基于INNODB的MYSQL关键技术原理概览以及常用规范

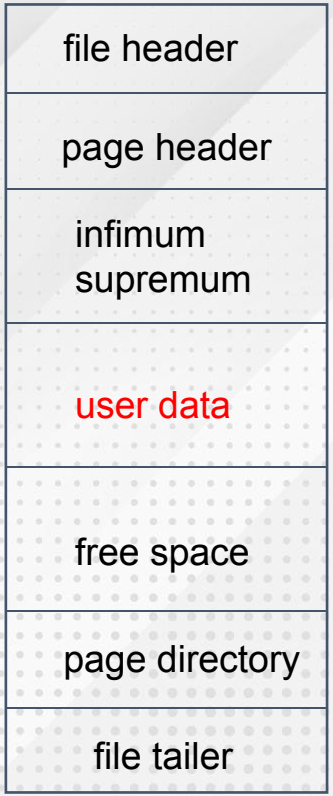
王晗

目录

CONTENTS

存储模型

物理页面



38个字节，页类型：索引页，undolog页，系统页以及事务系统数据等等

56个字节，数据页的状态以及控制信息，比如页内slot的数量以及数据的首地址等信息

26个字节，两个虚拟行记录，用于标识页的边界；最大和最小；

实际数据，数据区；

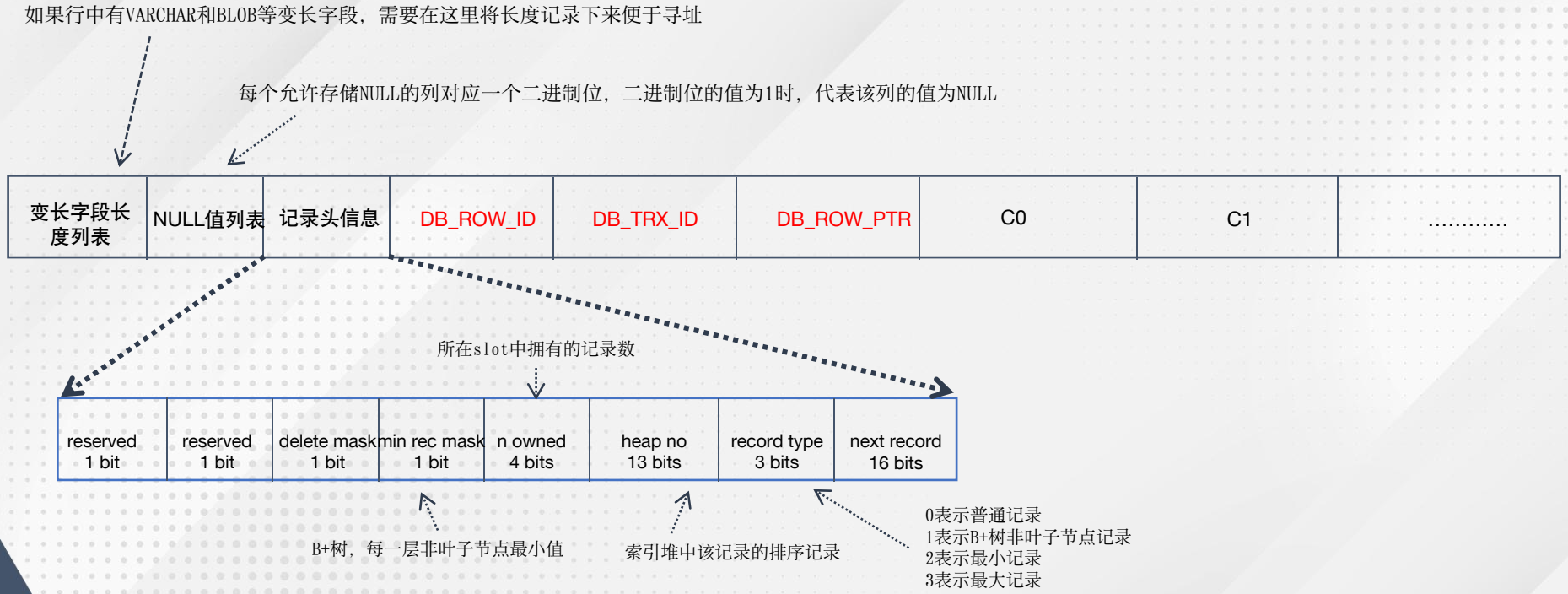
页内尚未使用的空间

