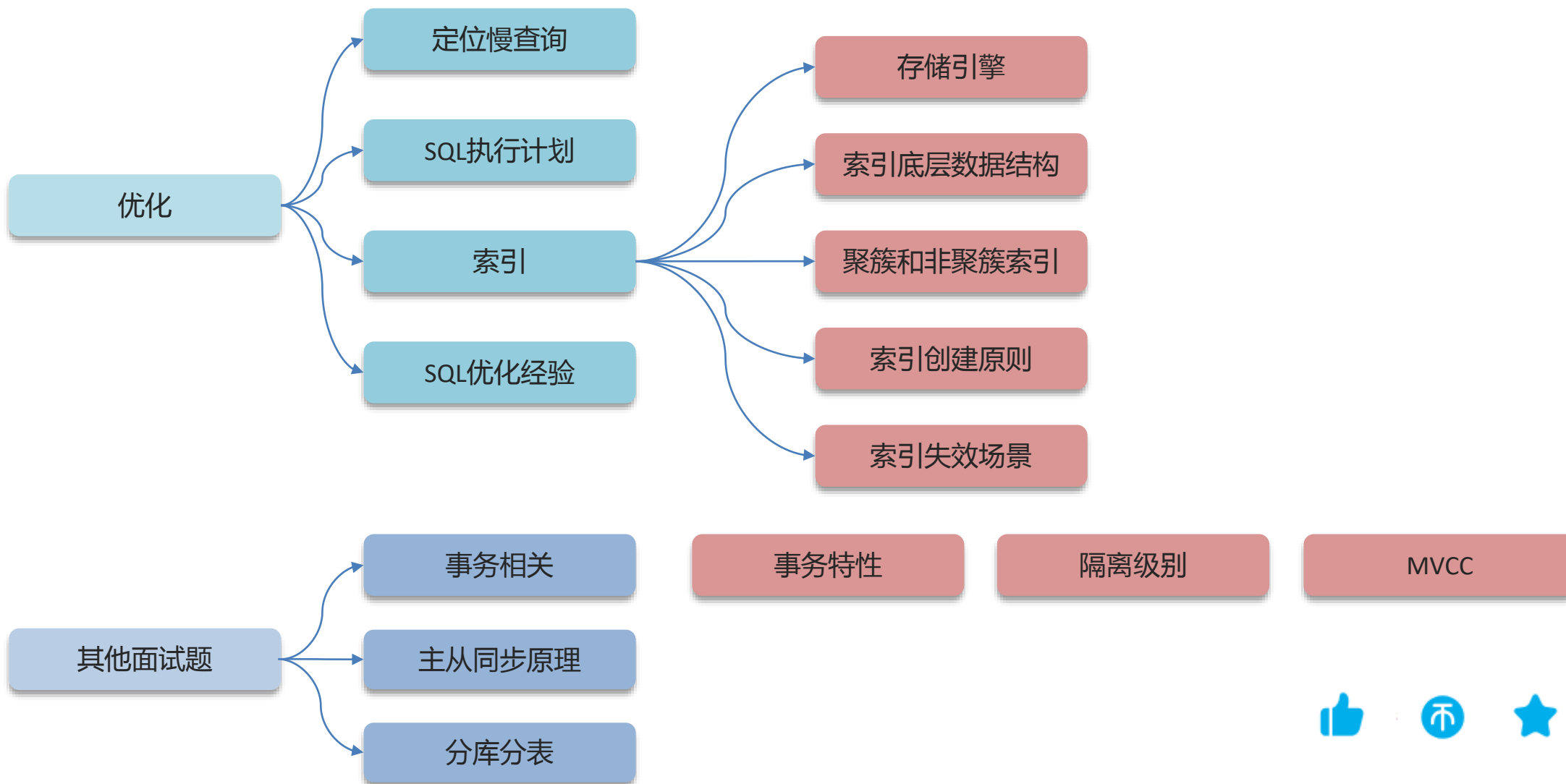


MySQL篇

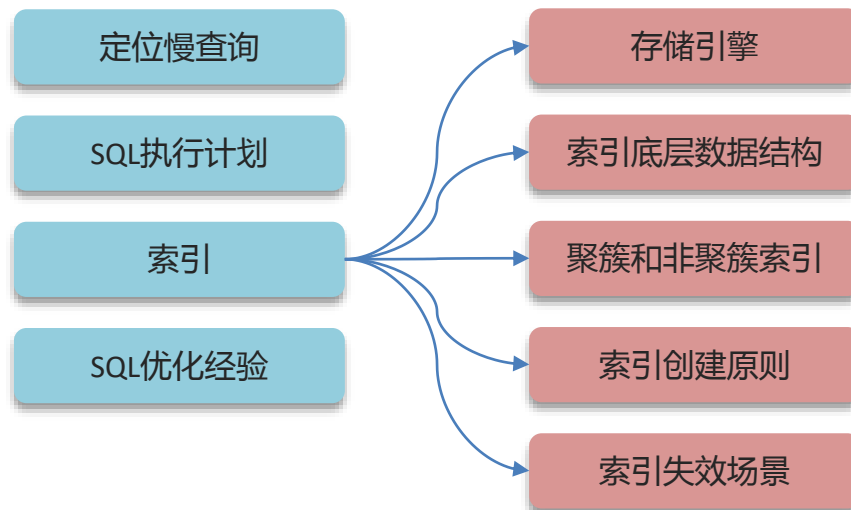


黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌



MySQL-优化





在MySQL中，如何定位慢查询?

- 聚合查询
- 多表查询
- 表数据量过大查询
- 深度分页查询

表象：页面加载过慢、接口压测响应时间过长（超过1s）

如何定位慢查询？

方案一：开源工具

- 调试工具：Arthas
- 运维工具：Prometheus、Skywalking



如何定位慢查询？

方案二：MySQL自带慢日志

慢查询日志记录了所有执行时间超过指定参数（long_query_time，单位：秒，默认10秒）的所有SQL语句的日志

如果要开启慢查询日志，需要在MySQL的配置文件（/etc/my.cnf）中配置如下信息：

```
# 开启MySQL慢日志查询开关
slow_query_log=1
# 设置慢日志的时间为2秒，SQL语句执行时间超过2秒，就会视为慢查询，记录慢查询日志
long_query_time=2
```

配置完毕之后，通过以下指令重新启动MySQL服务器进行测试，查看慢日志文件中记录的信息
/var/lib/mysql/localhost-slow.log。

```
# Time: 2023-03-15T15:21:55.178101Z
# User@Host: root[root] @ localhost [::1] Id: 3
# Query_time: 45.472697 Lock_time: 0.003903 Rows_sent: 10000000 Rows_examined: 10000000
use db01;
SET timestamp=1678893715;
select * from tb_sku;
```



如何定位慢查询?

1. 介绍一下当时产生问题的场景（我们当时的一个接口测试的时候非常的慢，压测的结果大概5秒钟）
2. 我们系统中当时采用了运维工具（Skywalking），可以监测出哪个接口，最终因为是sql的问题
3. 在mysql中开启了慢日志查询，我们设置的值就是2秒，一旦sql执行超过2秒就会记录到日志中（调试阶段）

面试官：MySQL中，如何定位慢查询？

候选人：

嗯~，我们当时做压测的时候有的接口非常的慢，接口的响应时间超过了2秒以上，因为我们当时的系统部署了运维的监控系统Skywalking，在展示的报表中可以看到是哪一个接口比较慢，并且可以分析这个接口哪部分比较慢，这里可以看到SQL的具体的执行时间，所以可以定位是哪个sql出了问题

如果，项目中没有这种运维的监控系统，其实在MySQL中也提供了慢日志查询的功能，可以在MySQL的系统配置文件中开启这个慢日志的功能，并且也可以设置SQL执行超过多少时间来记录到一个日志文件中，我记得上一个项目配置的是2秒，只要SQL执行的时间超过了2秒就会记录到日志文件中，我们就可以在日志文件找到执行比较慢的SQL了。



如何定位慢查询?

1. 介绍一下当时产生问题的场景（我们当时的一个接口测试的时候非常的慢，压测的结果大概5秒钟）
2. 我们系统中当时采用了运维工具（Skywalking），可以监测出哪个接口，最终因为是sql的问题
3. 在mysql中开启了慢日志查询，我们设置的值就是2秒，一旦sql执行超过2秒就会记录到日志中（调试阶段）



那这个SQL语句执行很慢, 如何分析呢?

- 聚合查询
- 多表查询
- 表数据量过大查询
- 深度分页查询



SQL执行计划（找到慢的原因）

一个SQL语句执行很慢, 如何分析

可以采用**EXPLAIN** 或者 **DESC**命令获取 MySQL 如何执行 SELECT 语句的信息

语法：

- 直接在select语句之前加上关键字 **explain / desc**
EXPLAIN SELECT 字段列表 **FROM** 表名 **WHERE** 条件；

```
mysql> explain select * from t_user where id = '1';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	t_user	NULL	const	PRIMARY	PRIMARY	98	const	1	100.00	NULL

1 row in set, 1 warning (0.00 sec)

一个SQL语句执行很慢, 如何分析

```
mysql> explain select * from t_user where id = '1';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	t_user	NULL	const	PRIMARY	PRIMARY	98	const	1	100.00	NULL

1 row in set, 1 warning (0.00 sec)

- possible_key 当前sql可能会使用到的索引
 - key 当前sql实际命中的索引
 - key_len 索引占用的大小
 - Extra 额外的优化建议
- 通过它们两个查看是否可能会命中索引

Extra	含义
Using where; Using Index	查找使用了索引，需要的数据都在索引列中能找到，不需要回表查询数据
Using index condition	查找使用了索引，但是需要回表查询数据

