

极客大学 Java 进阶训练营

第 13 课

性能与 SQL 优化 (2)



KimmKing

Apache Dubbo/ShardingSphere PMC



个人介绍

Apache Dubbo/ShardingSphere PMC

前某集团高级技术总监/阿里架构师/某银行北京研发中心负责人

阿里云 MVP、腾讯 TVP、TGO 会员

10 多年研发管理和架构经验

熟悉海量并发低延迟交易系统的设计实现

目录

1. MySQL 事务与锁*
2. DB 与 SQL 优化*
3. 常见场景分析*
4. 总结回顾与作业实践

1. MySQL 事务与锁

MySQL 事务

事务可靠性模型 ACID:

- **Atomicity**: 原子性，一次事务中的操作要么全部成功，要么全部失败。
- **Consistency**: 一致性，跨表、跨行、跨事务，数据库始终保持一致状态。
- **Isolation**: 隔离性，可见性，保护事务不会互相干扰，包含4种隔离级别。
- **Durability**: 持久性，事务提交成功后，不会丢数据。如电源故障，系统崩溃。

性能 VS. 可靠性

InnoDB:

双写缓冲区、故障恢复、操作系统、fsync()、磁盘存储、缓存、UPS、网络、备份策略

MySQL 事务

表级锁

意向锁: 表明事务稍后要进行哪种类型的锁定

- **共享意向锁 (IS)** : 打算在某些行上设置共享锁
- **排他意向锁 (IX)** : 打算对某些行设置排他锁
- **Insert 意向锁**: Insert 操作设置的间隙锁

其他:

- 自增锁 (AUTO-INC)
- LOCK TABLES/DDL

上锁前需要先上意向锁!

锁类型的兼容性

	X	IX	S	IS
X		冲突	冲突	冲突
IX	冲突		兼容	冲突
S	冲突	冲突		兼容
IS	冲突	兼容	兼容	

• **共享锁(S)**

• **排他锁(X)**

SHOW ENGINE INNODB STATUS;

MySQL 事务

行级锁 (InnoDB)

- 记录锁 (Record) : 始终锁定索引记录, 注意隐藏的聚簇索引
- 间隙锁 (Gap) : 锁住一个范围
- 临键锁 (Next-Key) : 记录锁+间隙锁的组合; 可 “锁定” 表中不存在记录
- 谓词锁 (Predicat) : 空间索引

死锁:

- 阻塞与互相等待
- 增删改、锁定读
- 死锁检测与自动回滚
- 锁粒度与程序设计

MySQL 事务

《SQL:1992标准》规定了四种事务隔离级别(Isolation):

- 读未提交: READ UNCOMMITTED
- 读已提交: READ COMMITTED
- 可重复读: REPEATABLE READ
- 可串行化: SERIALIZABLE

隔离级别	并发性
	可靠性
	一致性
	可重复性

事务隔离是数据库的基础特征。

MySQL:

- 可以设置全局的默认隔离级别
- 可以单独设置会话的隔离级别
- InnoDB 实现与标准之间的差异

MySQL 事务

读未提交: READ UNCOMMITTED

- 很少使用
- 不能保证一致性
- 脏读 (dirty read) : 使用到从未被确认的数据 (例如: 早期版本、回滚)

锁:

- 以非锁定方式执行
- 可能的问题: 脏读、幻读、不可重复读

MySQL 事务

读已提交: READ COMMITTED

- 每次查询都会设置和读取自己的新快照。
- 仅支持基于行的 bin-log
- UPDATE 优化: 半一致读 (semi-consistent read)
- 不可重复读: 不加锁的情况下, 其他事务 UPDATE 或 DELETE 会对查询结果有影响
- 幻读 (Phantom) : 加锁后, 不锁定间隙, 其他事务可以 INSERT

锁:

- 锁定索引记录, 而不锁定记录之间的间隙
- 可能的问题: 幻读、不可重复读

MySQL 事务

可重复读: REPEATABLE READ

- InnoDB 的默认隔离级别
- 使用事务第一次读取时创建的快照
- 多版本技术

锁:

- 使用唯一索引的唯一查询条件时, 只锁定查找到的索引记录, 不锁定间隙。
- 其他查询条件, 会锁定扫描到的索引范围, 通过间隙锁或临键锁来阻止其他会话在这个范围中插入值。
- 可能的问题: InnoDB 不能保证没有幻读, 需要加锁

MySQL 事务

串行化: SERIALIZABLE

最严格的级别，事务串行执行，资源消耗最大；

问题回顾:

- 脏读 (dirty read) : 使用到从未被确认的数据 (例如: 早期版本、回滚)
- 不可重复读(unrepeated read): 不加锁的情况下，其他事务 update 会对结果集有影响
- 幻读 (phantom read) : 相同的查询语句，在不同的时间点执行时，产生不同的结果集不同

怎么解决?

