



# 高并发高性能 数据库设计-优化篇



• 讲师 : KimmKing •

# 目录 | Contents

1. 性能分析与性能优化
2. SQL 与查询优化
3. MySQL 高可用架构
4. 数据库分库分表
5. BASE 分布式事务
6. 数据库集群分布式治理
7. 数据库框架与中间件

# 一、性能分析与性能优化

- 吞吐与延迟：有些结论是反直觉的，指导我们关注什么
- 没有量化就没有改进：监控与度量指标，指导我们怎么去入手
- 80/20原则：先优化性能瓶颈问题，指导我们如何去优化
- 过早的优化是万恶之源：指导我们要选择优化的时机

性能是个综合性问题

# 一、性能分析与性能优化

业务系统的分类：计算密集型、数据密集型

业务处理本身无状态，数据状态最终要保存到数据库

- 一般来说，DB/SQL 操作的消耗在一次处理中占比最大
- 业务系统发展的不同阶段和时期，性能瓶颈要点不同，类似木桶装水
- 脱离场景谈性能都是耍流氓：指导我们对性能要求要符合实际

例如传统软件改成 SaaS 软件

## 二、SQL 与查询优化

SQL异类排序

```
mysql> SELECT * FROM dbsql.t_user_info WHERE f_id < 10;
```

f_id	f_username	f_gender	f_idno	f_age	f_created_at	f_updated_at
1	8137683923	0	0.5408197031736557	75	1556932514	1556932514
2	0337434114	1	0.0359018771312217	97	1556932515	1556932515
3	1212279919	0	0.5759212612881524	116	1556932516	1556932516
4	3248705376	0	0.6240283061042692	44	1556932517	1556932517
6	8254726555	0	0.2372036587793414	8	1556932518	1556932518
7	6656017665	1	0.3779560638386347	144	1556932519	1556932519
8	2146530787	1	0.587886469986184	53	1556932520	1556932520
9	8966252235	1	0.19774059410927872	48	1556932521	1556932521

f_id	f_username	f_gender	f_idno	f_age	f_created_at	f_updated_at
4	3248705376	0	0.6240283061042692	44	1556932517	1556932517
3	1212279919	0	0.5759212612881524	116	1556932516	1556932516
2	0337434114	1	0.0359018771312217	97	1556932515	1556932515
1	8137683923	0	0.5408197031736557	75	1556932514	1556932514
6	8254726555	0	0.2372036587793414	8	1556932518	1556932518
7	6656017665	1	0.3779560638386347	144	1556932519	1556932519
8	2146530787	1	0.587886469986184	53	1556932520	1556932520
9	8966252235	1	0.19774059410927872	48	1556932521	1556932521

## 二、SQL 与查询优化

```
SELECT f_id, f_username, f_gender, f_idno, f_age, f_created_at, f_updated_at
FROM dbsql.t_user_info
WHERE f_id < 10
ORDER BY IF(f_id < 5, -f_id, f_id)|
```

## 二、SQL 与查询优化

如何发现需要优化的 SQL?

你了解的 SQL 优化方法有哪些?

SQL 优化有哪些好处?

## 二、SQL 与查询优化

### 虚拟业务组：

- 业务分析人员：宇哥
- 开发攻城狮：睿哥
- 著名DBA：隆兄台
- 此三人是某项目组的核心人员，承接了一个大型系统中某些模块的设计开发工作
- 所有组都是此项目的设计协作方，为该项目组提供技术支持，并且验证方案可行性