一个SQL语句执行很慢, 如何分析

```
mysql> explain select * from t_user where id = '1';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	t_user	NULL	const	PRIMARY	PRIMARY	98	const	1	100.00	NULL

1 row in set, 1 warning (0.00 sec)

● type 这条sql的连接的类型，性能由好到差为NULL、system、const、eq_ref、ref、range、index、all

➤ system：查询系统中的表

➤ const：根据主键查询

➤ eq_ref：主键索引查询或唯一索引查询

➤ ref：索引查询

➤ range：范围查询

➤ index：索引树扫描

➤ all：全盘扫描



那这个SQL语句执行很慢, 如何分析呢?

可以采用MySQL自带的分析工具 EXPLAIN

- 通过key和key_len检查是否命中了索引（索引本身存在是否有失效的情况）
- 通过type字段查看sql是否有进一步的优化空间，是否存在全索引扫描或全盘扫描
- 通过extra建议判断，是否出现了回表的情况，如果出现了，可以尝试添加索引或修改返回字段来修复

面试官：那这个SQL语句执行很慢, 如何分析呢?

候选人：如果一条sql执行很慢的话，我们通常会使用mysql自动的执行计划explain来去查看这条sql的执行情况，比如在这里面可以通过key和key_len检查是否命中了索引，如果本身已经添加了索引，也可以判断索引是否有失效的情况，第二个，可以通过type字段查看sql是否有进一步的优化空间，是否存在全索引扫描或全盘扫描，第三个可以通过extra建议来判断，是否出现了回表的情况，如果出现了，可以尝试添加索引或修改返回字段来修复



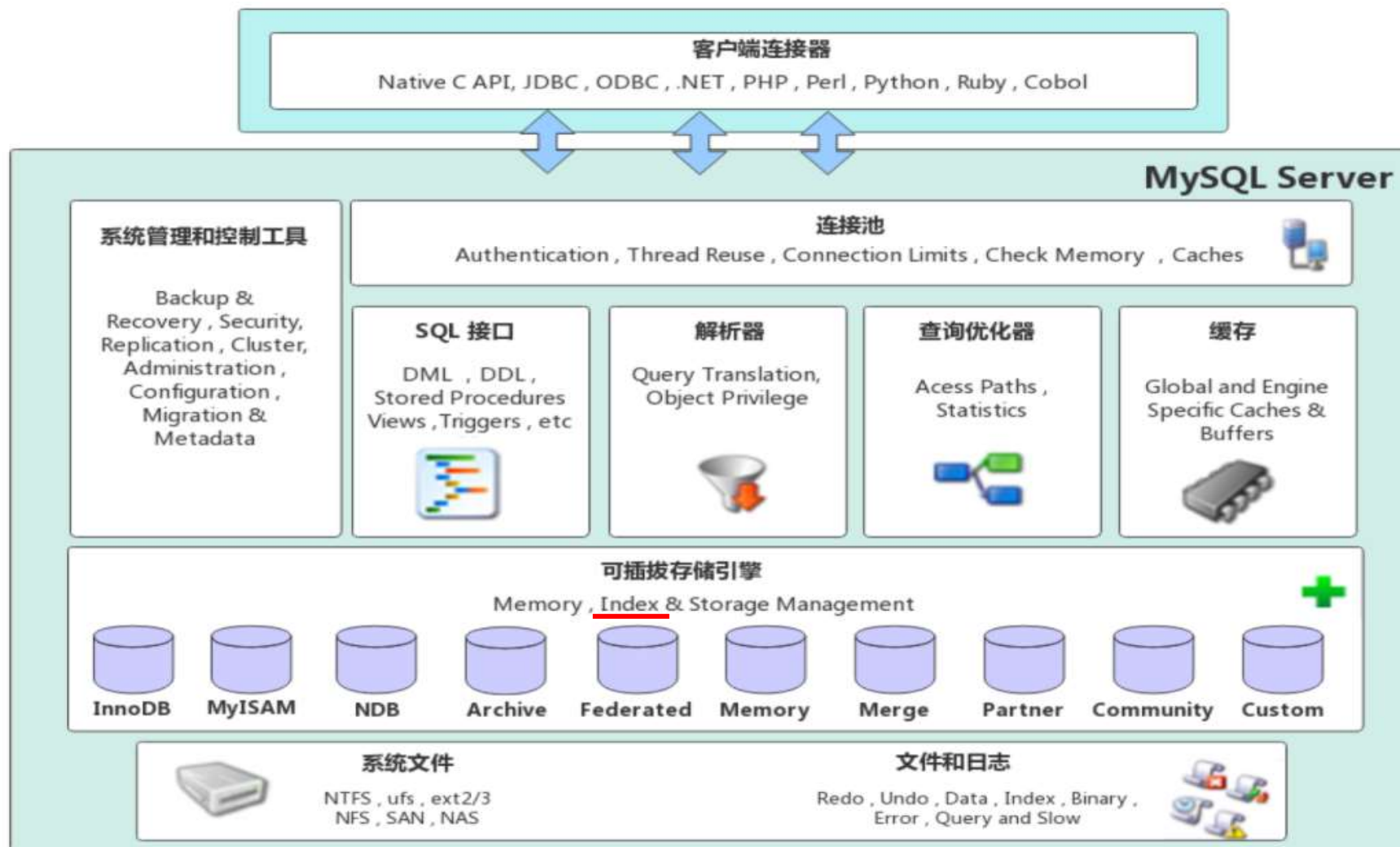
MYSQL支持的存储引擎有哪些, 有什么区别?

存储引擎就是存储数据、建立索引、更新/查询数据等技术的实现方式。存储引擎是基于表的，而不是基于库的，所以存储引擎也可被称为表类型。

特性	MyISAM	InnoDB	MEMORY
事务安全	不支持	支持	不支持
锁机制	表锁	表锁/行锁	表锁
外键	不支持	支持	不支持

- MySQL体系结构
- InnoDB存储的特点

MySQL体系结构



- 连接层
- 服务层
- 引擎层
- 存储层

存储引擎特点

- InnoDB

- 介绍

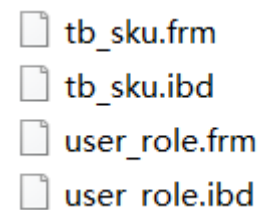
InnoDB是一种兼顾高可靠性和高性能的通用存储引擎，在 MySQL 5.5 之后，**InnoDB是默认的 MySQL 存储引擎**。

- 特点

- DML操作遵循ACID模型，支持事务
- 行级锁，提高并发访问性能
- 支持外键 FOREIGN KEY约束，保证数据的完整性和正确性

- 文件

- xxx.ibd：xxx代表的是表名，innoDB引擎的每张表都会对应这样一个表空间文件，存储该表的表结构（frm、sdi）、数据和索引。
- xxx.frm 存储表结构（MySQL8.0时，合并到表名.ibd中）



- tb_sku.frm
- tb_sku.ibd
- user_role.frm
- user_role.ibd



MYSQL支持的存储引擎有哪些, 有什么区别?

在mysql中提供了很多的存储引擎，比较常见有InnoDB、MyISAM、Memory

- InnoDB存储引擎是mysql5.5之后是默认的引擎，它支持事务、外键、表级锁和行级锁
- MyISAM是早期的引擎，它不支持事务、只有表级锁、也没有外键，用的不多
- Memory主要把数据存储在内存，支持表级锁，没有外键和事务，用的也不多



存储引擎在mysql的体系结构哪一层，主要特点是什么

- MySQL体系结构
- InnoDB存储的特点



索引在项目中的使用方式

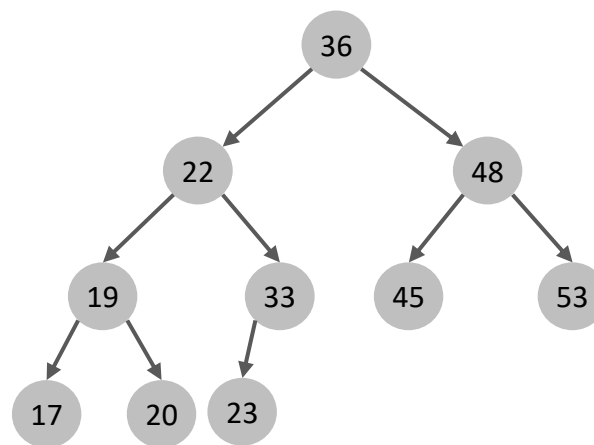
- 一是验证你的项目场景的真实性，二是为了作为深入发问的切入点
- 缓存
- 分布式锁
- 消息队列、延迟队列
-

了解过索引吗？（什么是索引）



索引 (index) 是帮助MySQL高效获取数据的数据结构 (有序)。在数据之外，数据库系统还维护着满足特定查找算法的数据结构 (B+树)，这些数据结构以某种方式引用 (指向) 数据，这样就可以在这些数据结构上实现高级查找算法，这种数据结构就是索引。

	id	name	age
0x07	1	金庸	36
0x56	2	张无忌	22
0x6A	3	杨逍	33