页目录，对user record进行分组，通过类似于跳跃表的方式组织，我们称之为slot数组，通过二分查找提升寻址效率

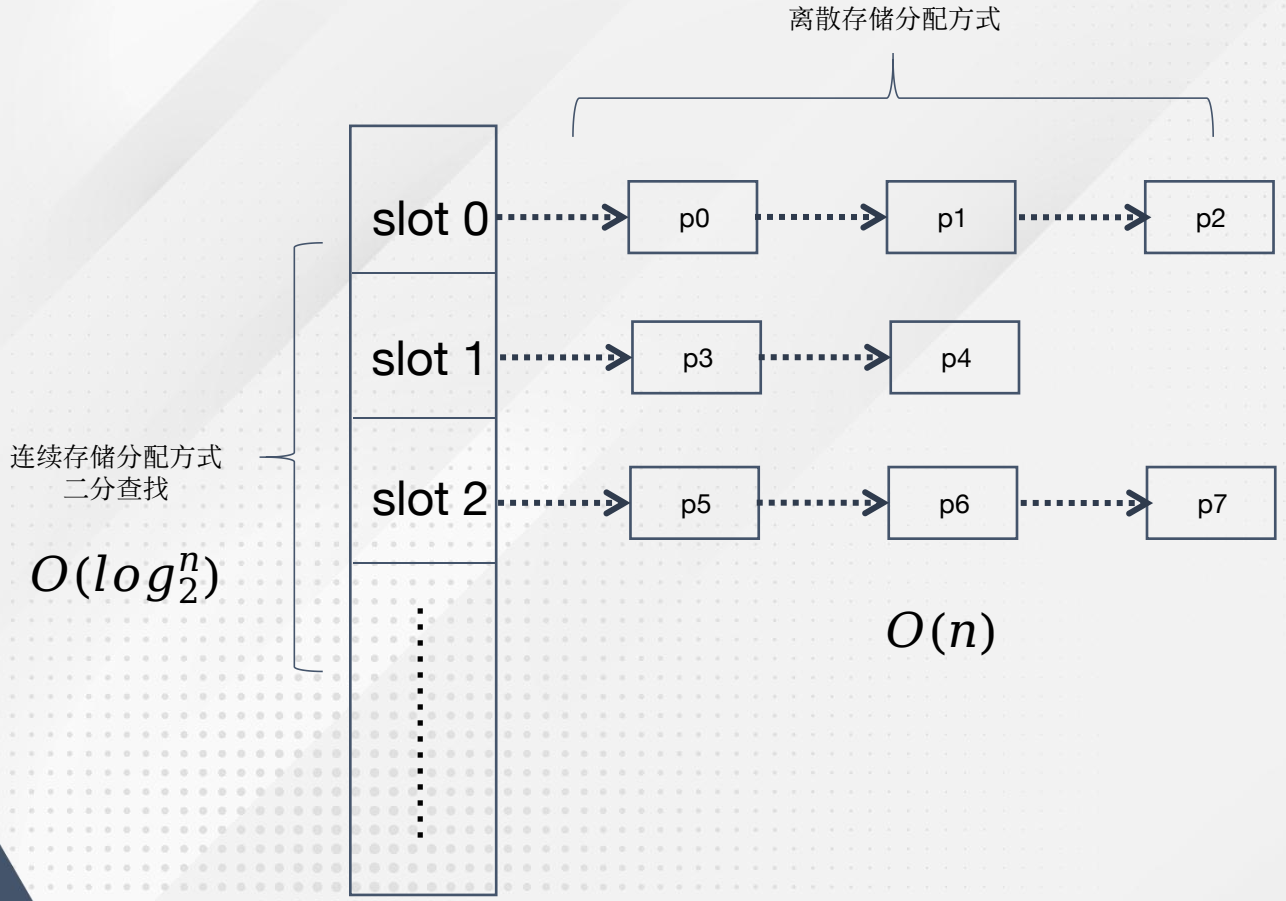
8字节，存放校验位，数据是否完整

页内数据行

设计了四种类型的行格式，包括Compact，Redundent(5.0之前)，Dynamic（5.7默认）和Compressed。



页内寻址

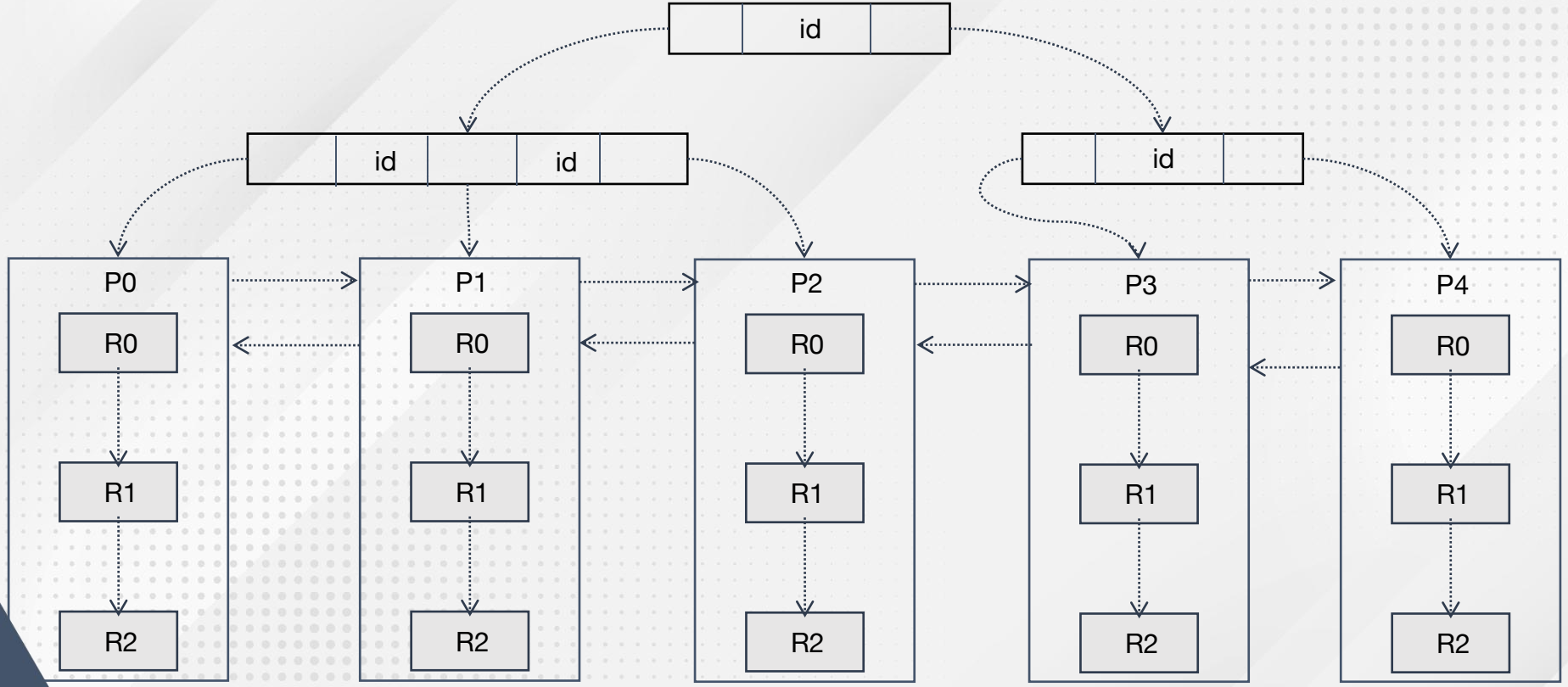

离散存储分配方式
slot 0
slot 1
slot 2
p0
p1
p2
p3
p4
p5
p6
p7
连续存储分配方式
二分查找
$O(log_2^n)$
$O(n)$