## 二、SQL 与查询优化

### 需求-1

增加可以保存用户信息的数据表，必要的用户信息包含：

表名：user\_info

用户名 username

密码 password

姓名 name

性别 gender

身份证号 id\_number

年龄 age

状态 state

数据库连接：

HOST: 172.18.2.55

PORT: 3306

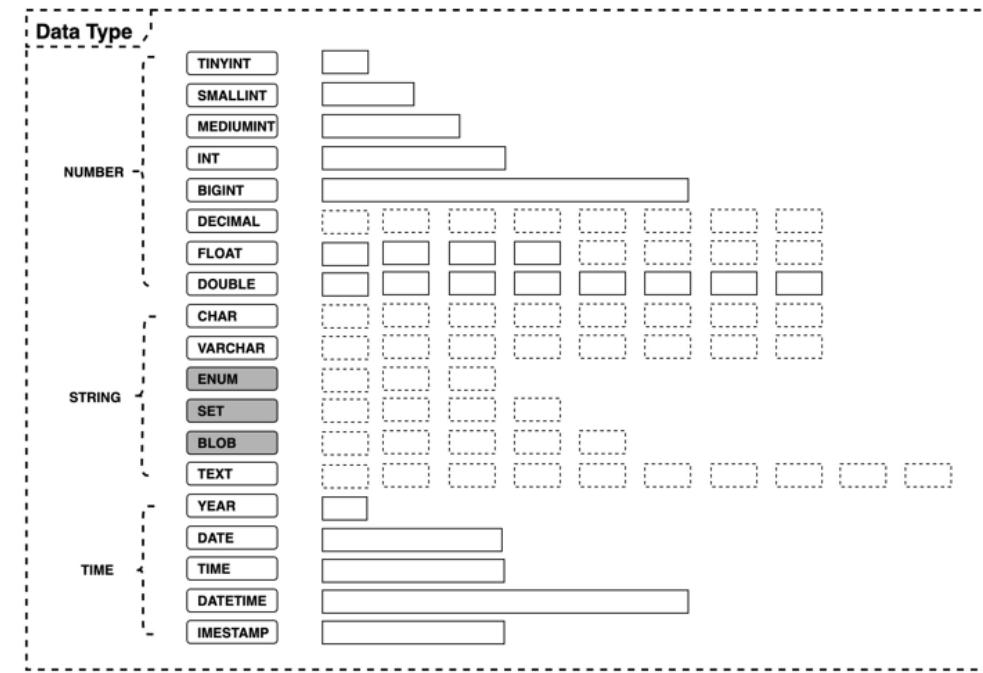
USER: dbsql\_user

PASS: dbsql\_user.12356890

DBNAME: dbsql\_user

## 二、SQL 与查询优化

数据类型是否越大越好？



1row = 4字节

100亿row = 40G

1GB/年 = \$1.8

40 X 1.8 = \$72/年

多维度：内存、网络、磁盘空间、IOPS

多系统：业务多表、财务、大数据

多副本：业务3个Slave，大数据3个副本

长时间：监管要求存储5年

1个数据类型能节省多少\$?

## 二、SQL 与查询优化

### 存储引擎的选择

#### InnoDB

- 1.聚集索引
- 2.锁粒度是行锁
- 3.InnoDB 支持事务

没有其他特别因素就用InnoDB

#### TokuDB

- 1.高压缩比，尤其适用与压缩和归档(1：12)
- 2.在线添加索引，不影响读写操作
- 3.支持完整的ACID特性和事务机制

#### 归档库

## 二、SQL 与查询优化

### 设计-1

```
DROP TABLE IF EXISTS t_user_info;
CREATE TABLE `t_user_info` (
    `f_id`          BIGINT(20)      NOT NULL AUTO_INCREMENT COMMENT '自增ID',
    `f_username`    VARCHAR(20)     NOT NULL DEFAULT ''   COMMENT '用户名',
    `f_password`    VARCHAR(64)     NOT NULL DEFAULT ''   COMMENT '用户密码',
    `f_gender`      TINYINT(11)    NOT NULL DEFAULT 0    COMMENT '性别',
    `f_idno`        CHAR(19)       NOT NULL DEFAULT ''   COMMENT '身份证号',
    `f_age`         SMALLINT(11)   NOT NULL DEFAULT 0    COMMENT '年龄',
    `f_state`       TINYINT(11)    NOT NULL DEFAULT 0    COMMENT '状态',
    `f_created_at` BIGINT(20)     NOT NULL DEFAULT 0    COMMENT '创建时间',
    `f_updated_at` BIGINT(20)     NOT NULL DEFAULT 0    COMMENT '更新时间',
    PRIMARY KEY (`f_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='用户信息';
```

## 二、SQL 与查询优化

### 小结-1

- 注意数据类型的选择
- 设计表之前，通读DBA的指导手册/dbaprinciples

## 二、SQL 与查询优化

### 需求-2

1. 根据身份证号查询用户详细信息
2. 可根据用户名密码登陆
3. 可统计当日新增用户个数

## 二、SQL 与查询优化

### 设计-2

#### 1. 根据身份证号查询用户详细信息

```
SELECT f_id, f_username, f_gender, f_idno, f_age, f_created_at, f_updated_at  
FROM t_user_info  
WHERE f_idno = 'xxx';
```

#### 2. 可根据用户名密码登陆

```
SELECT f_id, f_username, f_gender, f_idno, f_age, f_created_at, f_updated_at  
FROM t_user_info  
WHERE f_username = 'xxx' AND f_password = 'xxxxx';
```

#### 3. 可统计当日新增用户个数

```
SELECT COUNT(*) FROM t_user_info  
WHERE f_created_at > xxxx AND f_created_at < xxxxx;
```

## 二、SQL 与查询优化

### 隐式转换

```
SELECT f_id, f_username, f_gender, f_idno, f_age, f_created_at, f_updated_at  
FROM t_user_info  
WHERE f_username = 'xxx' AND f_password = 0;
```

- show warnings;