0xF3	4	韦一笑	48
0x90	5	常遇春	53
0x77	6	小昭	19
0xD1	7	灭绝	45
0x32	8	周芷若	17
0xE5	9	丁敏君	23
0xF2	10	赵敏	20



二叉树

红黑树

B树

B+树

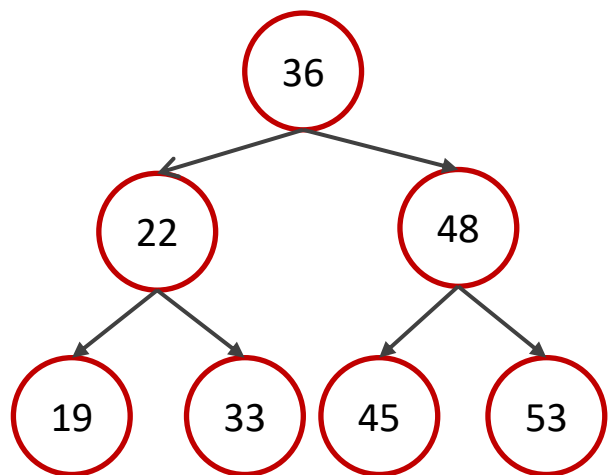
索引的底层数据结构了解过嘛？



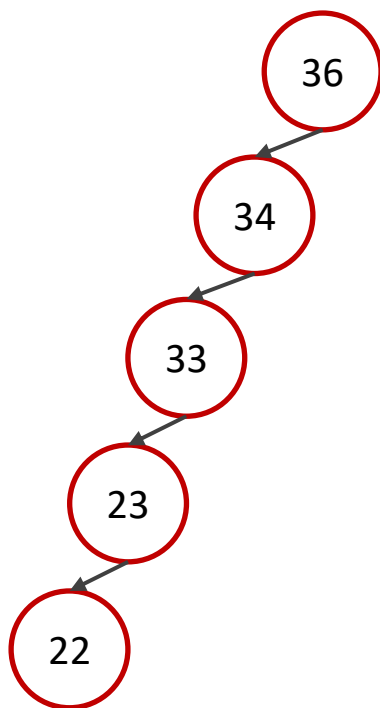
数据结构对比

MySQL默认使用的索引底层数据结构是B+树。再聊B+树之前，我们先聊聊二叉树和B树

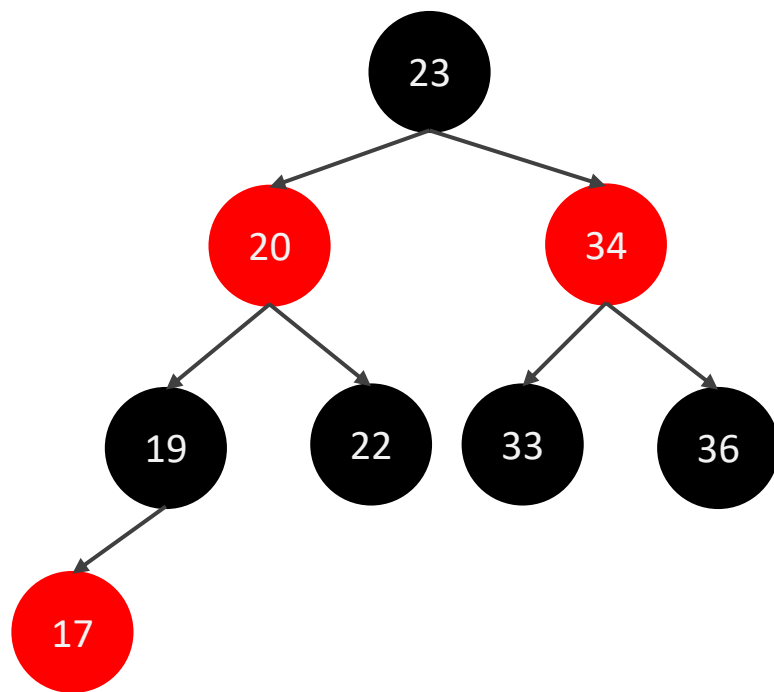
二叉搜索树



最坏的二叉树



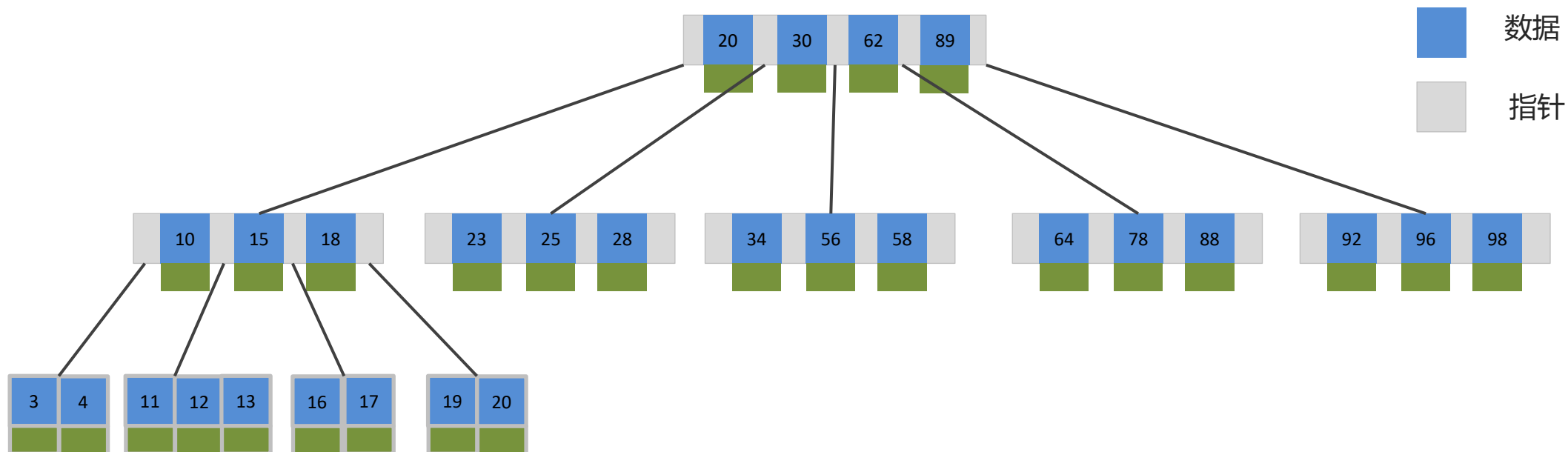
红黑树



数据结构对比

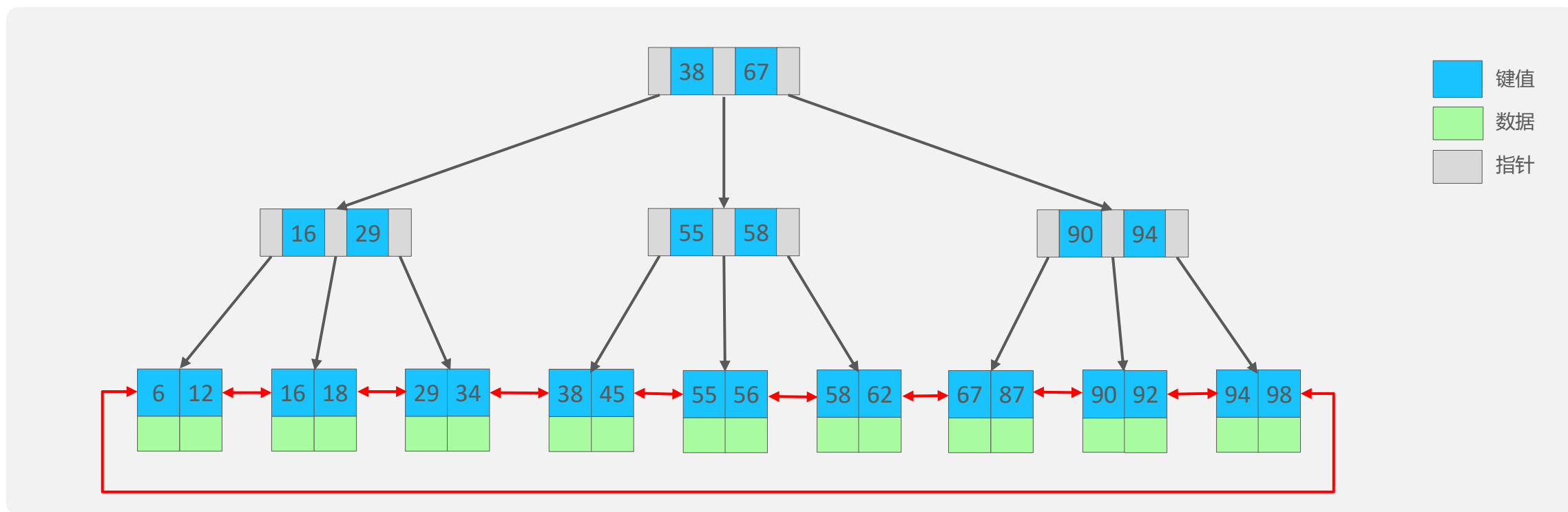
B-Tree，B树是一种多叉路平衡查找树，相对于二叉树，B树每个节点可以有多个分支，即多叉。

以一颗最大度数（max-degree）为5(5阶)的b-tree为例，那这个B树每个节点最多存储4个key



数据结构对比

B+Tree是在BTree基础上的一种优化，使其更适合实现外存储索引结构，InnoDB存储引擎就是用B+Tree实现其索引结构



B树与B+树对比:

①：磁盘读写代价B+树更低；②：查询效率B+树更加稳定；③：B+树便于扫库和区间查询



了解过索引吗？（什么是索引）

- 索引（index）是帮助MySQL
- 提高数据检索的效率，降低数
- 通过索引列对数据进行排序，

索引的底层数据结构了解过嘛？

MySQL的InnoDB引擎采用的B+

- 阶数更多，路径更短
- 磁盘读写代价B+树更低，非叶
- B+树便于扫库和区间查询，叶

面试官：了解过索引吗？（什么是索引）

候选人：嗯，索引在项目中还是比较常见的，它是帮助MySQL高效获取数据的数据结构，主要是用来提高数据检索的效率，降低数据库的IO成本，同时通过索引列对数据进行排序，降低数据排序的成本，也能降低了CPU的消耗

面试官：索引的底层数据结构了解过嘛？

候选人：MySQL的默认的存储引擎InnoDB采用的B+树的数据结构来存储索引，选择B+树的主要的原因是：第一阶数更多，路径更短，第二个磁盘读写代价B+树更低，非叶子节点只存储指针，叶子阶段存储数据，第三是B+树便于扫库和区间查询，叶子节点是一个双向链表

面试官：B树和B+树的区别是什么呢？

候选人：第一：在B树中，非叶子节点和叶子节点都会存放数据，而B+树的所有的数据都会出现在叶子节点，在查询的时候，B+树查找效率更加稳定

第二：在进行范围查询的时候，B+树效率更高，因为B+树都在叶子节点存储，并且叶子节点是一个双向链表



什么是聚簇索引什么是非聚簇索引？

什么是聚集索引，什么是二级索引（非聚集索引）

什么是回表？

id	name	gender
5	Kit	男
8	Ruby	男
10	Arm	女
11	Rose	女
12	Zoom	男
14	Tim	男
15	Dawn	男
21	Pine	男
25	Jack	男
26	Geek	女
27	Hero	男
28	Lem	男
30	Roxy	男
45	Oil	女
60	Xint	男
72	Mina	女
80	Lee	男
90	Lily	女

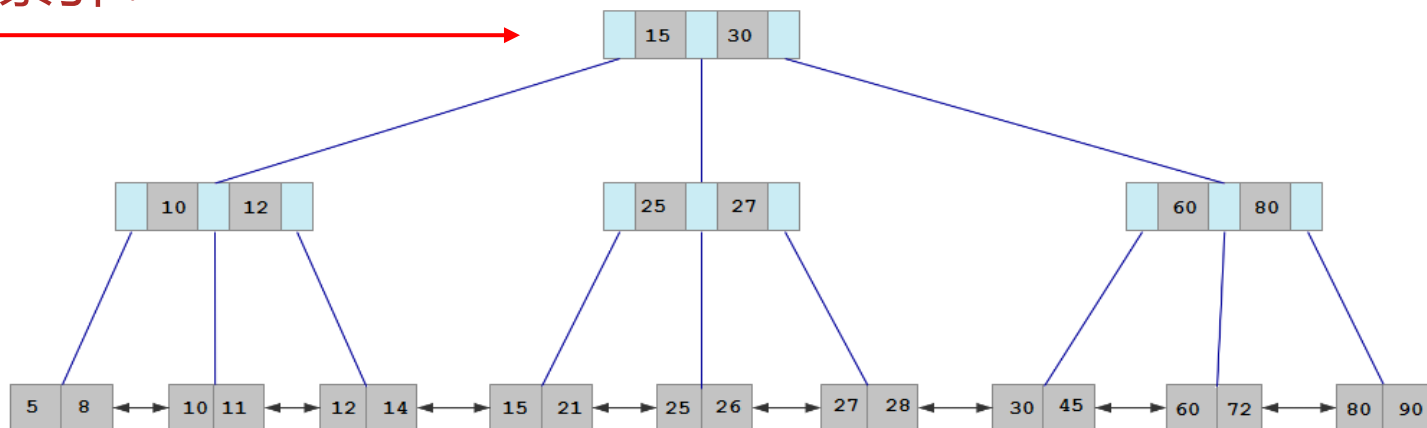
分类	含义	特点
聚集索引(Clustered Index)	将数据存储与索引放到了一块，索引结构的叶子节点保存了行数据	必须有,而且只有一个
二级索引(Secondary Index)	将数据与索引分开存储，索引结构的叶子节点关联的是对应的主键	可以存在多个

聚集索引选取规则:

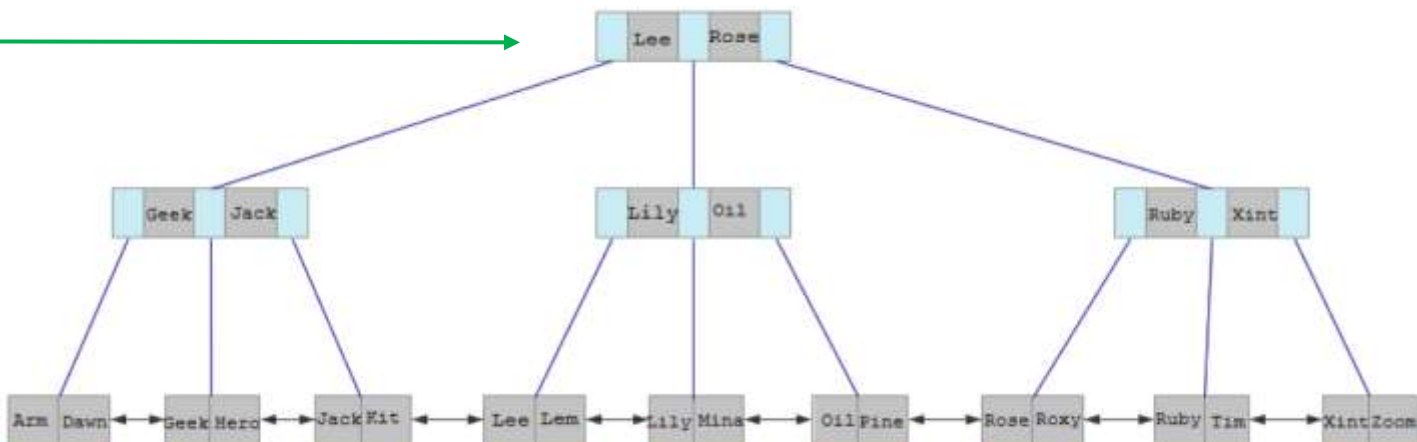
- 如果存在主键，主键索引就是聚集索引。
- 如果不存在主键，将使用第一个唯一（UNIQUE）索引作为聚集索引。
- 如果表没有主键，或没有合适的唯一索引，则InnoDB会自动生成一个rowid作为隐藏的聚集索引。

什么是聚簇索引和非聚簇索引？

id	name	gender
5	Kit	男
8	Ruby	男
10	Arm	女
11	Rose	女
12	Zoom	男
14	Tim	男
15	Dawn	男
21	Pine	男
25	Jack	男
26	Geek	女
27	Hero	男
28	Lem	男
30	Roxy	男
45	Oil	女
60	Xint	男
72	Mina	女
80	Lee	男
90	Lily	女



聚集索引



二级索引

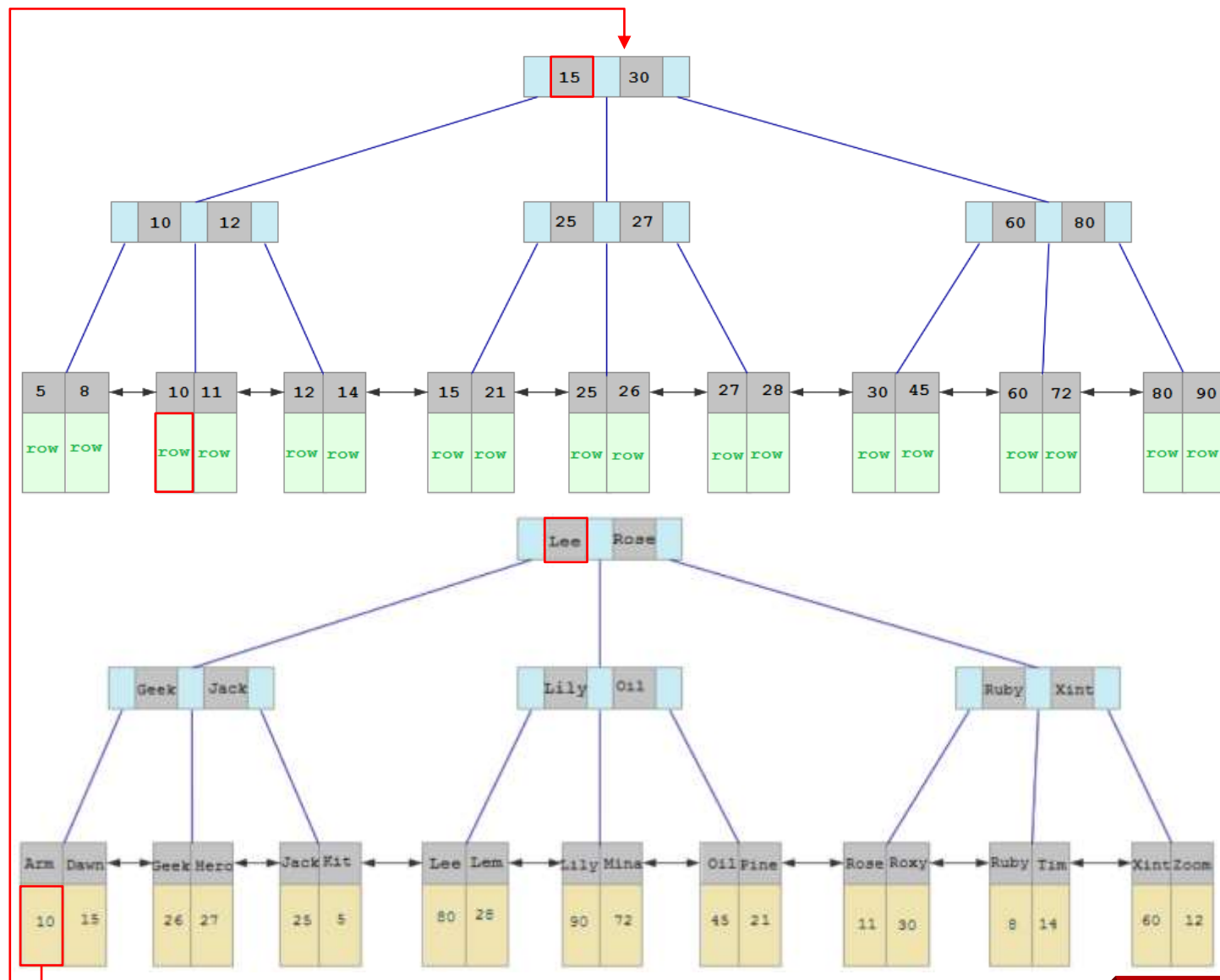
回表查询