提高隔离级别、使用间隙锁或临键锁

MySQL 事务

undo log: 撤消日志

- 保证事务的原子性
- 用处: 事务回滚, 一致性读、崩溃恢复。
- 记录事务回滚时所需的撤消操作
- 一条 INSERT 语句, 对应一条 DELETE 的 undo log
- 每个 UPDATE 语句, 对应一条相反 UPDATE 的 undo log

保存位置:

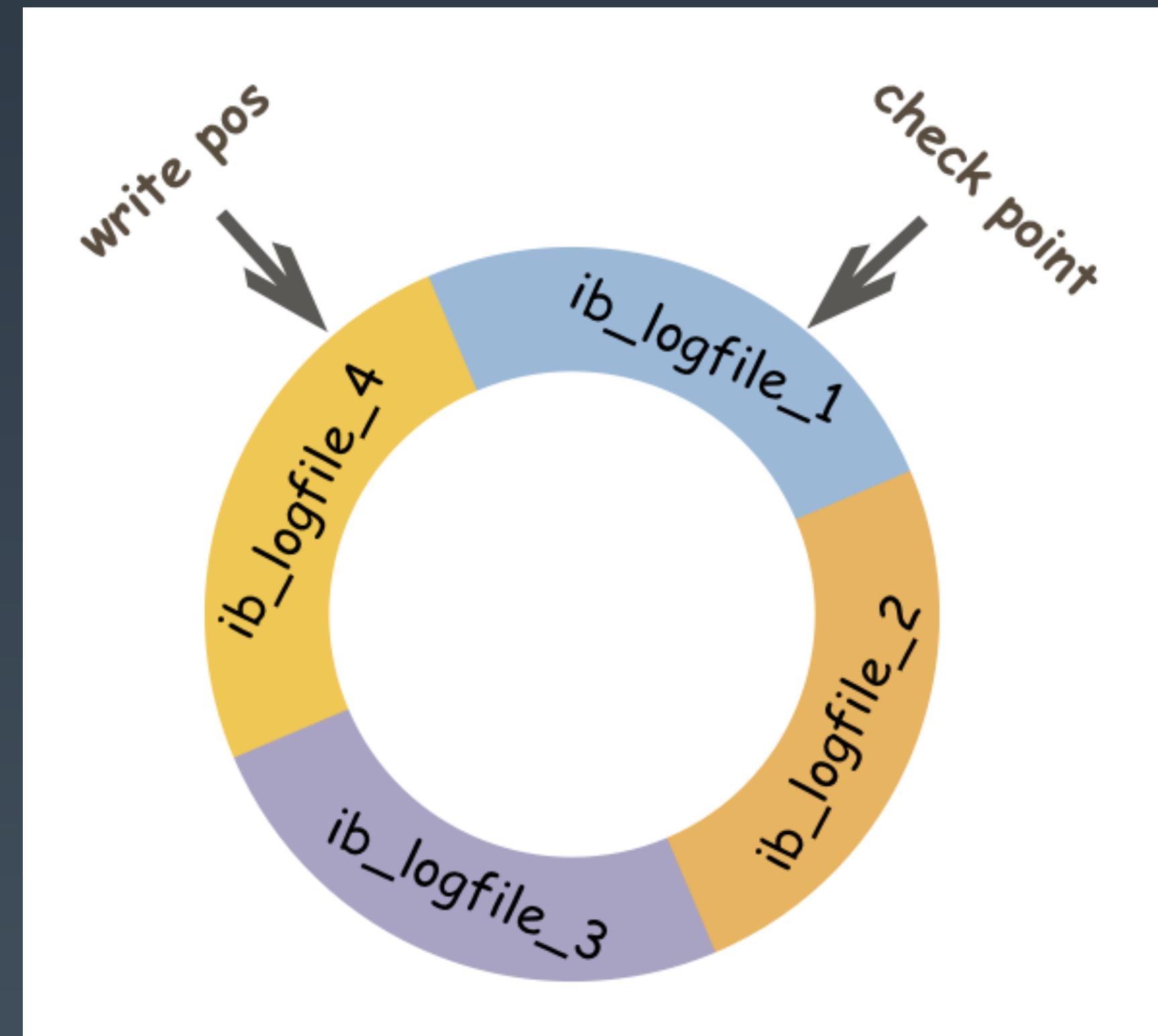
- system tablespace (MySQL 5.7 默认)
- undo tablespaces (MySQL 8.0 默认)

回滚段 (rollback segment)

MySQL 事务

redo log: 重做日志

- 确保事务的持久性，防止事务提交后数据未刷新到磁盘就掉电或崩溃。
- 事务执行过程中写入 redo log，记录事务对数据页做了哪些修改。
- 提升性能：WAL (Write-Ahead Logging) 技术，先写日志，再写磁盘。
- 日志文件：ib_logfile0, ib_logfile1
- 日志缓冲：innodb_log_buffer_size
- 强刷：fsync()



MySQL 事务

MVCC: 多版本并发控制

- 使 InnoDB 支持一致性读: READ COMMITTED 和 REPEATABLE READ
- 让查询不被阻塞、无需等待被其他事务持有的锁, 这种技术手段可以增加并发性能
- InnoDB 保留被修改行的旧版本
- 查询正在被其他事务更新的数据时, 会读取更新之前的版本
- 每行数据都存在一个版本号, 每次更新时都更新该版本
- 这种技术在数据库领域的使用并不普遍。某些数据库, 以及某些 MySQL 存储引擎都不支持

聚簇索引的更新 = 替换更新

二级索引的更新 = 删除+新建

MySQL 事务

MVCC 实现机制

- 隐藏列
- 事务链表，保存还未提交的事务，事务提交则会从链表中摘除
- Read view: 每个 SQL 一个，包括 `rw trx_ids`, `low_limit_id`, `up_limit_id`, `low_limit_no` 等
- 回滚段: 通过 `undo log` 动态构建旧版本数据

隐藏列:	DB_TRX_ID	DB_ROLL_PTR	DB_ROW_ID
长度:	6-byte	7-byte	6-byte
说明:	指示最后插入或更新该行的事务 ID	回滚指针。指向回滚段中写入的 <code>undo log</code> 记录	聚簇 row ID / 聚簇索引

MySQL 事务

演示事务与锁

2.DB 与 SQL 优化

从一个简单例子讲起

```
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   |
+-----+-----+-----+-----+-----+
```

从一个简单例子讲起

```
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)|
```

说说 SQL 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?