带来的问题：

- 错误

另外的问题：

- 不走索引

## 二、SQL 与查询优化

### 小结-2

简单的SQL可能带来大的问题，where条件中注意数据类型，  
避免类型转换

## 二、SQL 与查询优化

### 需求-3

系统经过一个月的运行，用户表增长约100万，DBA接到告警，CPU升高，查询越来越慢，请定位问题并给出解决方案

## 二、SQL 与查询优化

### 分析-3

定位问题的方法:

- 慢查询日志
- 看应用和运维的监控

# 慢查询日志

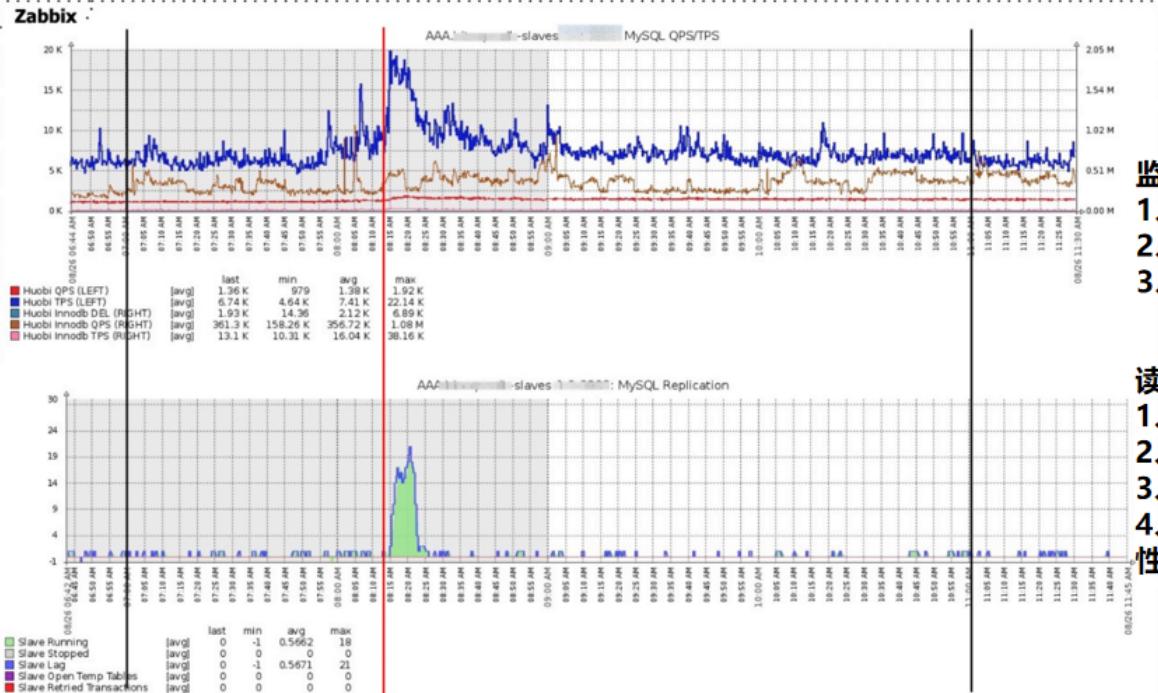
```
slowlog  
# 164.4s user time, 1.1s system time, 36.32M rss, 101.80M vsz          # Query 1: 24.51 QPS, 20.56x concurrency, ID 0x1E33FEA9BC0 at byte ??  
# Current date: Tue Aug 27 10:11:28 2019                                # This item is included in the report because it matches --limit.  
# Hostname: db-monitor.dba.com                                         # Scores: V/M = 0.09  
# Files: aws_slowlog_instance_20190827.log                            # Time range: 2019-08-26T15:00:05 to 2019-08-27T02:00:15  
# Overall: 982.57k total, 17 unique, 24.80 QPS, 20.63x concurrency      # Attribute   pct   total      min      max      avg     95% stddev median  
# Time range: 2019-08-26T15:00:01 to 2019-08-27T02:00:15              # ======  ==  ======  ======  ======  ======  ======  ======  ======  ======  
# Attribute      total      min      max      avg     95% stddev median  # Count      98 970941  
# ======  ==  ======  ======  ======  ======  ======  ======  ======  ======  # Exec time    99 814216s  200ms       3s    839ms      1s    268ms  816ms  
# Exec time     817129s  200ms       3s    832ms      1s    274ms  777ms  # Lock time    99    89s     45us     17ms    91us   103us   88us   89us  
# Lock time      90s    23us     17ms    91us   103us   87us   89us   # Rows sent    98 356.56k      0    544     0.38      0    4.77      0  
# Rows sent     363.69k      0    544     0.38      0    4.77      0  # Rows examine  96 24.03G  25.92k  26.48k  25.95k  25.99k  285.50  24.75k  
# Rows examine   24.81G      0  1.28M  26.48k  25.99k  16.78k  24.75k  # String:  
# Query size    99 521.00M    559    568    562.65    563.87    4.50  537.02  
# Query size    524.62M    90    946    559.86    563.87    32.03    537.02  # Databases  dbname  
# Profile  
# Rank Query ID      Response time      Calls R/Call V/M   Item      # Query_time distribution  
# ======  ======  ======  ======  ======  ======  ======  ======  ======  ======  # 1ms  
# 1 0x1E33FEA9BC0  814216.0925  99.6% 970941  0.8386  0.09  SELECT tbname  # 10ms  
# MISC 0xMISC      2913.2860  0.4% 11632  0.2505  0.0 <16 ITEMS>  # 100ms #####  
# Tables  
# SHOW TABLE STATUS FROM `dbname` LIKE `tbname`\G  
# SHOW CREATE TABLE `dbname`.`tbname`\G  
# EXPLAIN /*!50100 PARTITIONS*/  
select id, YYYY from tbname where X = X order by id desc limit 200\G;
```