```
select * from user where name = 'Arm';
```

回表查询

聚集索引

二级索引





什么是聚簇索引什么是非聚簇索引？

- 聚簇索引（聚集索引）
- 非聚簇索引（二级索引）

知道什么是回表查询嘛？

通过二级索引找到对应的三

面试官：什么是聚簇索引什么是非聚簇索引？

候选人：

好的~，聚簇索引主要是指数据与索引放到一块，B+树的叶子节点保存了整行数据，有且只有一个，一般情况下主键在作为聚簇索引的

非聚簇索引值的是数据与索引分开存储，B+树的叶子节点保存对应的主键，可以有多个，一般我们自己定义的索引都是非聚簇索引

面试官：知道什么是回表查询嘛？

候选人：嗯，其实跟刚才介绍的聚簇索引和非聚簇索引是有关系的，回表的意思就是通过二级索引找到对应的主键值，然后再通过主键值找到聚集索引中所对应的整行数据，这个过程就是回表

【备注：如果面试官直接问回表，则需要先介绍聚簇索引和非聚簇索引】



知道什么叫覆盖索引嘛？

覆盖索引是指查询使用了索引，并且需要返回的列，在该索引中已经全部能够找到。

id	name	gender	createdate
2	Arm	1	2021-01-01
3	Lily	0	2021-05-01
5	Rose	0	2021-02-14
6	Zoo	1	2021-06-01
8	Doc	1	2021-03-08
11	Lee	1	2020-12-03

- id为主键，默认是主键索引
- name字段为普通索引