模拟一个需求：版本1

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，为改项目组提供技术支持，并且验证方案可行性

模拟一个需求

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

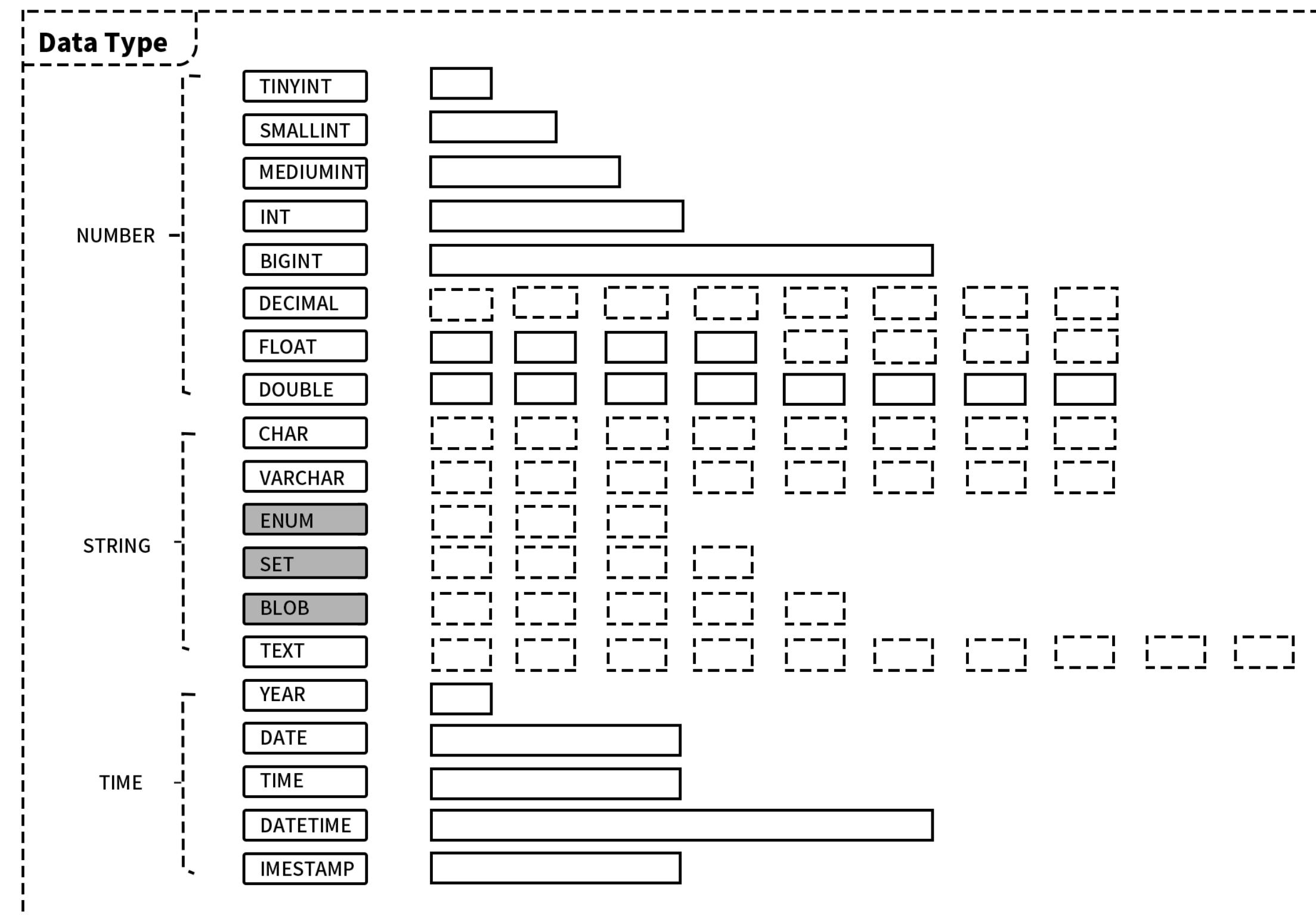
性别 gender

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：监管要求储存5年

1个数据类型能节省多少\$？

继续深入。 . .

存储引擎的选择

InnoDB

1. 聚焦索引
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，尤其适用于压缩和归档 (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

继续深入。 . .

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';
```

继续深入。 . .

小结-1

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

继续深入。。。需求升级到版本2

需求-2

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

继续深入。 . .

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录

```
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 ;
```

继续深入。 . .

隐式转换

```
SELECT f_id , f_use name , 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

继续深入。 . .

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

继续深入。。。版本3

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。

继续深入。 . .