哪些是重点?

- 1、Rank
- 2、Response time
- 3、Rows examine
- 4、min

## 二、SQL 与查询优化

### 监控



监控三要素：

- 1、采集间隔
- 2、指标计算方法，最大值、最小值、平均值
- 3、数据来源

读出的信息：

- 1、TPS和延迟异常飙升
- 2、两指标的危害？持续延迟可能导致高可用失效
- 3、两指标的依赖关系？TPS推升导致延迟升高
- 4、推到出什么性能瓶颈？TPS15K时目前架构的性能瓶颈

# 监控

Orzdba

## oltp\_insert(并发线程数相同)

cpu-usage				swap				QPS				TPS				Hit%				innodb rows status				threads				bytes			
time	[usr	sys	idl	iow]	si	so]	ins	upd	del	sel	iud]	lor	hit]	ins	upd	del	read	run	con	cre	cac]	recv	send]								
16:04:18	33	11	56	0	0	0	52992	0	0	3	52992	754785	100.00	52983	0	0	0	65	65	0	1	12.5m	1.0m								
16:04:19	33	11	56	0	0	0	48337	0	0	3	48337	693143	100.00	48333	0	0	0	65	65	0	1	11.4m	951k								
16:04:20	32	11	57	0	0	0	49229	0	0	3	49229	702590	100.00	49221	0	0	0	64	65	0	1	11.6m	968k								

## oltp\_delete

cpu-usage				swap				QPS				TPS				Hit%				innodb rows status				threads				bytes			
time	[usr	sys	idl	iow]	si	so]	ins	upd	del	sel	iud]	lor	hit]	ins	upd	del	read	run	con	cre	cac]	recv	send]								
16:09:55	20	9	67	4	0	0	0	0	0	46861	3	46861	669528	99.23	0	0	19534	19535	65	65	0	1	1.1m	560k							
16:09:56	17	9	71	3	0	0	0	0	0	41133	3	41133	552182	99.19	0	0	17126	17126	65	65	0	1	966k	520k							
16:09:57	18	9	70	4	0	0	0	0	0	42850	3	42850	571911	99.16	0	0	17918	17918	65	65	0	1	1006k	497k							

## oltp\_update\_non\_index

cpu-usage				swap				QPS				TPS				Hit%				innodb rows status				threads				bytes			
time	[usr	sys	idl	iow]	si	so]	ins	upd	del	sel	iud]	lor	hit]	ins	upd	del	read	run	con	cre	cac]	recv	send]								
16:15:23	30	15	49	5	0	0	0	102723	0	3	102723	843626	99.46	0	38274	0	38274	61	65	0	1	14.1m	5.1m								
16:15:24	28	15	52	5	0	0	0	98394	0	3	98394	810753	99.46	0	36603	0	36603	62	65	0	1	13.5m	4.9m								
16:15:26	30	14	51	5	0	0	0	95284	0	3	95284	788117	99.48	0	35534	0	35534	59	65	0	1	13.1m	4.8m								

## oltp\_update\_index

cpu-usage				swap				QPS				TPS				Hit%				innodb rows status				threads				bytes			
time	[usr	sys	idl	iow]	si	so]	ins	upd	del	sel	iud]	lor	hit]	ins	upd	del	read	run	con	cre	cac]	recv	send]								
16:19:31	31	13	54	3	0	0	0	79907	0	3	79907	1212343	99.85	0	29652	0	29661	61	65	0	1	1.8m	4.0m								
16:19:32	30	12	54	3	0	0	0	77709	0	3	77709	1184309	99.85	0	29032	0	29026	59	65	0	1	1.8m	3.9m								
16:19:33	30	14	53	4	0	0	0	83050	0	3	83050	1264000	99.85	1	31112	1	31114	62	65	0	1	1.9m	4.2m								