如何组织物理页面

索引组织树，聚簇索引

m阶的B+树定义

1. 每个节点至多有m个子女；
2. 除根节点外，每个结点至少有[m/2]个子女，根节点至少有两个子女；
3. 有k个子女的节点必有k个关键字；



查找

找到数据所在的页 ⇨ B+树查找，从跟节点一直到叶子节点

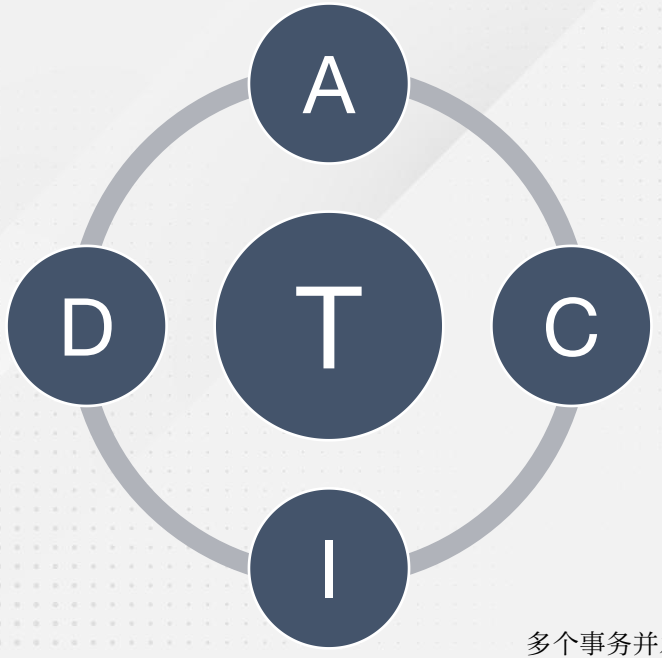
页内查找 ⇨ 页内寻址

事务机制

事务特性

操作要吗都执行，要吗都不执行

undo log



事务提交之后，会持久化结果

redo log

执行前后，数据完整性必须保持一致

redo log + undo log

多个事务并发执行，相互隔离，互不影响

Lock + MVCC

Undo Log

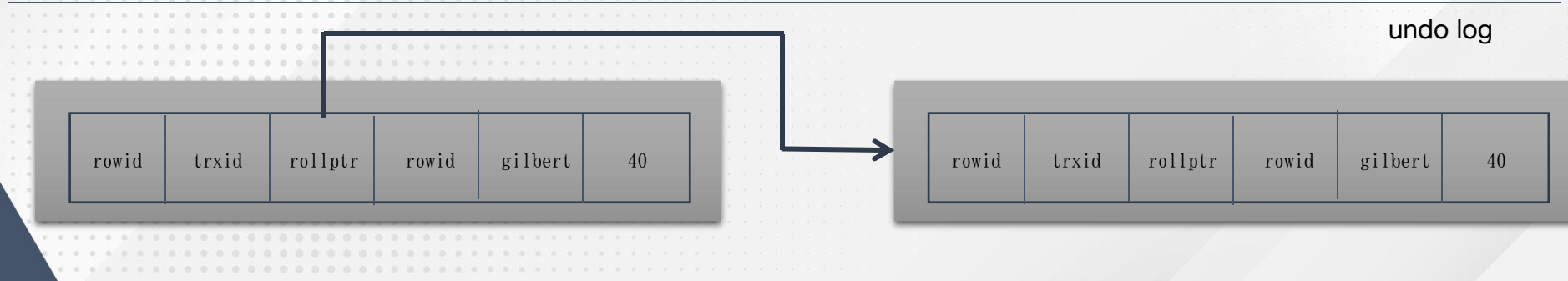
在事务没提交之前，MySQL会先记录更新前的数据到 undo log日志文件里面，当事务回滚时或者数据库崩溃时，可以利用 undo log来进行回退。