分析-3

定位问题的方法：

- 慢查询日志
- 看应用和运维的监控

继续深入。 o o

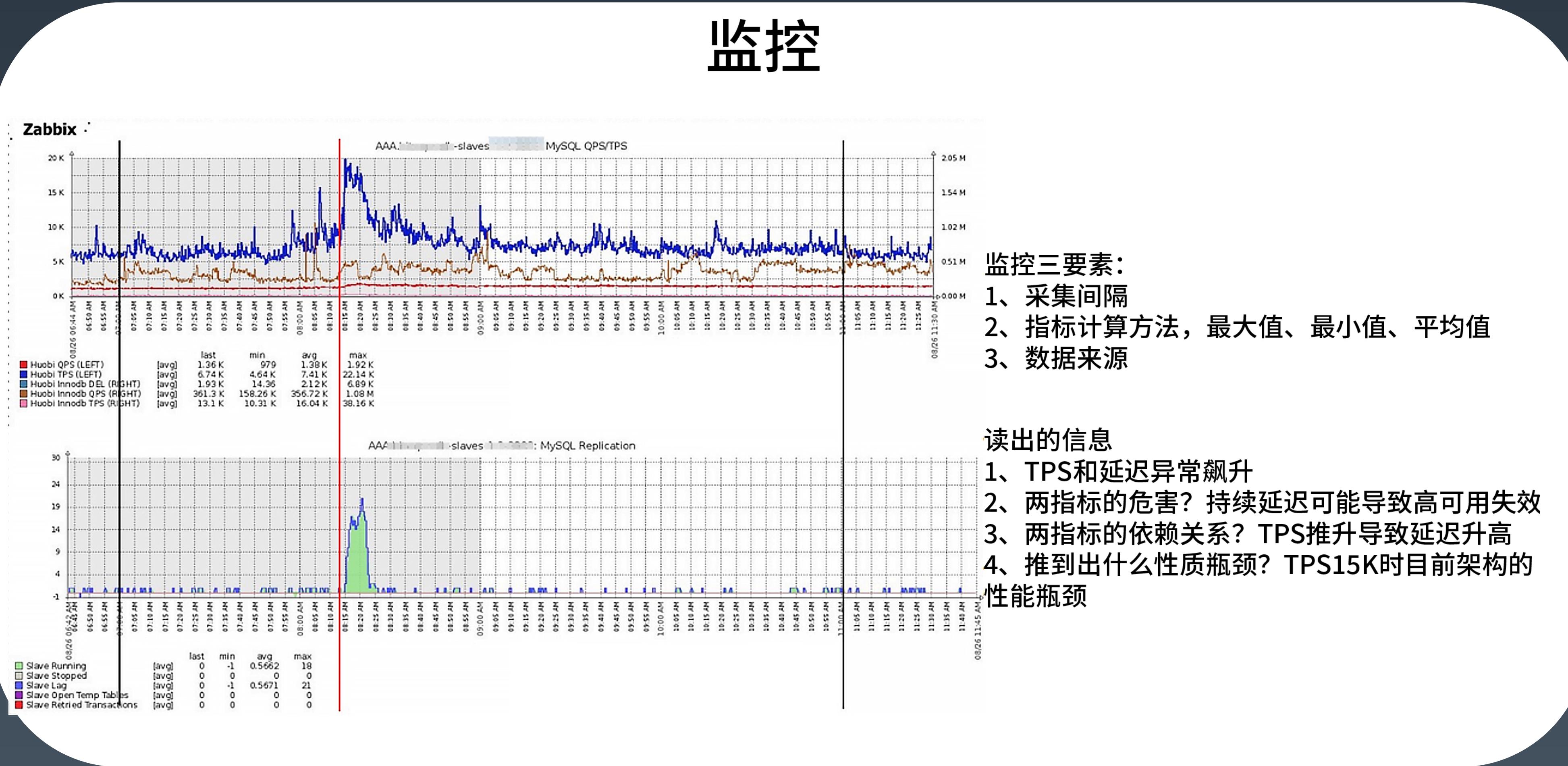
慢查询日志

哪些是重点？

1. Rank
 2. Response time
 3. Rows examine
 4. min

继续深入。 . .

监控



继续深入。 . .

监控

Orzdba

oltp_insert (并发线程数相同)

time	usr	sys	idl	iow	si	so	ins	upd	del	sel	iudl	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	529921	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	483371	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	492291	702590	100.00	49221	0	0	0	64	65	0	1	11.6m	968k

oltp_delete

time	usr	sys	idl	iow	si	so	ins	upd	del	sel	iudl	lor	hit	ins	upd	del	read	run	con	cre	cac	recv	send
16:09:55	20	9	67	4	0	0	0	46861	3	468611	669528	99.23	0	0	19534	195351	65	65	0	1	1.1m	560k	
16:09:56	17	9	71	3	0	0	0	41133	3	411331	552182	99.19	0	0	17126	171261	65	65	0	1	966k	520k	
16:09:57	18	9	70	4	0	0	0	42850	3	428501	571911	99.16	0	0	17918	179181	65	65	0	1	1006k	497k	

oltp_update_non_index

time	usr	sys	idl	iow	si	so	ins	upd	del	sel	iudl	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	1027231	843626	99.46	0	38274	0	382741	61	65	0	1	14.1m	5.1m
16:15:24	28	15	52	5	0	0	0	98394	0	3	983941	810753	99.46	0	36603	0	366031	62	65	0	1	13.5m	4.9m
16:15:26	30	14	51	5	0	0	0	95284	0	3	952841	788117	99.48	0	35534	0	355341	59	65	0	1	13.1m	4.8m

oltp_update_index

time	usr	sys	idl	iow	si	so	ins	upd	del	sel	iudl	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	799071	1212343	99.85	0	29652	0	296611	61	65	0	1	1.8m	4.0m
16:19:32	30	12	54	3	0	0	0	77709	0	3	777091	1184309	99.85	0	29032	0	290261	59	65	0	1	1.8m	3.9m
16:19:33	30	14	53	4	0	0	0	83050	0	3	830501	1264000	99.85	1	31112	1	311141	62	65	0	1	1.9m	4.2m