```
select * from tb_user where id = 1
```

覆盖索引

```
select id , name from tb_user where name = 'Arm'
```

覆盖索引

```
select id , name , gender from tb_user where name = 'Arm'
```

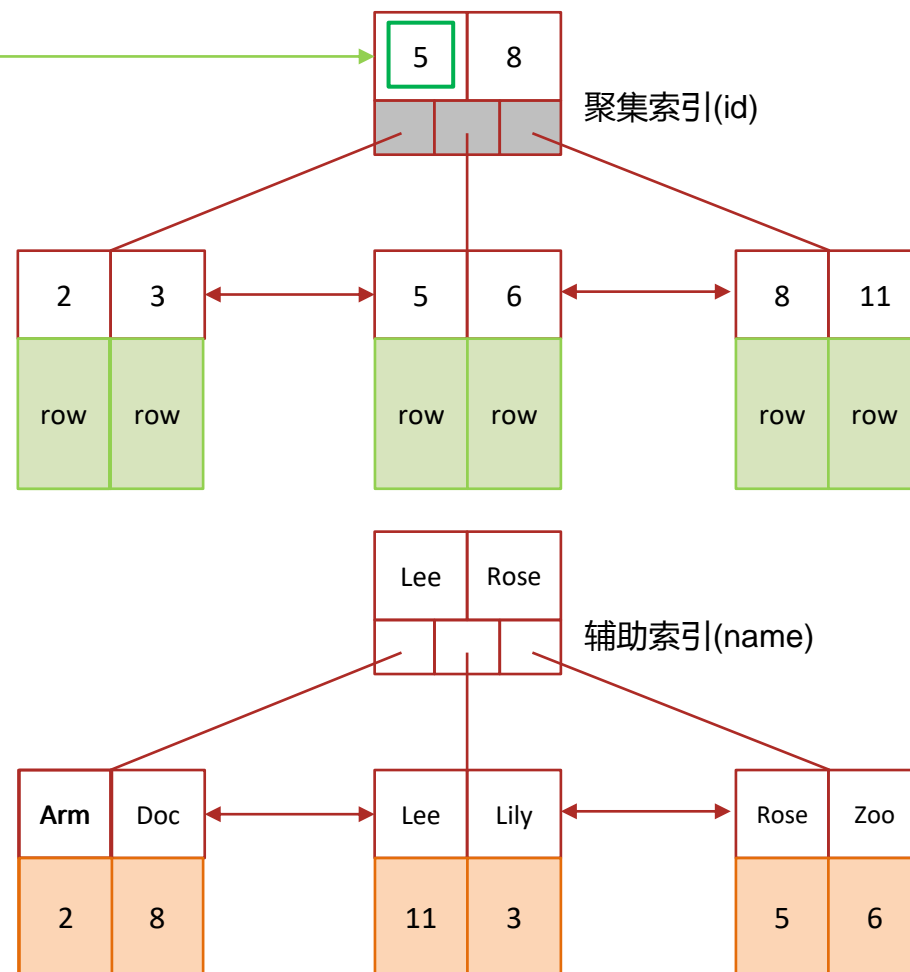
非覆盖索引(需要回表查询)

覆盖索引

覆盖索引是指 查询使用了索引，并且需要返回的列，在该索引中已经全部能够找到。

id	name	gender	createdate
2	Arm	1	2021-01-01
3	Lily	0	2021-05-01
5	Rose	0	2021-02-14
6	Zoo	1	2021-06-01
8	Doc	1	2021-03-08
11	Lee	1	2020-12-03

```
select * from tb_user where id = 2;
```



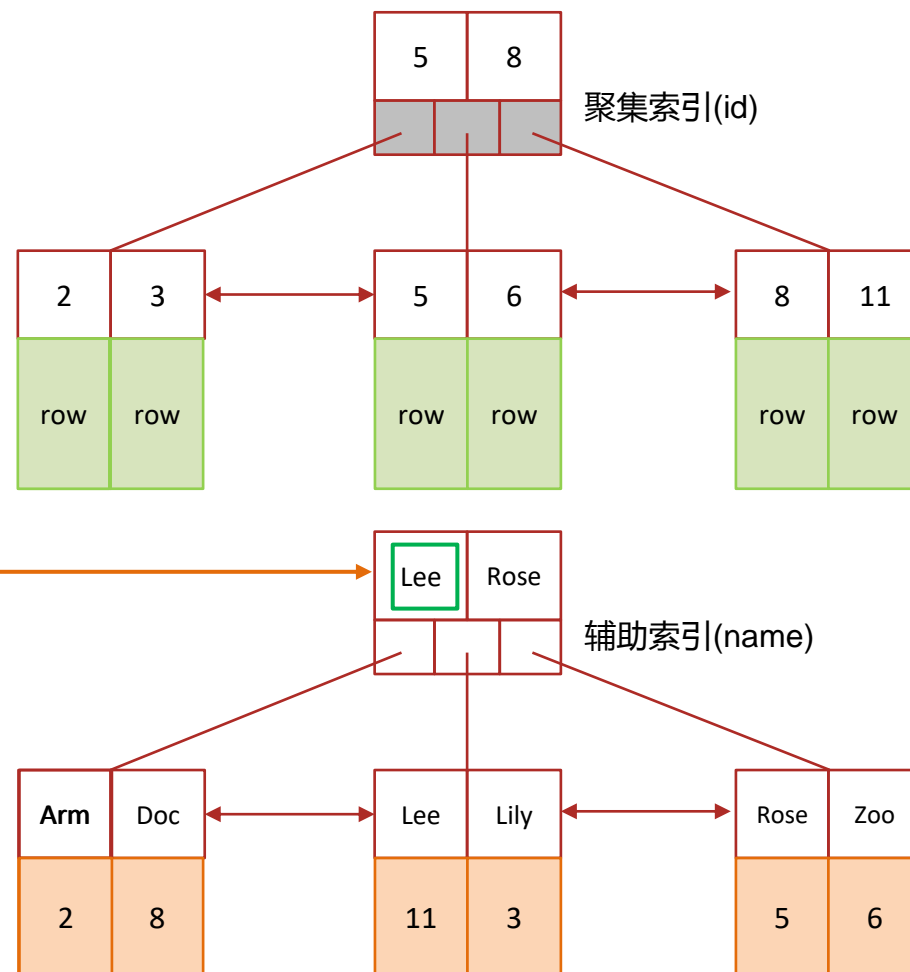
覆盖索引

覆盖索引是指 查询使用了索引，并且需要返回的列，在该索引中已经全部能够找到。

id	name	gender	createdate
2	Arm	1	2021-01-01
3	Lily	0	2021-05-01
5	Rose	0	2021-02-14
6	Zoo	1	2021-06-01
8	Doc	1	2021-03-08
11	Lee	1	2020-12-03

```
select * from tb_user where id = 2;
```

```
select id,name from tb_user where name = 'Arm';
```



覆盖索引

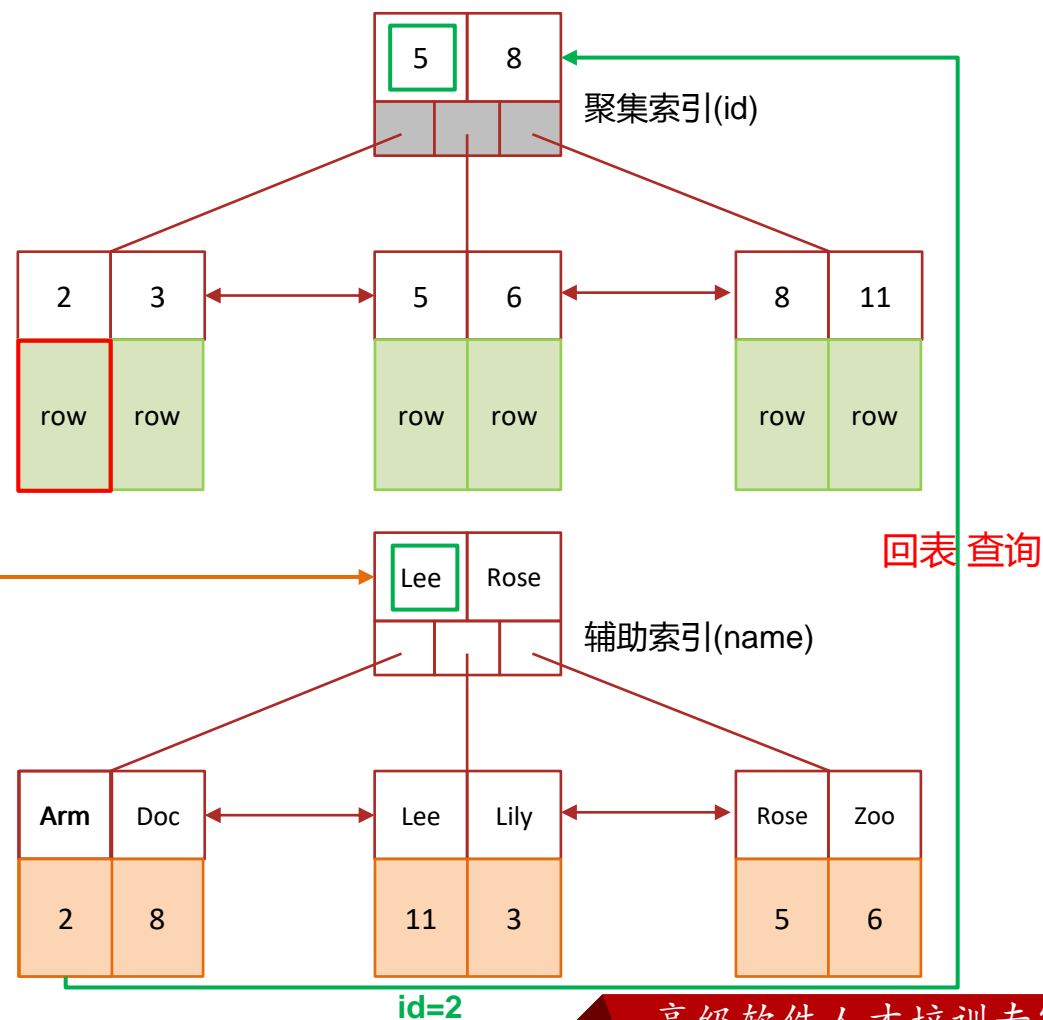
覆盖索引是指 查询使用了索引，并且需要返回的列，在该索引中已经全部能够找到。

id	name	gender	createdate
2	Arm	1	2021-01-01
3	Lily	0	2021-05-01
5	Rose	0	2021-02-14
6	Zoo	1	2021-06-01
8	Doc	1	2021-03-08
11	Lee	1	2020-12-03