回滚

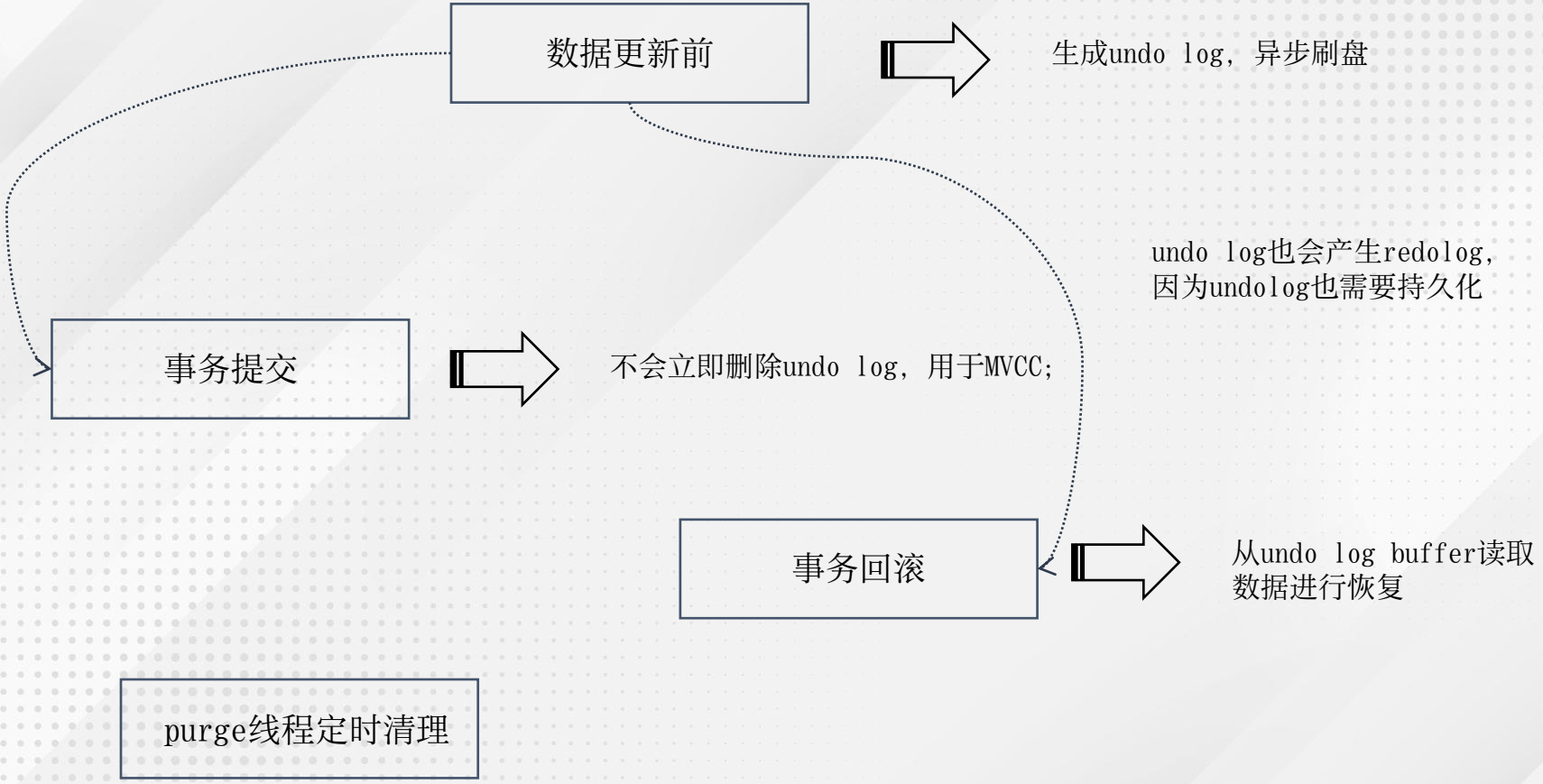
多版本控制

T0: insert into user(name , age) values （“gilbert” , 40);

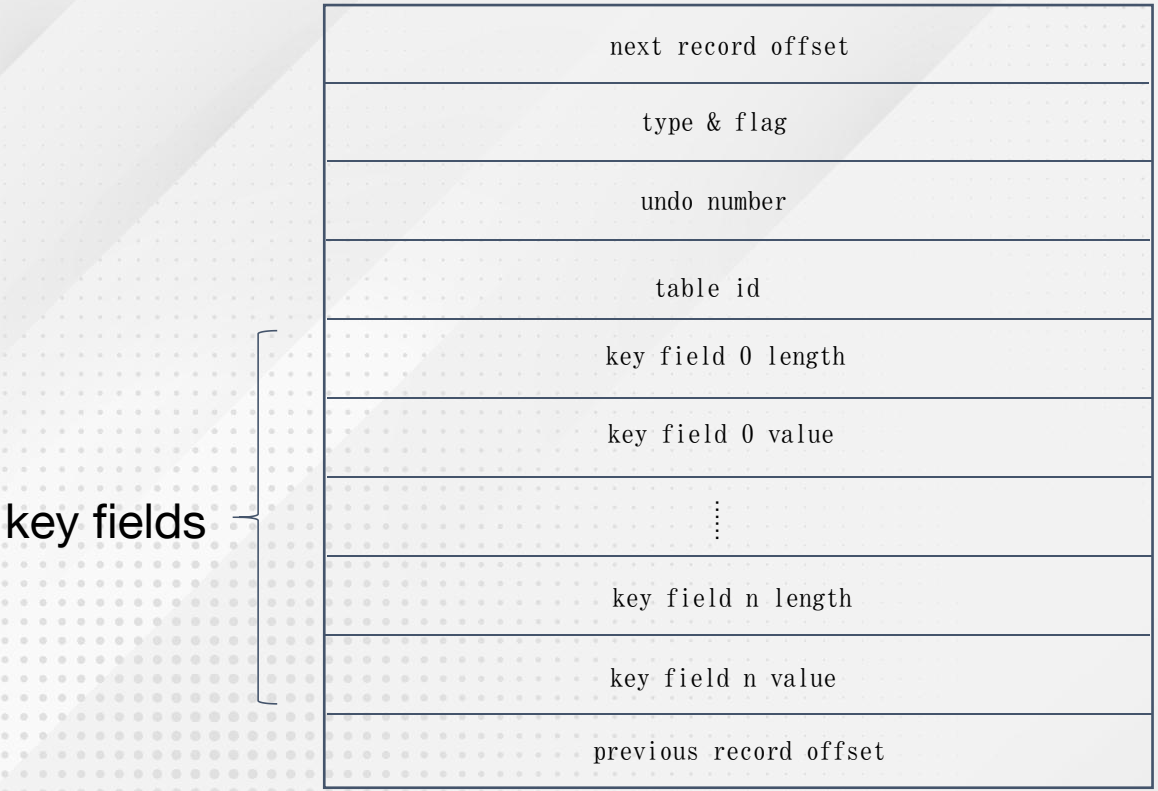
T1: update user set age=50 where name=“gilbert”;