## select\_random\_points

cpu-usage				swap				QPS				TPS				Hit%				innodb rows status				threads				bytes			
time	[usr	sys	idl	iow]	si	so]	ins	upd	del	sel	iud]	lor	hit]	ins	upd	del	read	run	con	cre	cac]	recv	send]								
08:02:59	88	6	5	0	0	0	0	0	0	61314	0	7795255	99.99	0	0	0	168307	64	65	0	0	5.7m	43.7m								
08:03:00	88	6	5	0	0	0	0	0	0	61503	0	7804784	99.99	0	0	0	167540	65	65	0	0	5.7m	43.6m								
08:03:01	88	7	5	0	0	0	0	0	0	62061	0	7853327	99.99	0	0	0	169703	65	65	0	0	5.7m	44.1m								

## oltp\_read\_only

cpu-usage				swap				QPS				TPS				Hit%				innodb rows status				threads				bytes			
time	[usr	sys	idl	iow]	si	so]	ins	upd	del	sel	iud]	lor	hit]	ins	upd	del	read	run	con	cre	cac]	recv	send]								
18:02:42	71	27	2	0	0	0	0	0	0	228472	0	1932862	99.99	0	0	0	2551798	61	65	0	0	6.1m	240.3m								
18:02:43	71	27	1	0	0	0	0	0	0	232214	0	1964946	99.99	0	0	0	2600169	59	65	0	0	6.2m	244.9m								
18:02:44	72	27	1	0	0	0	0	0	0	230389	0	1949755	99.99	0	0	0	2585775	64	65	0	0	6.2m	243.2m								