```
select * from tb_user where id = 2;
```

```
select id,name from tb_user where name = 'Arm';
```

```
select id,name,gender from tb_user where name = 'Arm';
```





知道什么叫覆盖索引嘛？

覆盖索引是指查询使用了索引，返回的列，必须在索引中全部能够找到

- 使用id查询，直接走聚集索引查询，一次索引扫描，直接返回数据，性能高。
- 如果返回的列中没有创建索引，有可能会触发回表查询，尽量避免使用select *



MYSQL超大分页怎么处理？

可以使用覆盖索引解决

MYSQL超大分页处理

在数据量比较大时，如果进行limit分页查询，在查询时，越往后，分页查询效率越低。

我们一起来看看执行limit分页查询耗时对比：

```
mysql> select * from tb_sku limit 0,10;  
10 rows in set (0.00 sec)  
  
mysql> select * from tb_sku limit 9000000,10;  
10 rows in set (11.05 sec)
```

因为，当在进行分页查询时，如果执行 limit 9000000,10，此时需要MySQL排序前9000010 记录，仅仅返回 9000000 - 9000010 的记录，其他记录丢弃，查询排序的代价非常大。

MYSQL超大分页处理

优化思路: 一般分页查询时，通过创建 **覆盖索引** 能够比较好地提高性能，可以通过**覆盖索引**加**子查询**形式进行优化

```
select *  
from tb_sku t,  
    (select id from tb_sku order by id limit 9000000,10) a  
where t.id = a.id;
```

10 rows in set (7.13 sec)

10 rows in set (7.37 sec)

10 rows in set (7.19 sec)



知道什么叫覆盖索引嘛？

覆盖索引是指查询使用了索引，返回

- 使用id查询，直接走聚集索引查
- 如果返回的列中没有创建索引，

面试官：知道什么叫覆盖索引嘛？

候选人：嗯~，清楚的

覆盖索引是指select查询语句使用了索引，在返回的列，必须在索引中全部能够找到，如果我们使用id查询，它会直接走聚集索引查询，一次索引扫描，直接返回数据，性能高。

如果按照二级索引查询数据的时候，返回的列中没有创建索引，有可能会触发回表查询，尽量避免使用select *，尽量在返回的列中都包含添加索引的字段

面试官：MYSQL超大分页怎么处理？

候选人：嗯，超大分页一般都是在数据量比较大时，我们使用了limit分页查询，并且需要对数据进行排序，这个时候效率就很低，我们可以采用覆盖索引和子查询来解决

先分页查询数据的id字段，确定了id之后，再用子查询来过滤，只查询这个id列表中的数据就可以了

因为查询id的时候，走的覆盖索引，所以效率可以提升很多

MYSQL超大分页怎么处理？

问题：在数据量比较大时，limit分页

解决方案：覆盖索引+子查询





索引创建原则有哪些？

- 先陈述自己在实际的工作中是怎么用的
- 主键索引
- 唯一索引
- 根据业务创建的索引(复合索引)

索引创建原则有哪些？

- 1). 针对于数据量较大，且查询比较频繁的表建立索引。 **单表超过10万数据（增加用户体验）**
- 2). 针对于常作为查询条件（where）、排序（order by）、分组（group by）操作的字段建立索引。
- 3). 尽量选择区分度高的列作为索引，尽量建立唯一索引，区分度越高，使用索引的效率越高。
- 4). 如果是字符串类型的字段，字段的长度较长，可以针对于字段的特点，建立前缀索引。

5). 尽量使用联合索引

6). 要控制索引的数量

7). 如果索引列不能

个索引最有效地用

id:		title:		sell point:		
sellerid:	name:	nickname:	password:	status:	address:	
1 alibaba	阿里巴巴	阿里小店	e10adc3949ba59abbe...	1	北京市	
2 baidu	百度科技有限公司	百度小店	e10adc3949ba59abbe...	1	北京市	
3 huawei	华为科技有限公司	华为小店	e10adc3949ba59abbe...	0	北京市	
4 itcast	传智播客教育科技有...	传智播客	e10adc3949ba59abbe...	1	北京市	
5 itheima	黑马程序员	黑马程序员	e10adc3949ba59abbe...	0	北京市	


```
mysql> show index from tb_seller;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name
tb_seller	1	tb_seller_index	1	name
tb_seller	1	tb_seller_index	2	status
tb_seller	1	tb_seller_index	3	address

847278	飞利浦 老...	超长待机，大爱无限，更好用！飞利浦简单健康老人手机！外观圆滑，手感极佳！
855739	三星 Galax...	三星经典旗舰机！5英寸1080P高清屏+1300万像素主摄像头！
856645	三星 Galax...	年货特价来袭！三星经典旗舰机！

查询效率。

以更好地确定哪



索引创建原则有哪些？

- 1). 数据量较大，且查询比较频繁的主
- 2). 常作为查询条件、
- 3). 字段内容区分度高
- 4). 内容较长，使用前缀索引
- 5). 尽量联合索引
- 6). 要控制索引的数量
- 7). 如果索引列不能存

面试官：索引创建原则有哪些？

候选人：嗯，这个情况有很多，不过都有一个大前提，就是表中的数据要超过10万以上，我们才会创建索引，并且添加索引的字段是查询比较频繁的字段，一般也是像作为查询条件，排序字段或分组的字段这些。

还有就是，我们通常创建索引的时候都是使用复合索引来创建，一条sql的返回值，尽量使用覆盖索引，如果字段的区分度不高的话，我们也会把它放在组合索引后面的字段。

如果某一个字段的内容较长，我们会考虑使用前缀索引来使用，当然并不是所有的字段都要添加索引，这个索引的数量也要控制，因为添加索引也会导致新增改的速度变慢。



什么情况下索引会失效？

索引失效的情况有很多，可以说一些自己遇到过的，不要张口就得得说一堆背诵好的面试题
(适当的思考一下，回想一下，更真实)

给tb_seller创建联合索引，字段顺序：name，status，address

```
mysql> show index from tb_seller;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name
tb_seller	1	tb_seller_index	1	name
tb_seller	1	tb_seller_index	2	status
tb_seller	1	tb_seller_index	3	address

那快读判断索引是否失效了呢？ 执行计划explain

什么情况下索引会失效？

1). 违反最左前缀法则

如果索引了多列，要遵守最左前缀法则。指的是查询从索引的最左前列开始，并且不跳过索引中的列。匹配最左前缀法则，走索引：

```
mysql> show index from tb_seller;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name
tb_seller	1	tb_seller_index	1	name
tb_seller	1	tb_seller_index	2	status
tb_seller	1	tb_seller_index	3	address

```
mysql> explain select * from tb_seller where name = '小米科技';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	tb_seller	NULL	ref	tb_seller_index	tb_seller_index	303	const	1	100.00	NULL

1 row in set, 1 warning (0.00 sec)

```
mysql> explain select * from tb_seller where name = '小米科技' and status = '1';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	tb_seller	NULL	ref	tb_seller_index	tb_seller_index	309	const,const	1	100.00	NULL

1 row in set, 1 warning (0.00 sec)

```
mysql> explain select * from tb_seller where name = '小米科技' and status = '1' and address = '北京市';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	tb_seller	NULL	ref	tb_seller_index	tb_seller_index	612	const,const,const	1	100.00	NULL

1 row in set, 1 warning (0.00 sec)

什么情况下索引会失效？

违法最左前缀法则，索引失效：

```
mysql> show index from tb_seller;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name
tb_seller	1	tb_seller_index	1	name
tb_seller	1	tb_seller_index	2	status
tb_seller	1	tb_seller_index	3	address

```
mysql> explain select * from tb_seller where status = '1' and address = '北京市';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	tb_seller	NULL	ALL	NULL	NULL	NULL	NULL	12	8.33	Using where

1 row in set, 1 warning (0.00 sec)

```
mysql> explain select * from tb_seller where status = '1';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	tb_seller	NULL	ALL	NULL	NULL	NULL	NULL	12	10.00	Using where

1 row in set, 1 warning (0.00 sec)

如果符合最左法则，但是出现跳跃某一行，只有最左列索引生效：

```
mysql> explain select * from tb_seller where name = '小米科技' and address = '北京市';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	tb_seller	NULL	ref	tb_seller_index	tb_seller_index	303	const	1	10.00	Using index condition

什么情况下索引会失效？

2). 范围查询右边的列，不能使用索引。

```
mysql> explain select * from tb_seller where name = '小米科技' and status = '1' and address = '北京市';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	tb_seller	NULL	ref	tb_seller_index	tb_seller_index	612	const,const,const	1	100.00	NULL

```
1 row in set, 1 warning (0.00 sec)
```

```
mysql> explain select * from tb_seller where name = '小米科技' and status > '1' and address = '北京市';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	tb_seller	NULL	range	tb_seller_index	tb_seller_index	309	NULL	1	10.00	Using index condition