Undo Log生成

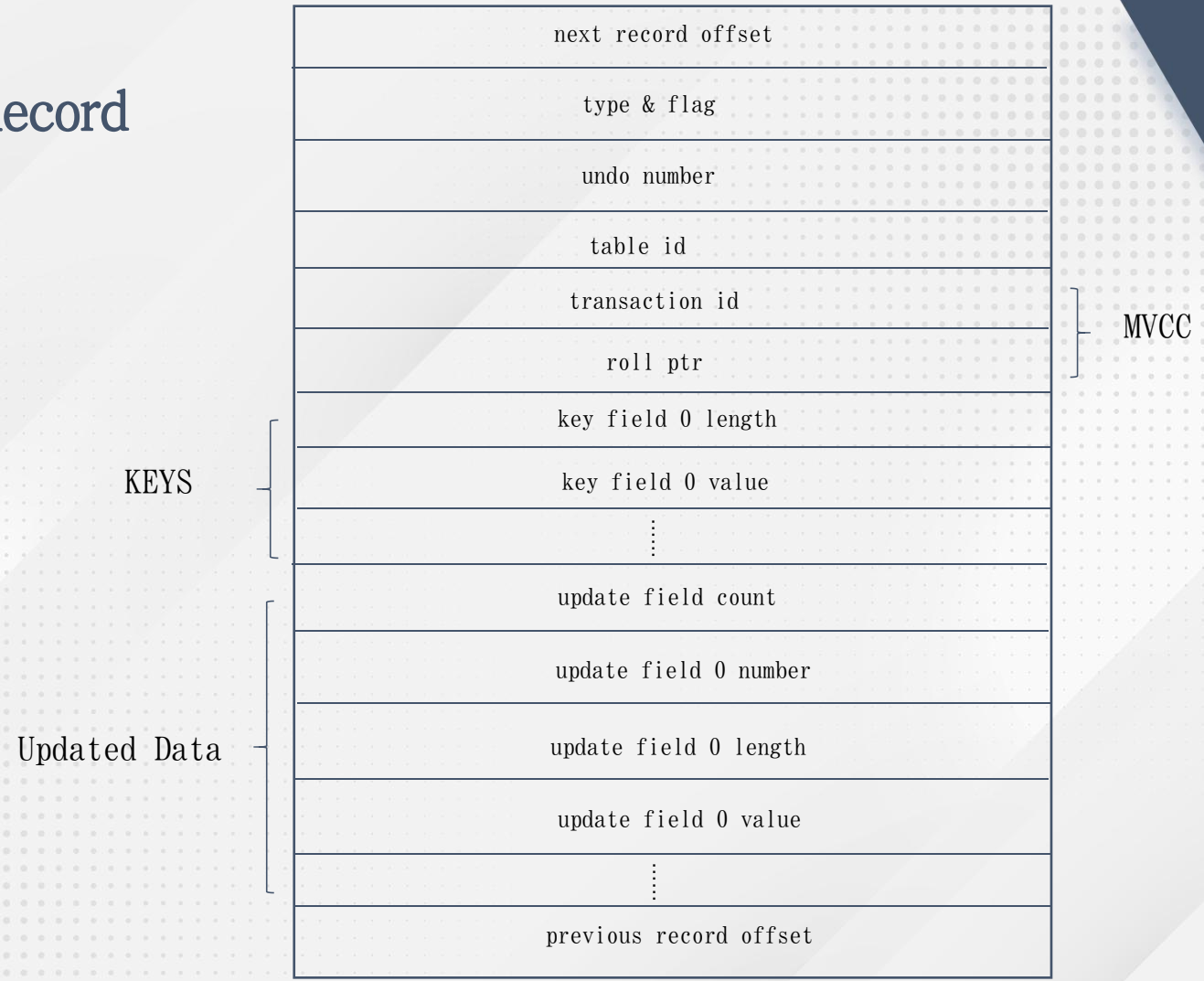

数据更新前
生成undo log，异步刷盘
undo log也会产生redolog，
因为undolog也需要持久化
事务提交
不会立即删除undo log，用于MVCC；
事务回滚
从undo log buffer读取
数据进行恢复
purge线程定时清理

Insert类型的Undo Record



在MVCC中不承担作用，只涉及到回滚，只需要记录KEY

Update类型的Undo Record



Undo Log逻辑组织方式

1:1

同一个事务

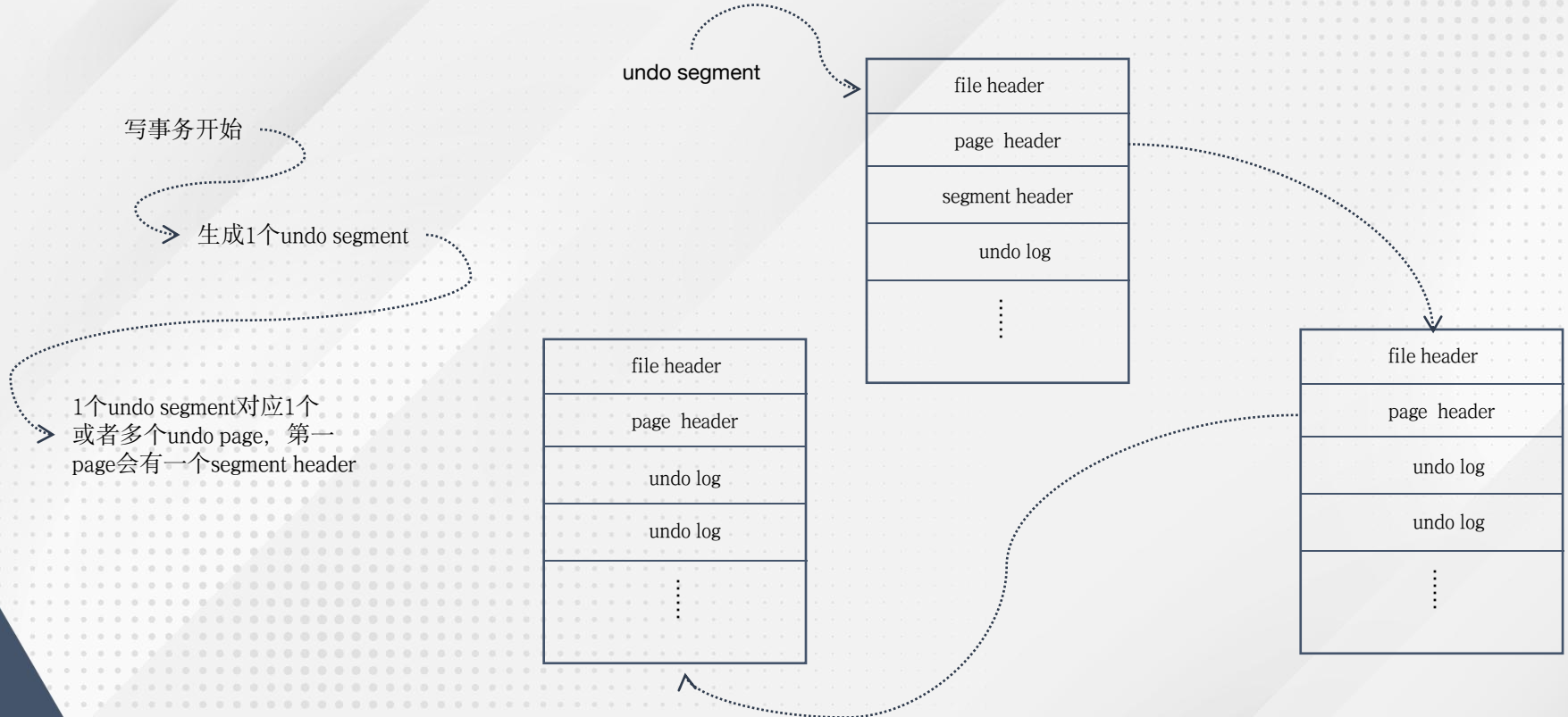
n:1

不同一个事务

(通过rollptr单向链表)