## 二、SQL 与查询优化

### 解决方案-3

需求2的查询是慢查询

增加索引：

```
alter table table_name add index index_name (column_list);
```

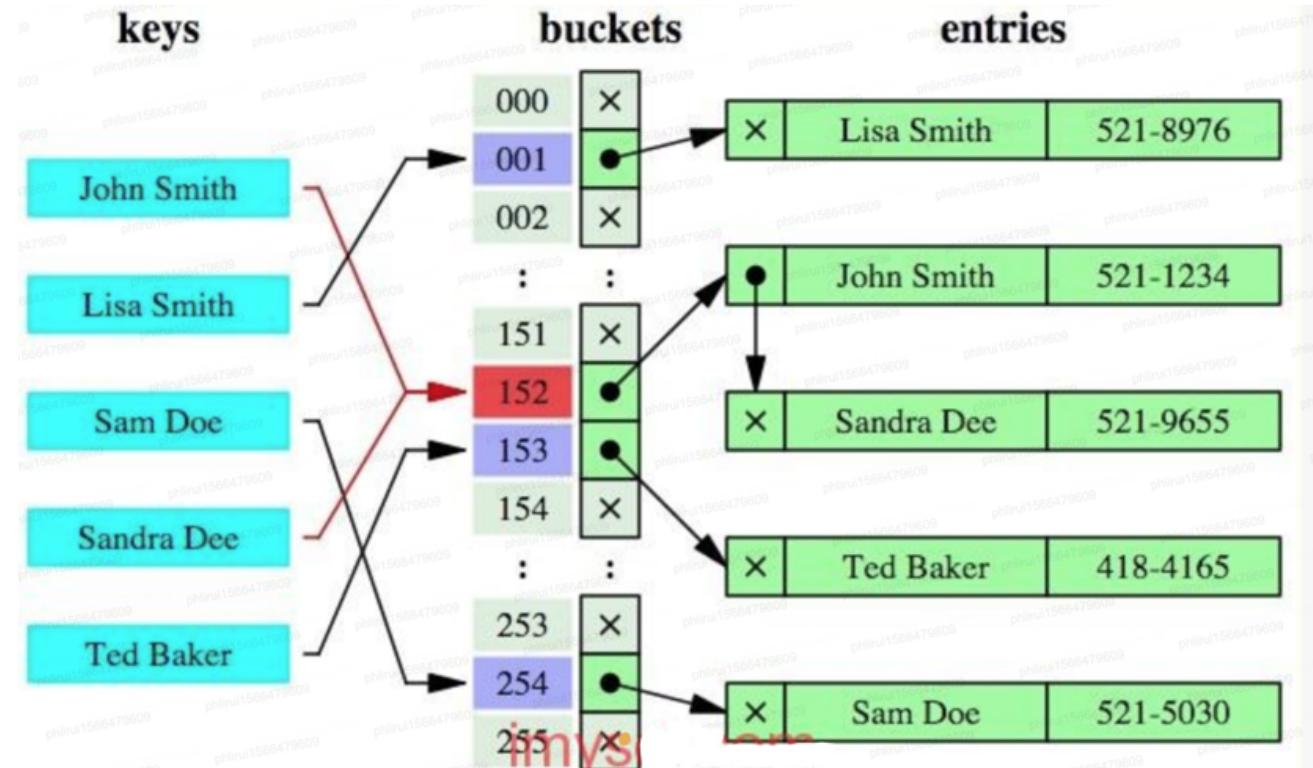
## 二、SQL 与查询优化

### 索引的类型

- Hash
- B-Tree/B+Tree

## 二、SQL 与查询优化

### Hash Index



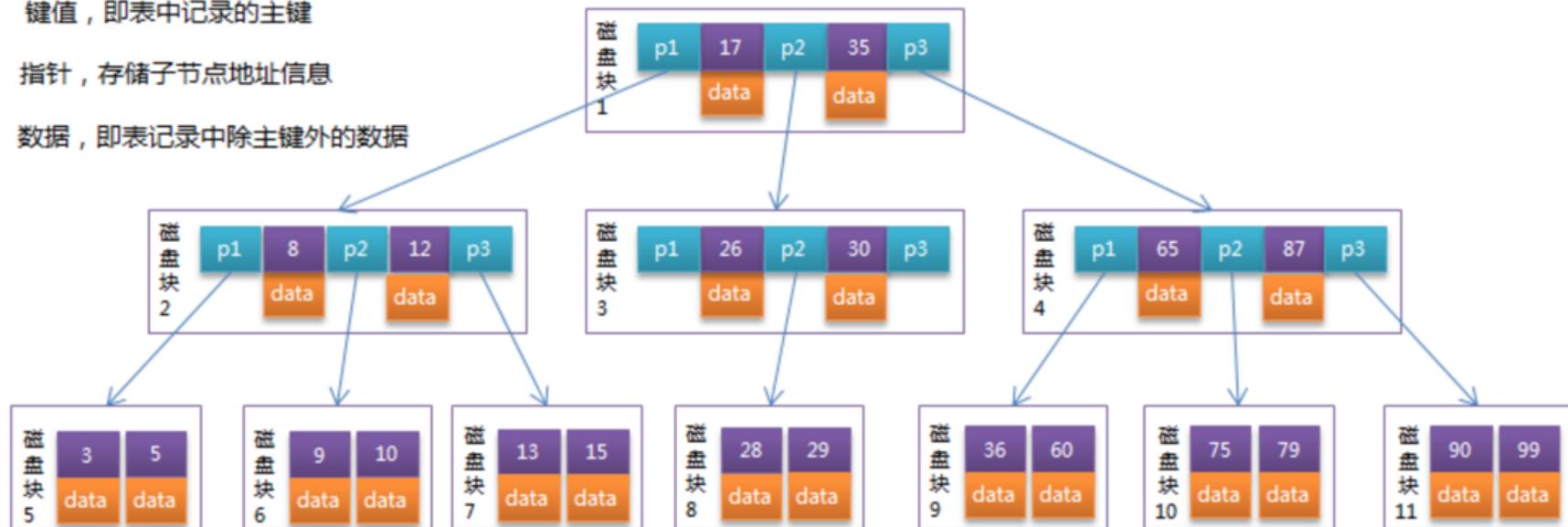
## 二、SQL 与查询优化

### B-Tree

键值，即表中记录的主键

指针，存储子节点地址信息

data，即表记录中除主键外的数据



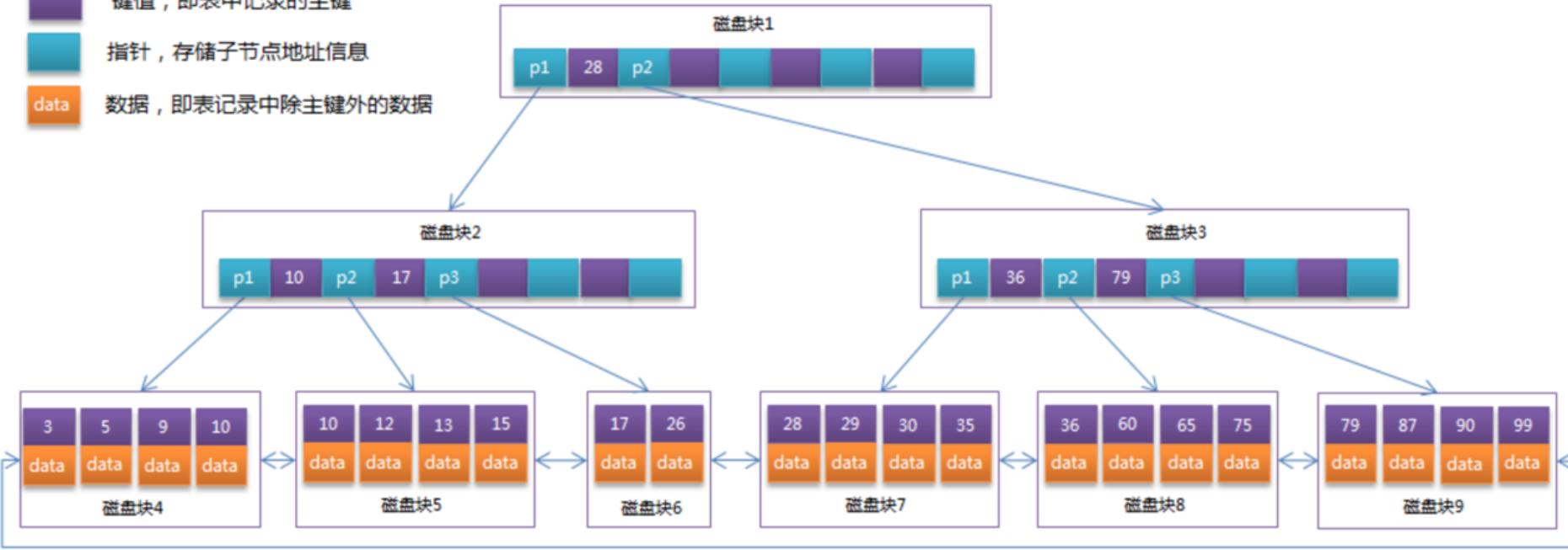
## 二、SQL 与查询优化

### B+Tree

键值，即表中记录的主键

指针，存储子节点地址信息

data 数据，即表记录中除主键外的数据



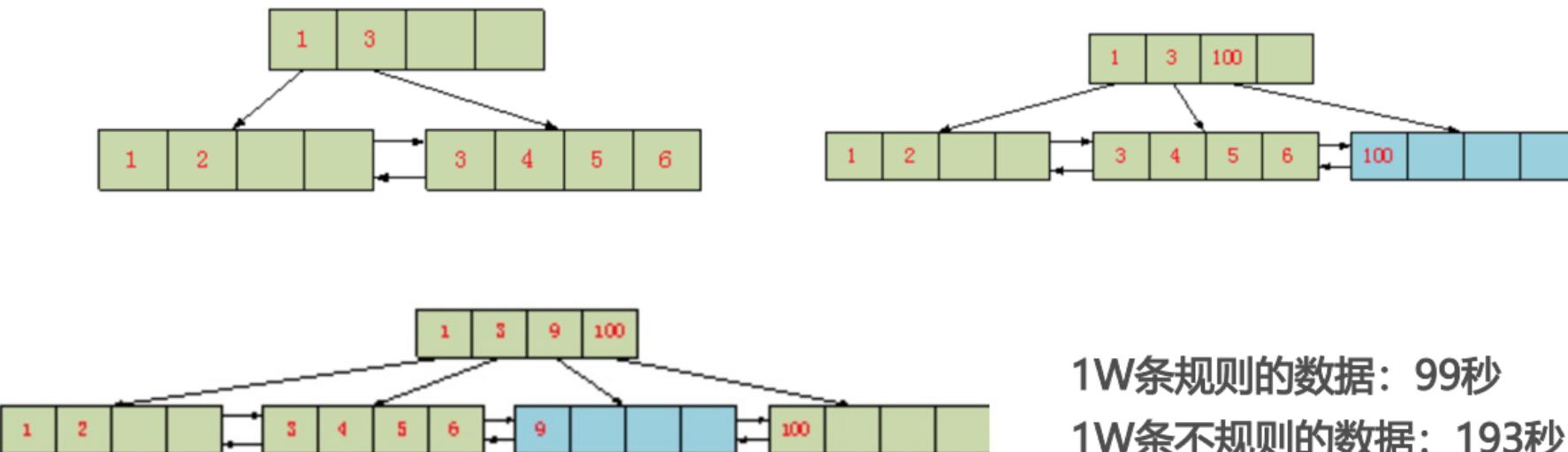
## 二、SQL 与查询优化

一个老生常谈的问题

为什么主键要单调递增？

## 二、SQL 与查询优化

### 页分裂



1W条规则的数据：99秒

1W条不规则的数据：193秒

## 二、SQL 与查询优化

### 索引思考题

- 为什么不使用hash index
- 为什么b+tree更适合做索引
- 为什么主键长度不能过大

## 二、SQL 与查询优化

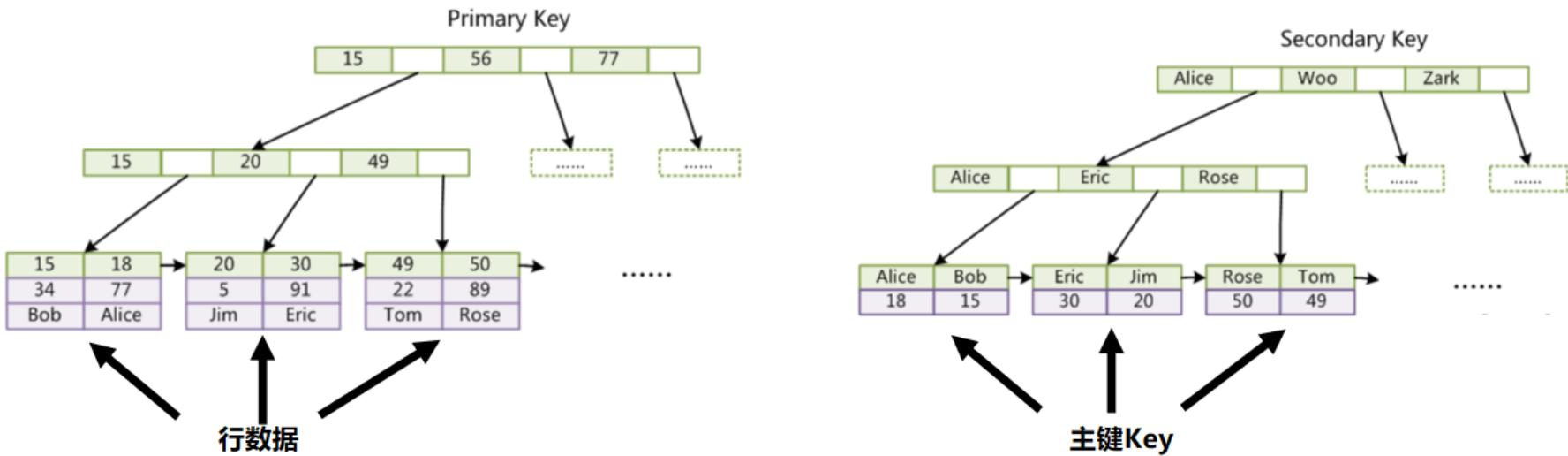
### 谁快

- `select * from t_user_info where f_id = XXX // f_id: primary key`