select_random_points

time	usr	sys	idl	iow	si	so	ins	upd	del	sel	iudl	lor	hit	ins	upd	del	read	run	con	cre	cac	recv	send
08:02:59	88	6	5	0	0	0	0	0	61314	0	7795255	99.99	0	0	0	1683071	64	65	0	0	5.7m	43.7m	
08:03:00	88	6	5	0	0	0	0	0	61503	0	7804784	99.99	0	0	0	1675401	65	65	0	0	5.7m	43.6m	
08:03:01	88	7	5	0	0	0	0	0	62061	0	7853327	99.99	0	0	0	1697031	65	65	0	0	5.7m	44.1m	

oltp_read_only

time	usr	sys	idl	iow	si	so	ins	upd	del	sel	iudl	lor	hit	ins	upd	del	read	run	con	cre	cac	recv	send
18:02:42	71	27	2	0	0	0	0	0	228472	0	1932862	99.99	0	0	0	25517981	61	65	0	0	6.1m	240.3m	
18:02:43	71	27	1	0	0	0	0	0	232214	0	1964946	99.99	0	0	0	26001691	59	65	0	0	6.2m	244.9m	
18:02:44	72	27	1	0	0	0	0	0	230389	0	1949755	99.99	0	0	0	25857751	64	65	0	0	6.2m	243.2m	

继续深入。 . .