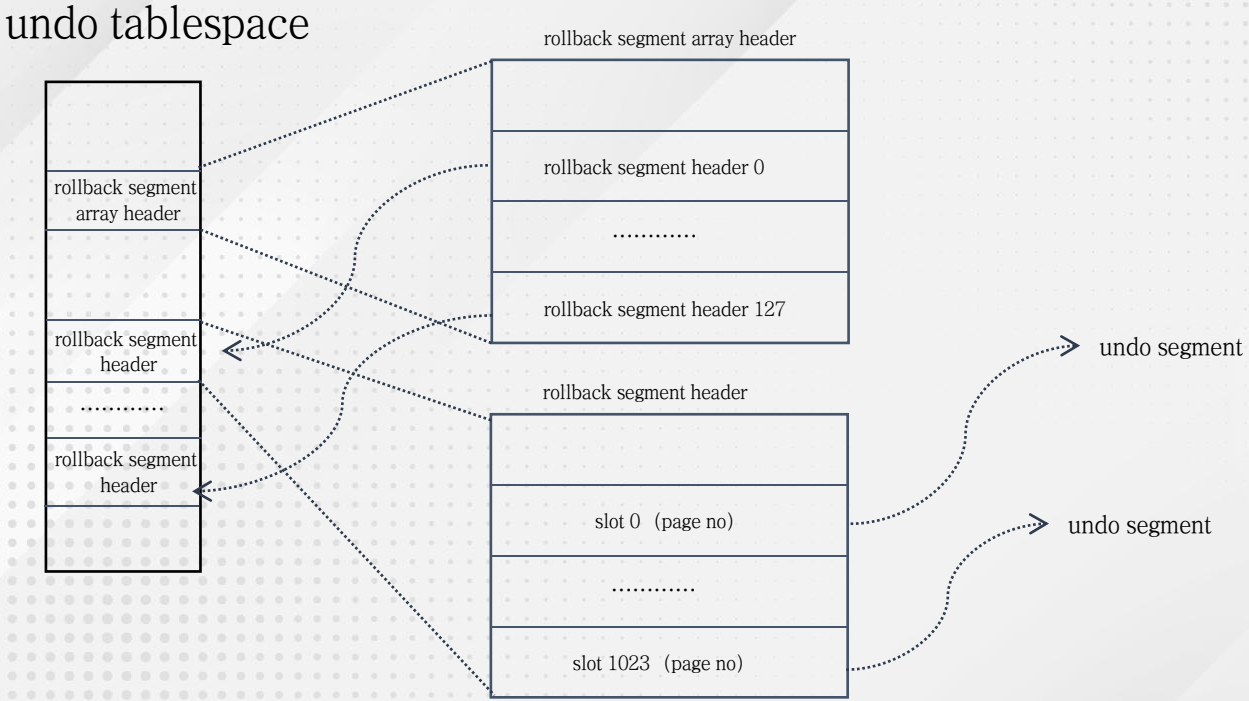
Undo Log物理组织方式

事务越大产生的Undo Log越大，但是一个存储却是按照固定的Page大小来存放数据的；通过将较小的undo log尽可能的放在一个Page中，提升存储的利用率。减小页内碎片；对于大事务，Undolog较大，可能会使用多个Page承载；

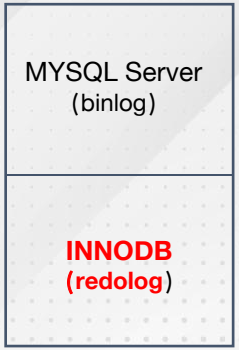


Undo Log物理组织方式

每一个写事务都会产生一个undo segment，当应用中有大量的并发写事务产生的时候，多个undo segment是按照slot的方式来组织，每组1024个segment；一个tablespace包含128个rollback segment；理论上一个tablespace可以支持同时创建128k个事务的undo segment；