- `select * from t_user_info where f_user_name = 'XXX' // f_user_name: index`

哪个快?                  为什么?

## 二、SQL 与查询优化

### 聚集索引和二级索引



## 二、SQL 与查询优化

### 字段选择性-最左原则

- 某个字段其值的重复程度，称为该字段的选择性
- $F = \text{DISTINCT}(\text{col}) / \text{count}(*)$

Identity_no	Name	Gender
3301021xxxx080312xx	张三	男
3403021xxxx112332xx	李四	男
2302011xxxx071111xx	王五	女
4301331xxxx032112xx	张三	男

选择性极好

选择性较好

选择性很差

## 二、SQL 与查询优化

### 索引冗余

(username, name, age) : (username)、(username, name)

- (username, name) : (username) 、 (name, username) 、 (name)
- (username) : (username, id)

## 二、SQL 与查询优化

### 修改表结构的危害

1. 索引重建
2. 锁表
3. 抢占资源
4. 主从延时

## 二、SQL 与查询优化

### 数据量



- 业务初期考虑不周，字段类型使用不合理，需要变更数据类型
- 随着业务的发展，需要增加新的字段
- 在无索引字段增加新的业务查询，需要增加索引

## 二、SQL 与查询优化

大批量写入的优化

PreparedStatement 减少 SQL 解析

Multiple Values/Add Batch 减少交互