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);
```

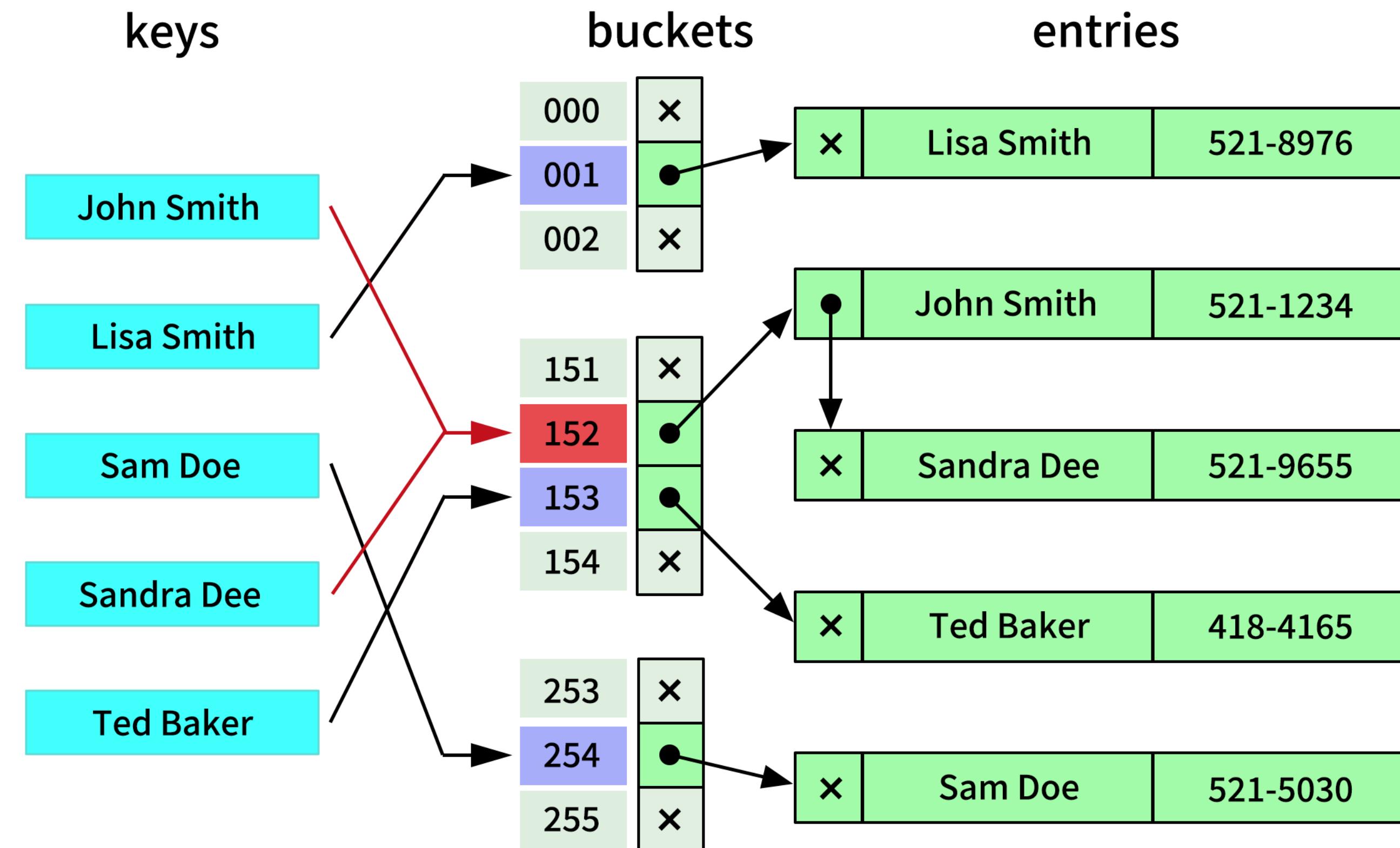
继续深入。 . .

索引的类型

- Hash
- B-Tree/ B+Tree

继续深入。 . .

Hash Index



继续深入。 . .

B-Tree



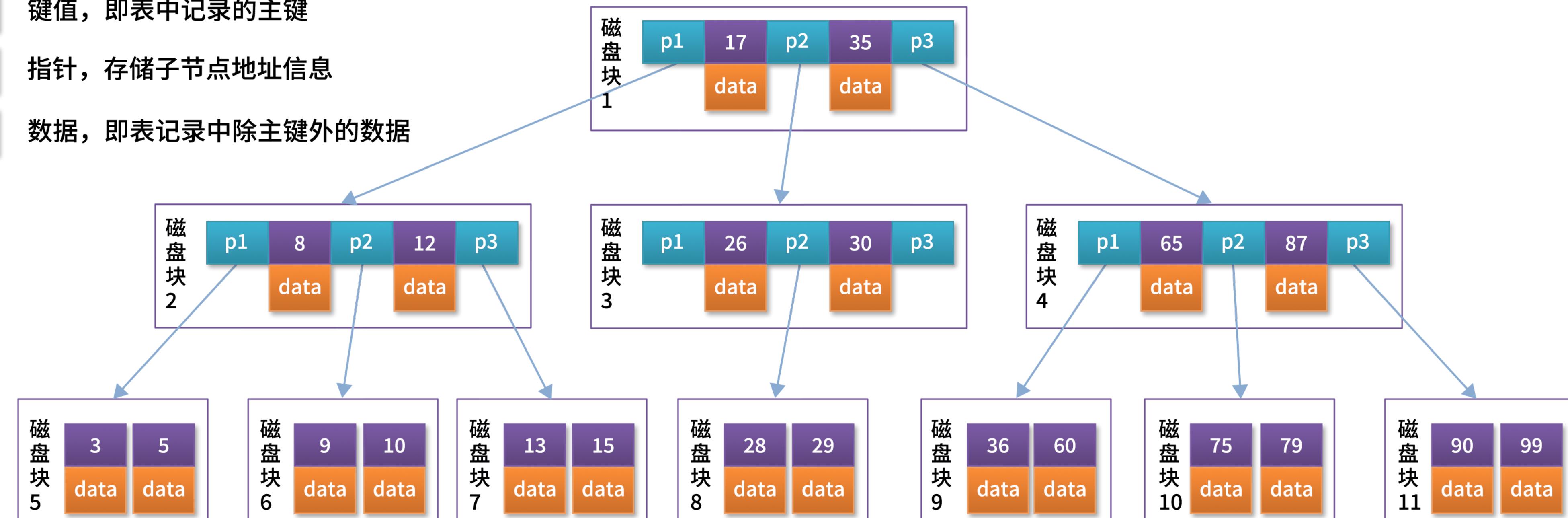
键值，即表中记录的主键



指针，存储子节点地址信息

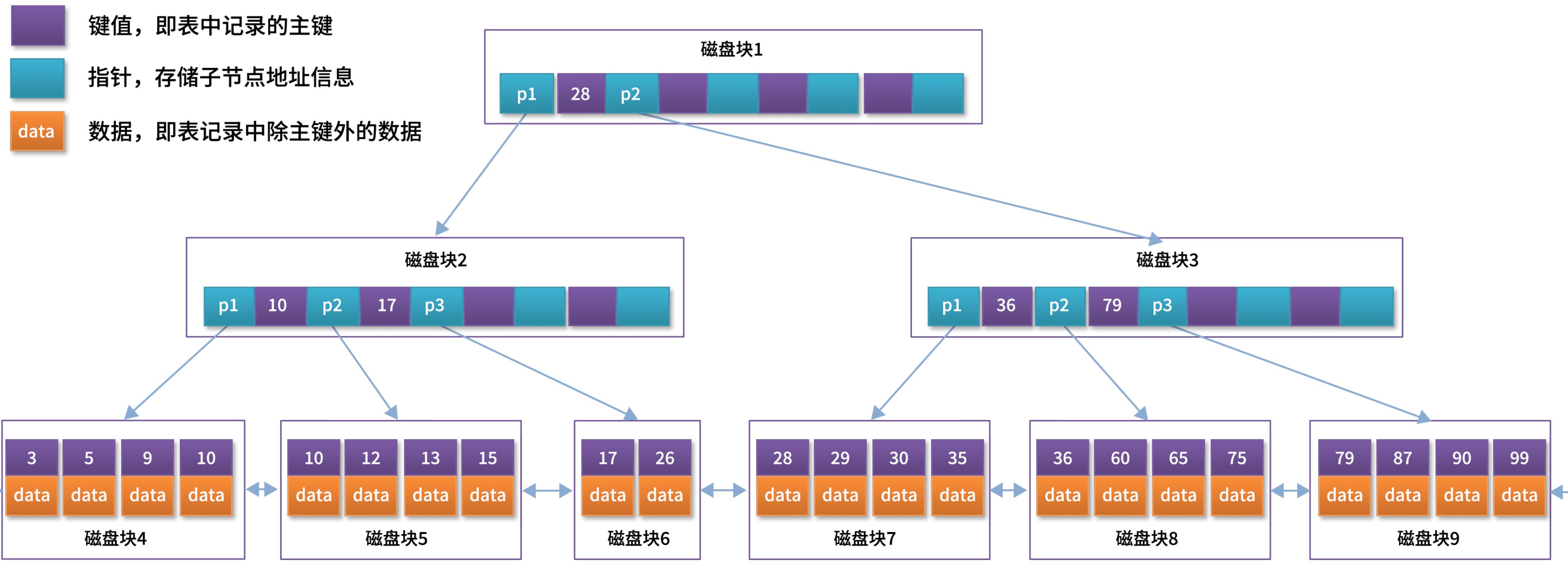


数据，即表记录中除主键外的数据



继续深入。 . .

B+Tree



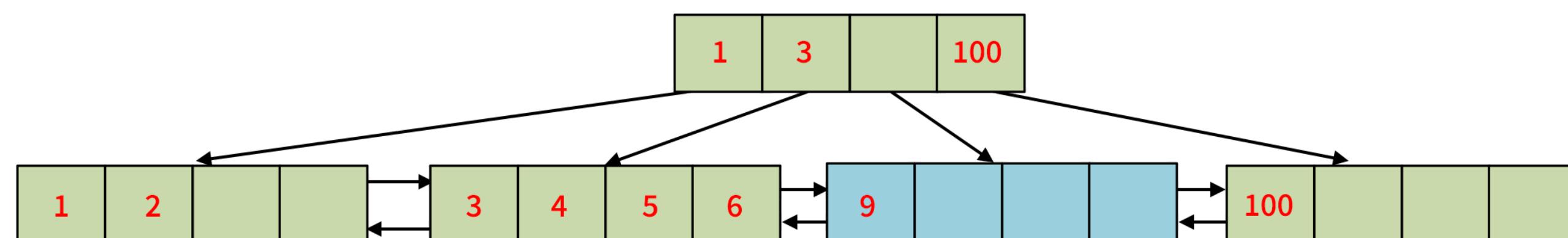
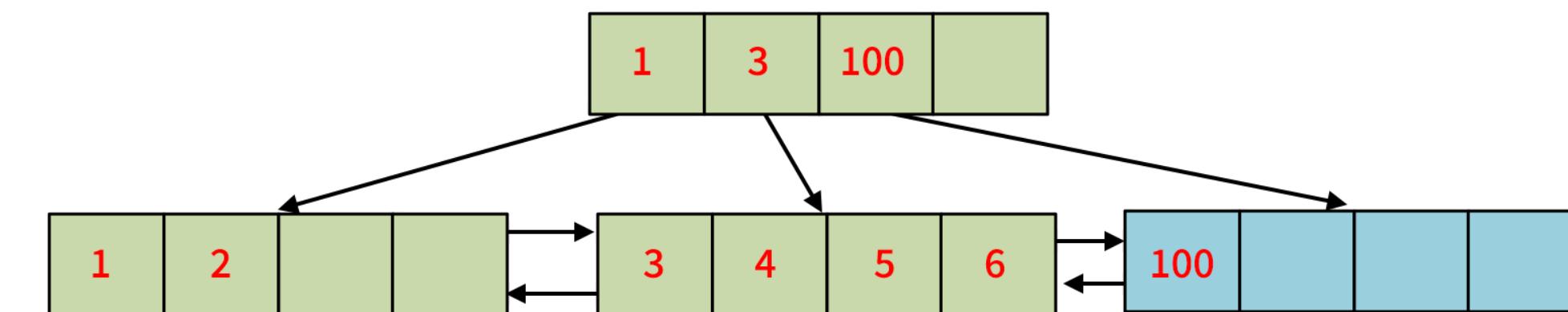
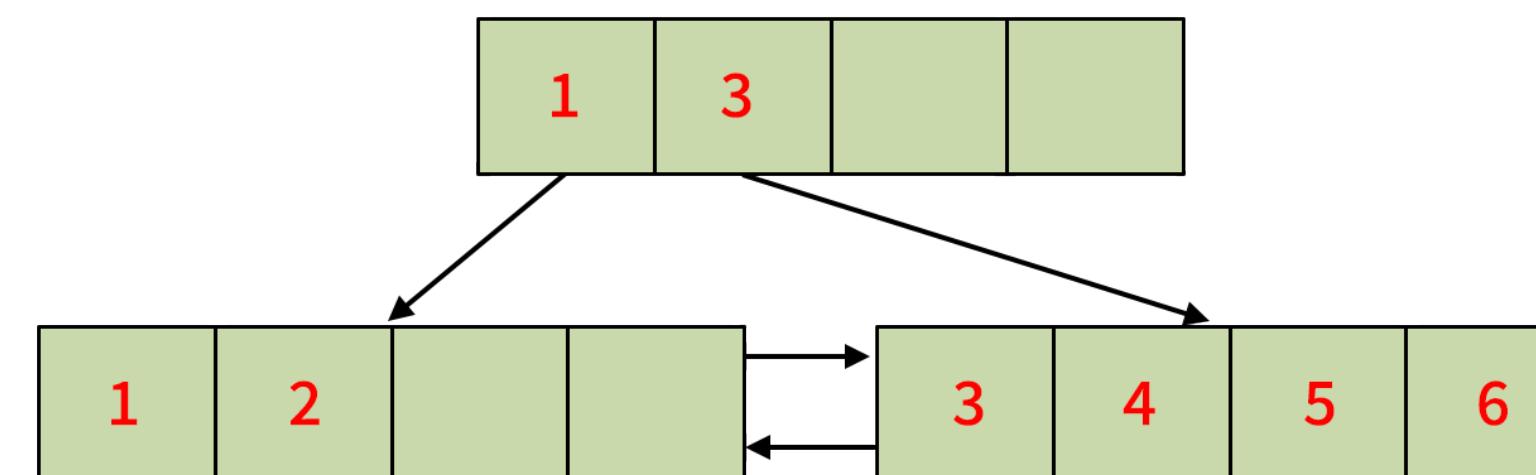
继续深入。 . .

一个老生常谈的问题

为什么主键要单调递增？

继续深入。 . .

页分裂



1W条规则的数据：99秒

1W条不规则的数据：193秒

继续深入。 . .

索引思考题

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

继续深入。 . .

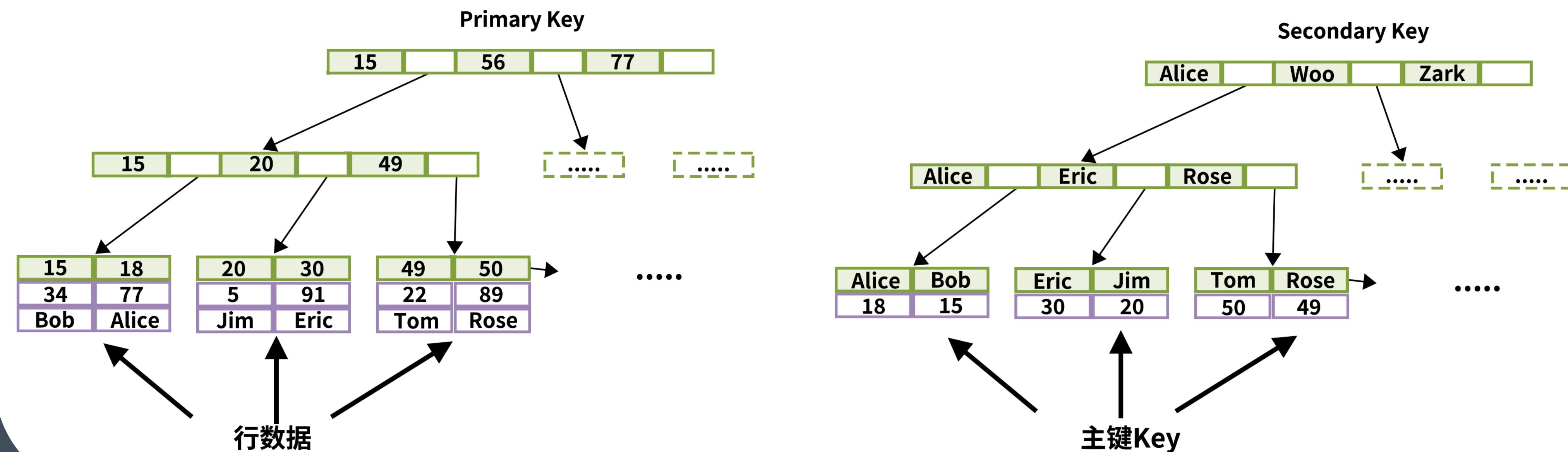
谁快

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

哪个快? 为什么?

继续深入。 . .

聚集索引和二级索引



继续深入。 . .

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