```
1 row in set, 1 warning (0.00 sec)
```

根据前面的两个字段 name，status 查询是走索引的，但是最后一个条件address 没有用到索引。

什么情况下索引会失效？

3). 不要在索引列上进行运算操作，索引将失效。

```
mysql> select * from tb_seller where substring(name,3,2) = '科技';
```

sellerid	name	nickname	password	status	address	createtime
baidu	百度科技有限公司	百度小店	e10adc3949ba59abbe56e057f20f883e	1	北京市	2088-01-01 12:00:00
huawei	华为科技有限公司	华为小店	e10adc3949ba59abbe56e057f20f883e	0	北京市	2088-01-01 12:00:00
luoji	罗技科技有限公司	罗技小店	e10adc3949ba59abbe56e057f20f883e	1	北京市	2088-01-01 12:00:00
ourpalm	掌趣科技股份有限公司	掌趣小店	e10adc3949ba59abbe56e057f20f883e	1	北京市	2088-01-01 12:00:00
qiandu	千度科技	千度小店	e10adc3949ba59abbe56e057f20f883e	2	北京市	2088-01-01 12:00:00
sina	新浪科技有限公司	新浪官方旗舰店	e10adc3949ba59abbe56e057f20f883e	1	北京市	2088-01-01 12:00:00
xiaomi	小米科技	小米官方旗舰店	e10adc3949ba59abbe56e057f20f883e	1	西安市	2088-01-01 12:00:00

7 rows in set (0.00 sec)

```
mysql> explain select * from tb_seller where substring(name,3,2) = '科技';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	tb_seller	NULL	ALL	NULL	NULL	NULL	NULL	12	100.00	Using where

1 row in set, 1 warning (0.00 sec)

什么情况下索引会失效？

4). 字符串不加单引号，造成索引失效。

```
mysql> explain select * from tb_seller where name = '科技' and status = '0';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	tb_seller	NULL	ref	tb_seller_index	tb_seller_index	309	const,const	1	100.00	NULL

```
1 row in set, 1 warning (0.00 sec)
```

```
mysql> explain select * from tb_seller where name = '科技' and status = 0;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	tb_seller	NULL	ref	tb_seller_index	tb_seller_index	303	const	1	10.00	Using index condition

```
1 row in set, 2 warnings (0.00 sec)
```

由于，在查询是，没有对字符串加单引号，MySQL的查询优化器，会自动的进行类型转换，造成索引失效。

什么情况下索引会失效？

5).以%开头的Like模糊查询，索引失效。如果仅仅是尾部模糊匹配，索引不会失效。如果是头部模糊匹配，索引失效。

```
mysql> explain select sellerid,name from tb_seller where name like '%黑马程序员';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	tb_seller	NULL	ALL	NULL	NULL	NULL	NULL	12	11.11	Using where

```
1 row in set, 1 warning (0.00 sec)
```

```
mysql> explain select sellerid,name from tb_seller where name like '黑马程序员';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	tb_seller	NULL	ALL	NULL	NULL	NULL	NULL	12	11.11	Using where

```
1 row in set, 1 warning (0.00 sec)
```

```
mysql> explain select sellerid,name from tb_seller where name like '黑马程序员%';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	tb_seller	NULL	range	tb_seller_index	tb_seller_index	303	NULL	1	100.00	Using index condition