Load Data , 直接导入

索引和约束问题

## 二、SQL 与查询优化

Like 的问题

前缀匹配

否则不走索引

## 二、SQL 与查询优化

数据的范围更新

注意 GAP Lock 的问题

导致锁范围扩大

## 二、SQL 与查询优化

连接查询优化

驱动表的选择问题

## 二、SQL 与查询优化

索引失效的情况汇总

NULL , not , not in , 函数等

减少使用 or , 可以用 union ( 注意 union all 的区别 ) , 以及前面提到的 like

大数据量下 , 放弃所有条件组合都走索引的幻想 , 出门左拐 “全文检索”

必要时可以使用 force index 来强制查询走某个索引

## 二、SQL 与查询优化

查询数据量和查询次数的平衡

避免不必要的大量重复数据传输

避免使用临时文件排序或临时表

## 二、SQL 与查询优化

分布式主键的生成问题

UUID

时间戳，随机数

Sequence

模拟 Seq

Snowflake

精确递增有什么缺点

分别适用什么场景

## 二、SQL 与查询优化

分页的最佳实践

常规误区：只写一个复杂的、甚至是所有条件任意组合的 selectSQL : select \* from N 个表。

然后 select count(1) from (selectSQL)

问题在哪里？

分页计算当前页的 start 和 offset , select \* from tablename limit start,offset

问题在哪里？

理想实现法：

- 1、select count 单独写，只查驱动表
- 2、判断正序或倒序
- 3、大数据量非精确分页

## 二、SQL 与查询优化

使用乐观锁代替悲观锁

注意避免 ABA 的问题

THANKS! |  极客大学