- $F = \text{DISTINCT}(\text{col})/\text{count}(\text{*})$

Identity_no	Name	Gender
3301021xxxx080312xx	张三	男
3403021xxxx112332xx	李四	男
2302021xxxx071111xx	王五	女
4301331xxxx032112xx	张三	男

选择性极好

选择性较好

选择性很差

继续深入。 . .

索引冗余

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

1) 长的包括短的，形成冗余

2) 有唯一约束的，组合冗余

继续深入。 . .

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

继续深入。 . .

数据量

1. 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
2. 随着业务的发展，需要增加新的字段
3. 在无索引字段增加新的业务查询，需要增加索引

总结1：写入优化

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

总结2：数据更新

数据的范围更新

注意 GAP Lock 的问题

导致锁范围扩大

总结3：模糊查询

Like 的问题

前缀匹配

否则不走索引

全文检索

solr/ES

总结4：连接查询

连接查询优化

驱动表的选择问题

避免笛卡尔积

总结5：索引失效

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

总结6：查询 SQL 到底怎么设计？

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

分析类需求，可以用汇总表

3. 常见场景分析

怎么实现主键 ID

- 自增
- sequence
- 模拟 seq
- UUID
- 时间戳/随机数
- snowflake

还有没有其他办法？

高效分页

- 分页: count/pageSize/pageNum, 带条件的查询语句
- 常见实现-分页插件: 使用查询 SQL, 嵌套一个 count, 性能的坑?
- 改进一下1, 重写 count
- 大数量级分页的问题, limit 100000,20
- 改进一下2, 反序
- 继续改进3, 技术向: 带 id
- 继续改进4, 需求向: 非精确分页
- 所有条件组合? 索引?

还能不能继续改进?

乐观锁与悲观锁

```
select * from xxx for update  
update xxx  
commit;
```

意味着什么？

```
select * from xxx  
update xxx where value=oldValue  
为什么叫乐观锁
```

有什么区别？

4. 总结回顾与作业实践

第 13 课总结回顾

MySQL 事务与锁

DB 与 SQL 优化

常见场景分析

第13节课作业实践

- 1、（选做）用今天课上学习的知识，分析自己系统的 SQL 和表结构
- 2、（必做）按自己设计的表结构，插入100万订单模拟数据，测试不同方式的插入效率。
- 3、（选做）按自己设计的表结构，插入1000万订单模拟数据，测试不同方式的插入效率。
- 4、（选做）使用不同的索引或组合，测试不同方式查询效率。
- 5、（选做）调整测试数据，使得数据尽量均匀，模拟1年时间内的交易，计算一年的销售报表：销售总额，订单数，客单价，每月销售量，前十的商品等等（可以自己设计更多指标）。
- 6、（选做）尝试自己做一个 ID 生成器（可以模拟 Seq 或 Snowflake）。
- 7、（选做）尝试实现或改造一个非精确分页的程序。

THANKS! |  极客大学