RedoLog和Binlog的区别

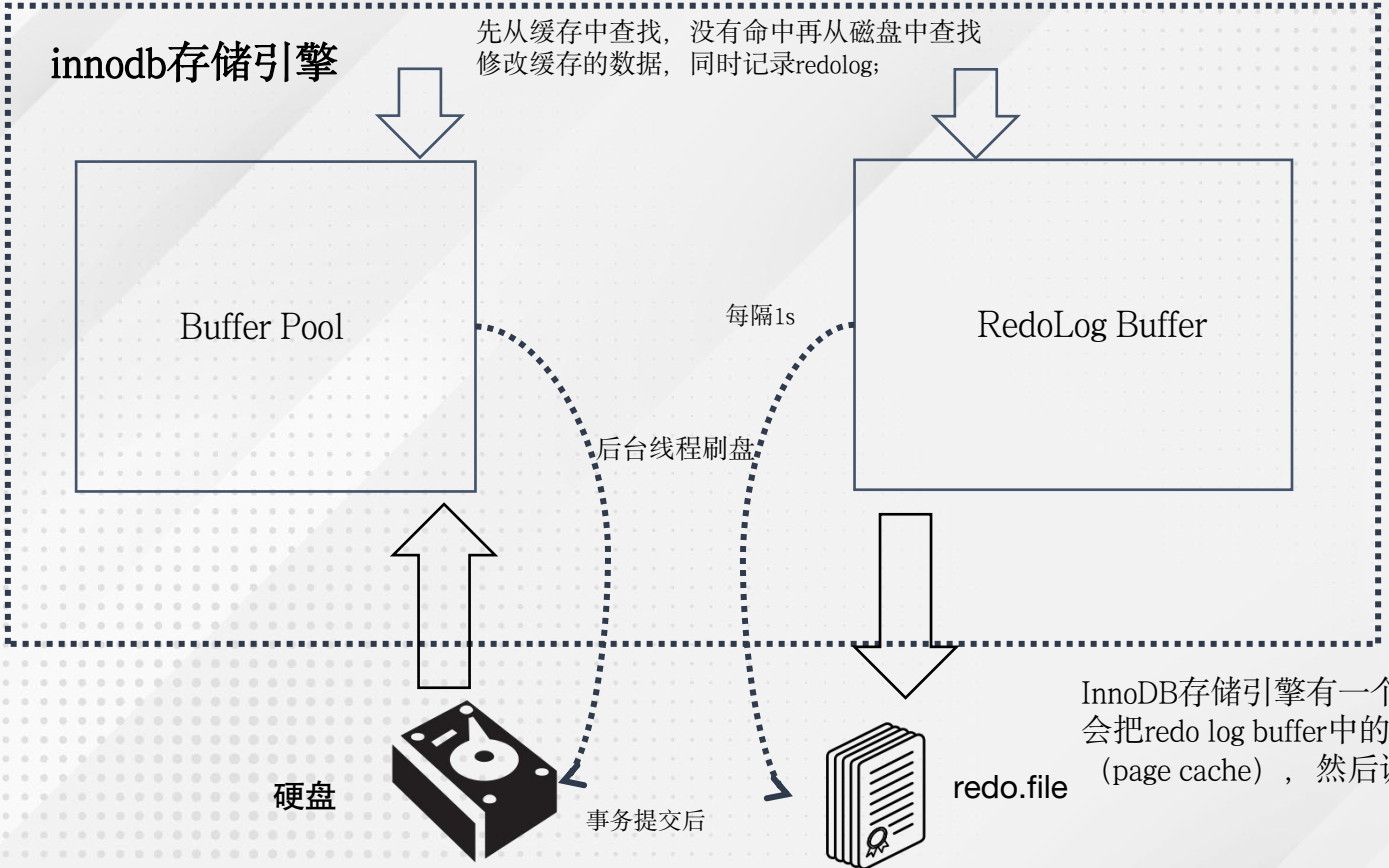


BINLOG	REDOLOG
MYSQL服务层概念，不止针对innodb一种存储引擎	INNODB存储层的概念
对数据的改动，表结构更改等	对数据的改动
事务提交后一次写入	事务中，不断写入
不是循环使用，在写满或者重启之后，生成新的binlog文件	文件固定大小，文件是循环写
人工恢复数据，主从复制	对用户不可见，异常宕机或者存储故障数据自动恢复

RedoLog工作方式

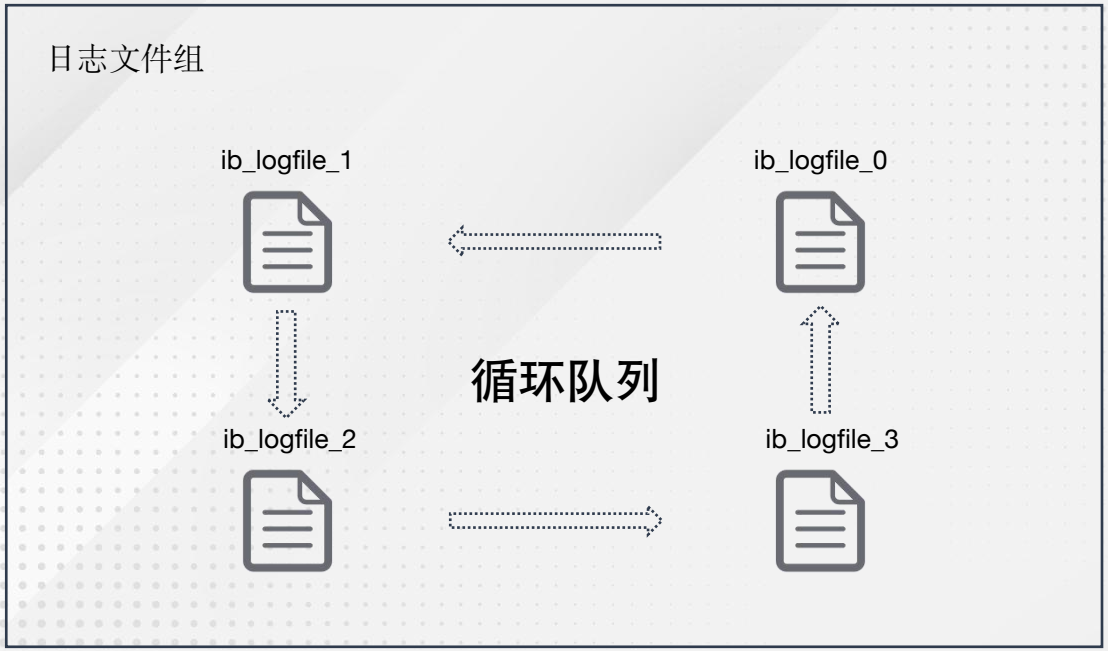
三种刷盘模式：

1. 表示每次事务提交时不进行刷盘操作
2. 表示每次事务提交时都将进行刷盘操作（默认值）
3. 表示每次事务提交时都只把redo log buffer内容写入page cache



InnoDB存储引擎有一个后台线程，每隔1秒，就会把redo log buffer中的内容写到文件系统缓存（page cache），然后调用fsync刷盘

RedoLog文件组



锁

锁类型	关注点
表锁	1. 性能差，锁全局； 2. 适合读多写少的场景；
行锁	1. 作用于索引之上的锁，锁粒度小，并发性较前者高； 2. 索引失效，那么退化成表锁；
间隙锁	1. 两条索引记录之间的开区间部分； 2. 阻止其他事务在间隙内插入数据，本事务则可以； 3. 事务隔离级别伟RC/RU有行锁没有间隙锁，RR和SR是有间隙锁的；
Next-Key Lock	1. Gap Lock+Row Lock，左开右闭区间； 2. 在默认事务隔离级别下，RR，使用排他读，是可以防止幻读的，但是使用快照读是不可以的；
共享锁/排他所	1. 行锁； 2. 共享锁，是并发读需要持有的锁，select ... in share mode,select ... for update； 3. 排他锁，是并发更新和删除需要持有的锁；
意向共享锁/意向排他锁	1. 表锁 2. 在获取共享锁和排他锁之前，需要先获取意向锁；
插入意向锁	
自增锁	1. 表锁 2. 自增id插入使用

