000
```



- 3.6MySQL中order by  
rowid 排序(双路排序)  
当查询字段大小>max\_length\_for\_sort\_data

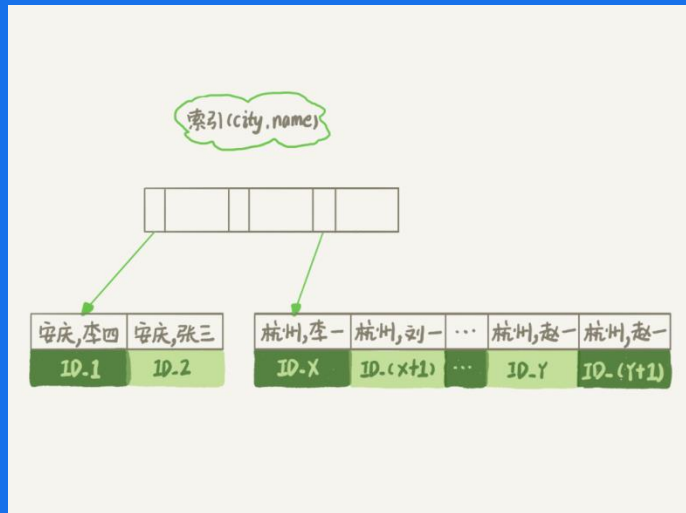


## ■ 3.6MySQL中order by

创建复合索引后的排序

```
select city,name,age from t where city='杭州' order by name limit 1000
```

```
alter table t add index city_user(city, name);
```



## ■ 3.6MySQL中order by

### 排序优化总结

- 尽量建立筛选条件和排序字段的复合索引，减少排序动作
- 合理设置sort\_buffer\_size和max\_length\_for\_sort\_data值
- 不能写select \*, select后的字段长度直接决定排序算法

## ■ 3.7MySQL分页

select * from t_gl_voucher where FORGID=52 limit 1000,10 ;	执行0.035s
select * from t_gl_voucher where FORGID=52 limit 10000,10 ;	执行0.078s
select * from t_gl_voucher where FORGID=52 limit 100000,10 ;	执行0.393s
select * from t_gl_voucher where FORGID=52 limit 1000000,10 ;	执行3.512s

limit1000000,10的意思扫描满足条件的1000010行，扔掉前面的1000000行，返回最后的10行，会导致磁盘io 100%消耗。

explain select * from t_gl_voucher where FORGID=52 limit 1000000,10 ;											
结果 1	剖析	状态									
	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	extra
1	SIMPLE	t_gl_voucher	(Null)	ref	IDX_GL_VCH	IDX_GL_VCH	8	const	2741044	100.00	(Null)



## ■ 3.7MySQL分页

思路：干掉或者利用 limit offset,size 中的offset

不是直接使用limit，而是首先获取到offset的id然后直接使用limit size来获取数据

1.子查询方法，利用覆盖索引

select \* from t\_gl\_voucher where FORGID=52 and

fid>=(select fid from t\_gl\_voucher where FORGID=52 limit 1000000,1 ) limit 10 ; 执行时间0.288s

```
##子查询
explain select * from t_gl_voucher where FORGID=52 and fid>=(select fid from t_gl_voucher where FORGID=52 limit 1000000,1 ) limit 10 ;
explain select * from t_gl_voucher as a
```

结果 1	剖析	状态									
	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	t_gl_voucher	(Null)	range	PRIMARY,IDX_GL_VCH	PRIMARY	8	(Null)	23186	26.17	Using where
2	SUBQUERY	t_gl_voucher	(Null)	ref	IDX_GL_VCH	IDX_GL_VCH	8	const	2741044	100.00	Using index

# MySQL优化

## ■ 3.7MySQL分页

```
select * from t_gl_voucher as a
join (select fid from t_gl_voucher where FORGID=52 limit 1000000,10 ) as b
on a.fid=b.fid
```

执行时间0.288s

```
17
18 explain select * from t_gl_voucher as a
19 join (select fid from t_gl_voucher where FORGID=52 limit 1000000,10 ) as b
20 on a.fid=b.fid
21
22
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	<derived2>	(Null)	ALL	(Null)	(Null)	(Null)	(Null)	1000010	100.00	(Null)
1	PRIMARY	a	(Null)	eq_ref	PRIMARY	PRIMARY	8	b.fid	1	100.00	(Null)
2	DERIVED	t_gl_voucher	(Null)	ref	IDX_GL_VCH	IDX_GL_VCH	8	const	2741044	100.00	Using index

## 2.使用 id 限定优化【这种方式假设数据表的id是连续递增的】

```
select * from orders_history where id >= 1000001 limit 100;
```

# Q&A





感謝  
ขอบคุณ  
terima kasih  
Thanks  
ありがとう  
谢谢