MVCC

在读-写这类场景下，不加锁的情况，不发生冲突，提升并发性

读快照REDVIEW，某一时刻活跃事务id构成集成m_ids

m_ids集合的生成逻辑

1. 如果当前事务id在m_ids中命中，说明该事务还未提交，数据不可见，基于回滚指针回溯到上一个版本；
2. 如果当前事务id和生成ReadView的事务id(m_creator_trx_id)相等，说明他们在同一个事务中，数据是可见的；
3. 如果当前事务id小于min(m_ids)，说明该事物已经提交，数据可见；
4. 如果当前事务id大于max(m_ids)+1，说明在该事务启动后才启动，数据不可见，通过回滚指针回溯到上一个版本进行判断；

作用于事务隔离级别RC（每次select）和RR（第一次select）

隔离级别

隔离级别	描述
Read Uncommited	脏读
Read Committed	解决脏读，不可重复读
Repeatable Read	可重复读，幻读
Serial Read	串行读，性能最差

不可重复读和幻读的区别?

执行计划

如何利用

字段	描述
id	id越大，越先执行
	id相同，从上到下执行

字段	类型	描述
select_type	simple	不包括union查询和子查询，可以包括join查询
	primary	包含子查询，外查询标记为此类型
	subquery	在select/where后面子查询
	derived	在from后面的子查询
	union	在union后的查询标记为此类型
	union result	在union结果集中，select出来的查询

如何利用

字段	类型	描述
type	system	系统表，表中只包含一行记录
	const	通过索引直接命中单行
	eq_ref	扫描主键或者唯一索引，命中单行或者多行
	ref	扫描非唯一索引，命中但行或者多行
	range	扫描索引，范围扫描，between，>= and <= etc.
	index	扫描索引，返回所有行；比如分页优化；
	all	全表扫描

如何利用

字段	描述
possible_keys	可能使用到的索引；
key	实际使用到的索引；
key_len	索引长度，单位字节key的长度建议不要太长，太长的话，需要考虑用前缀索引优化，对于主键一定要求使用bigint，防止在插入新行的时候产生页分裂带来的性能问题；
ref	显示该表的索引字段关联了哪张表的哪个字段
rows	SQL执行所涉及到的行数；
filtered	返回结果的行数占读取行数的百分比，值越大越好；

如何利用

字段	类型	描述
extra	using filesort	使用了索引外排序，性能非常差，造成额外的I/O;常见在order by语句中；
	using temporary	使用了临时表，性能也是比较差的，常见group by的语句中；
	using index	走覆盖索引，性能不错，避免回表
	using where	用了where子句；
	using join buffer	使用了连接查询，并且使用了缓存；建议在join的字段上加上索引；
	distinct	行去重；

常用规范

DDL

编号	规则
1	DDL不允许回退
2	数据库命名规范，pc_XXXXX，不超过40个字符
3	使用innodb引擎，使用utf8mb4，默认排序utf8_general_ci
4	禁止使用存储过程，触发器和视图；
5	自增id作为primary key，bigint unsigned
6	表定义，字段禁止使用NULL，为其赋默认值；
7	字段类型必须符合实际业务的需求，不能滥用varchar
8	尽量不要使用TEXT/BLOB/CHAR，尽量使用varchar
9	每张表控制在3000w行以内
10	每张表必须提供的字段，id，create_time，update_time以及is_deleted
11	尽量选好时间点做DDL

SQL

编号	规则
1	对于2C端的应用控制关联查询在3张表以内
2	建议不要使用SQL拼接，尽量使用PreparedStatement，防止SQL注入
3	Select具体的字段，避免使用Select *
4	不要在索引字段使用集合函数，防止索引失效
5	在代码中的SQL不能包含任何DDL
6	能用batch insert尽量使用，提升性能
7	Innodb不要使用count()，通过redis等其他方案替代
8	走explain分析，提前预防

分库分表

编号	规则
1	考虑分库不分表，分库可以解决数据量和性能的问题，分表只能解决数据量的问题；
2	能用归档解决的，就不要分库分表
3	分库之后的分布式事务，建议采用最终一致性解决方案
4	能使用读写分离解决的尽量不分库不分表，一主多从，集联模式；
5	有先采用客户端分片，再次服务端；
6	单库的容量请按照3k TPS，1w QPS评估集群容量；做好预留；
7	请按照流量较大的业务接口维度进行分库分表；涉及到数据聚合实现，合理使用ES和Hadoop；

事务

编号	规则
1	一个事务，处理的行数不能超过1000 rows/s；
2	事务如果没有设置auto-commit，请及时commit释放锁
3	并发下避免死锁，如果存在并发对相同对数据做DML，请按顺序操作，比如使用kafka，尽量让同一个订单落在同一个partition；
4	在高并发下，尽量不要使用分布式事务，建议采用最终一致性方案确保；
5	事务传播级别，禁止使用REQUIRED_NEW
6	事务隔离级别建议使用RC

索引

编号	规则
1	自增id作为主键，防止插入数据时产生页分裂
2	将区分度较高的字段建索引；
3	order by和group by之后的字段使用索引
4	防止索引失效，or,in,like以及函数等；
5	尽量避免建立冗余索引，利用好联合索引，索引多了影响写入性能；
6	合理利用覆盖索引，避免回表；
7	避免深度分页，如果需要，可以走先扫描覆盖索引等方案；

约束

编号	规则
1	表结构设计满足三范式
2	主键的内容不能被修改
3	外键约束一般不在数据库上创建，只表达一个逻辑的概念，由程序控制

感谢大家