```
1 row in set, 1 warning (0.00 sec)
```



什么情况下索引会失效？

① 违反最左原则 面试官：什么情况下索引会失效？

② 范围查询不连续 候选人：嗯，这个情况比较多，我说一些自己的经验，以前遇到过的

③ 不要在索引字段上使用函数或运算 比如，索引在使用的时候没有遵循最左匹配法则，第二个是，模糊查询，如果%号在前面也会导致索引失效。如果在添加索引的字段上进行了运算操作或者类型转换也都会导致索引失效。

④ 字符串不连续 我们之前还遇到过一个就是，如果使用了复合索引，中间使用了范围查询，右边的条件索引也会失效

⑤ 以%开头的模糊查询 所以，通常情况下，想要判断出这条sql是否有索引失效的情况，可以使用explain执行计划来分析



谈一谈你对sql的优化的经验

- 表的设计优化
- 索引优化 参考优化创建原则和索引失效
- SQL语句优化
- 主从复制、读写分离
- 分库分表 后面有专门章节介绍

谈谈你对sql的优化的经验

● 表的设计优化（参考阿里开发手册《嵩山版》）

- ① 比如设置合适的数值（tinyint int bigint），要根据实际情况选择
- ② 比如设置合适的字符串类型（char和varchar）char定长效率高，varchar可变长度，效率稍低

● SQL语句优化

- ① SELECT语句务必指明字段名称（避免直接使用select *）
- ② SQL语句要避免造成索引失效的写法
- ③ 尽量用union all代替union union会多一次过滤，效率低
- ④ 避免在where子句中对字段进行表达式操作
- ⑤ Join优化 能用innerjoin 就不用left join right join，如必须使用 一定要以小表为驱动，
内连接会对两个表进行优化，优先把小表放到外边，把大表放到里边。left join 或 right join，不会重新调整顺序

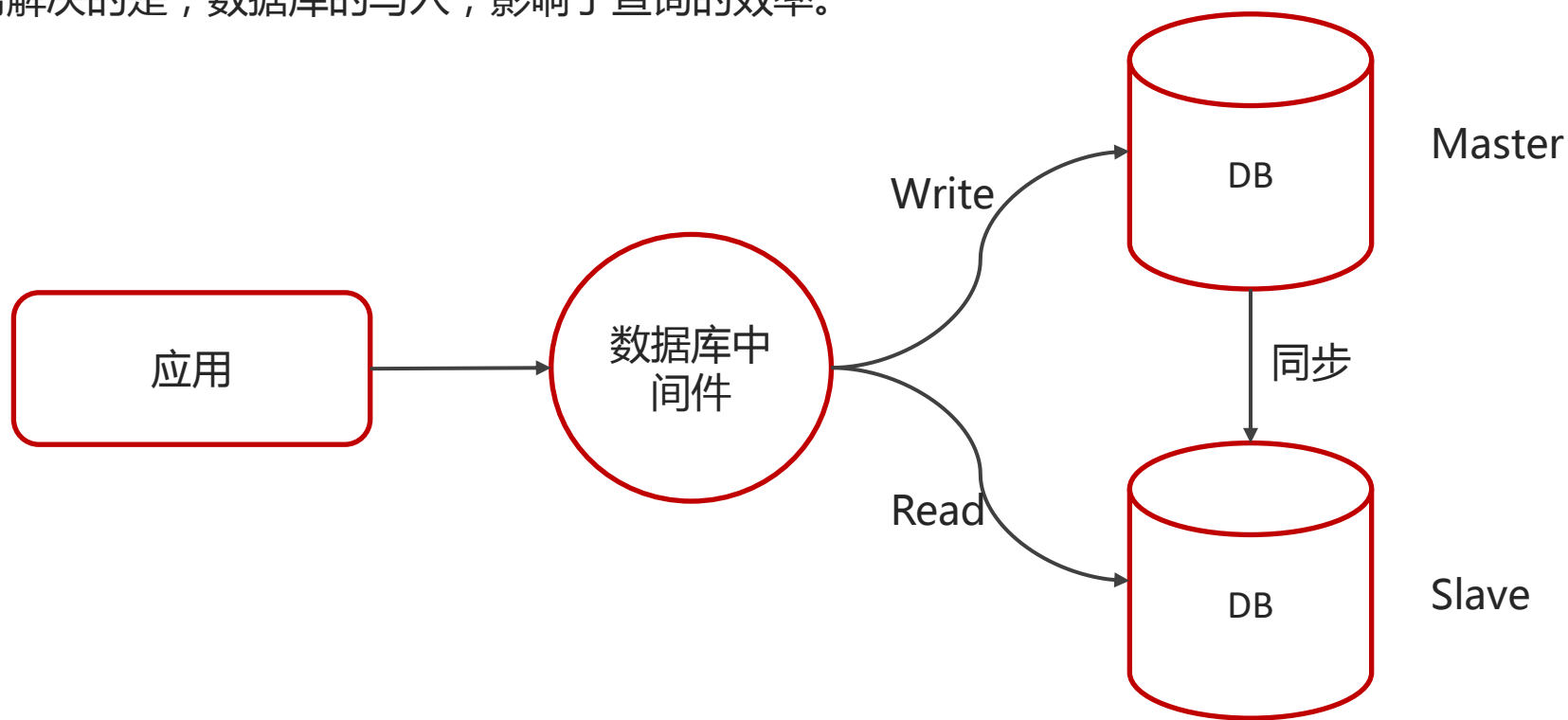
```
for (int i = 0; i < 3; i++) {  
    select * from t_user where id > 2  
    union all | union  
    select * from t_user where id < 5  
}
```


谈谈你对sql的优化的经验

- 主从复制、读写分离

如果数据库的使用场景读的操作比较多的时候，为了避免写的操作所造成的性能影响 可以采用读写分离的架构。

读写分离解决的是，数据库的写入，影响了查询的效率。





谈一谈你对sql的优化的经验

1. 表的设计优化，数据
2. 索引优化，索引创建
3. sql语句优化，避免
4. 主从复制、读写分离
5. 分库分表

面试官：sql的优化的经验

候选人：嗯，这个在项目还是挺常见的，当然如果直说sql优化的话，我们会从这几方面考虑，比如建表的时候、使用索引、sql语句的编写、主从复制，读写分离，还有一个是如果量比较大的话，可以考虑分库分表

面试官：创建表的时候，你们是如何优化的呢？

候选人：这个我们主要参考的阿里出的那个开发手册《嵩山版》，就比如，在定义字段的时候需要结合字段的内容来选择合适的类型，如果是数值的话，像tinyint、int、bigint这些类型，要根据实际情况选择。如果是字符串类型，也是结合存储的内容来选择char和varchar或者text类型

面试官：那在使用索引的时候，是如何优化呢？

候选人：【参考索引创建原则 进行描述】

面试官：你平时对sql语句做了哪些优化呢？

候选人：嗯，这个也有很多，比如SELECT语句务必指明字段名称，不要直接使用select *，还有就是要注意SQL语句避免造成索引失效的写法；如果是聚合查询，尽量用union all代替union，union会多一次过滤，效率比较低；如果是表关联的话，尽量使用innerjoin，不要使用left join right join，如必须使用一定要以小表为驱动

MySQL-其他面试题

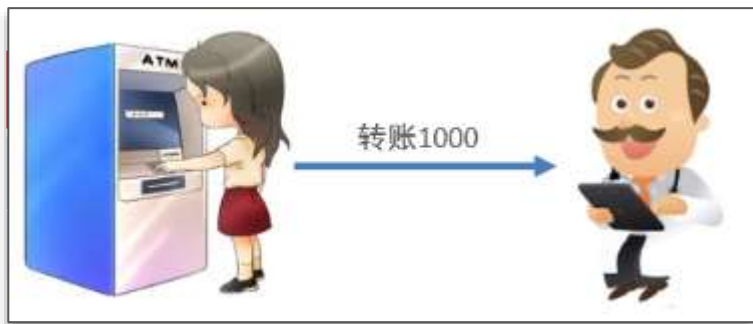




事务的特性是什么？可以详细说一下吗？

ACID

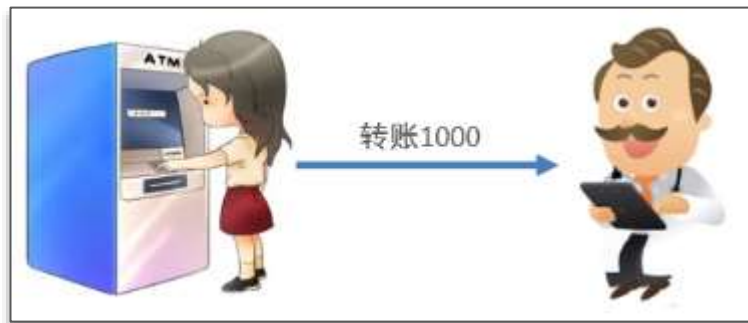
事务是一组操作的集合，它是一个不可分割的工作单位，事务会把所有的操作作为一个整体一起向系统提交或撤销操作请求，即这些操作要么同时成功，要么同时失败。



id	name	money
1	张三	1000
2	李四	3000

ACID是什么？可以详细说一下吗？

- 原子性 (**A**tomicity)：事务是不可分割的最小操作单元，要么全部成功，要么全部失败。
- 一致性 (**C**onsistency)：事务完成时，必须使所有的数据都保持一致状态。
- 隔离性 (**I**solation)：数据库系统提供的隔离机制，保证事务在不受外部并发操作影响的独立环境下运行。
- 持久性 (**D**urability)：事务一旦提交或回滚，它对数据库中的数据的改变就是永久的。





事务的特性是什么？可以详细说一下吗？

- 原子性(Atomicity)
- 一致性(Consistency)
- 隔离性(Isolation)
- 持久性(Durability)

结合转账案例进行说明

面试官：事务的特性是什么？可以详细说一下吗？

候选人：嗯，这个比较清楚，ACID，分别指的是：原子性、一致性、隔离性、持久性；我举个例子：

A向B转账500，转账成功，A扣除500元，B增加500元，原子操作体现在要么都成功，要么都失败

在转账的过程中，数据要一致，A扣除了500，B必须增加500

在转账的过程中，隔离性体现在A像B转账，不能受其他事务干扰

在转账的过程中，持久性体现在事务提交后，要把数据持久化（可以说是落盘操作）

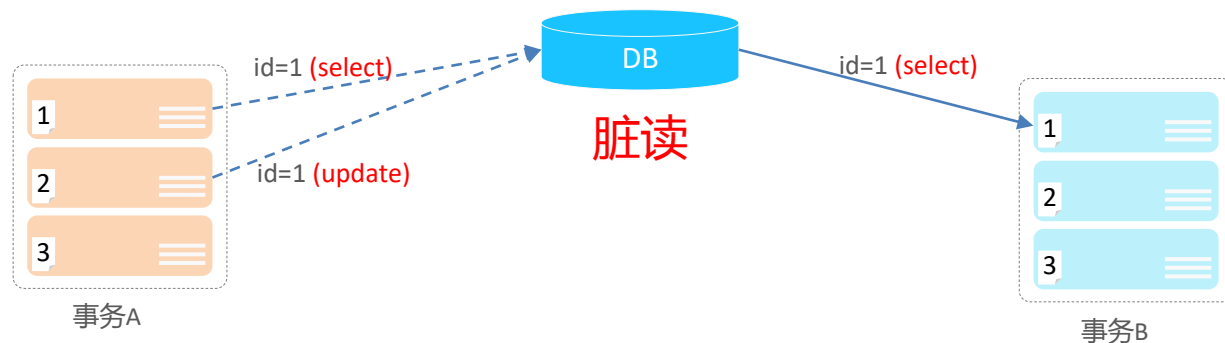


并发事务带来哪些问题？怎么解决这些问题呢？MySQL的默认隔离级别是？

- 并发事务问题：脏读、不可重复读、幻读
- 隔离级别：读未提交、读已提交、**可重复读**、串行化

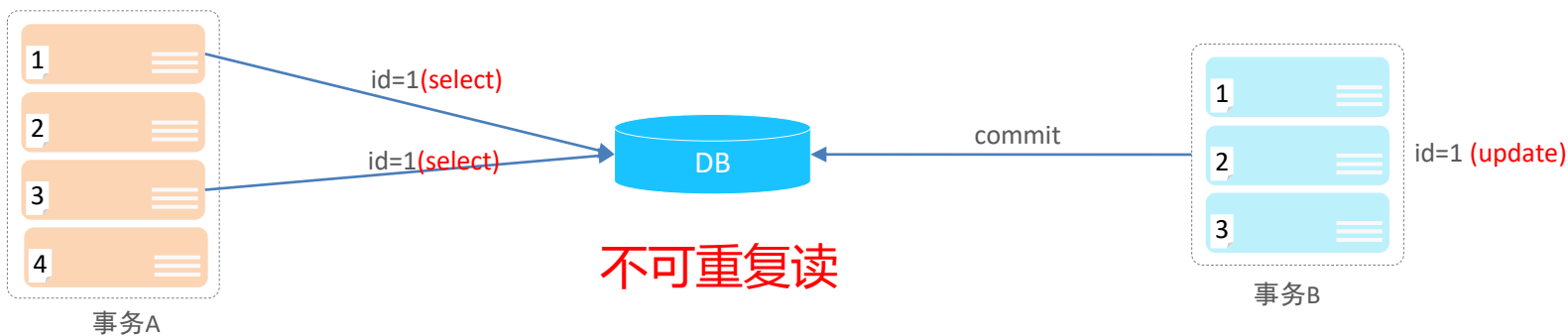
并发事务问题

问题	描述
脏读	一个事务读到另外一个事务还没有提交的数据。
不可重复读	一个事务先后读取同一条记录，但两次读取的数据不同，称之为不可重复读。
幻读	一个事务按照条件查询数据时，没有对应的数据行，但是在插入数据时，又发现这行数据已经存在，好像出现了“幻影”。



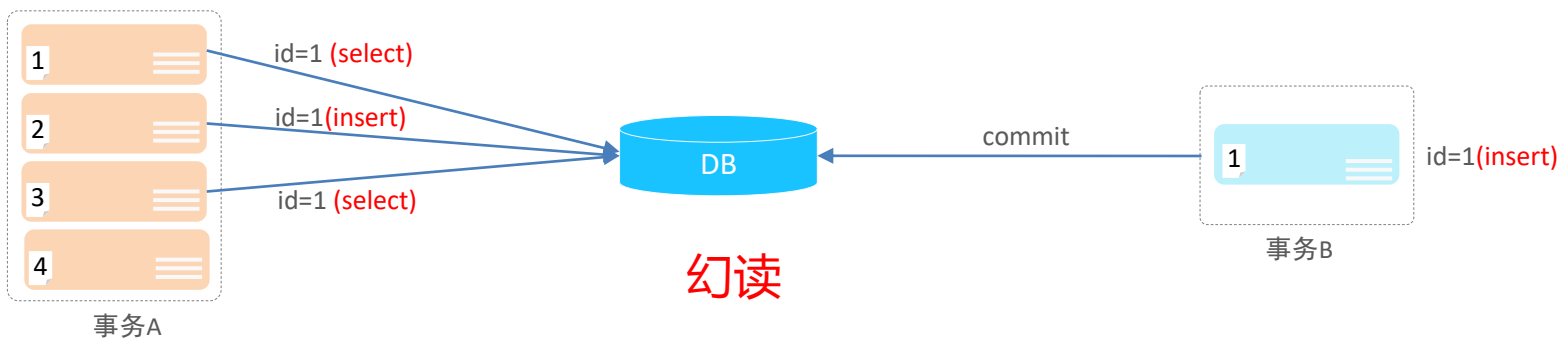
并发事务问题

问题	描述
脏读	一个事务读到另外一个事务还没有提交的数据。
不可重复读	一个事务先后读取同一条记录，但两次读取的数据不同，称之为不可重复读。
幻读	一个事务按照条件查询数据时，没有对应的数据行，但是在插入数据时，又发现这行数据已经存在，好像出现了“幻影”。



并发事务问题

问题	描述
脏读	一个事务读到另外一个事务还没有提交的数据。
不可重复读	一个事务先后读取同一条记录，但两次读取的数据不同，称之为不可重复读。
幻读	一个事务按照条件查询数据时，没有对应的数据行，但是在插入数据时，又发现这行数据已经存在，好像出现了“幻影”。

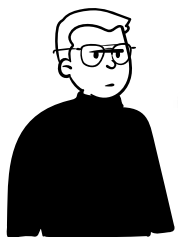


怎么解决并发事务的问题呢？

解决方案：对事务进行隔离

隔离级别	脏读	不可重复读	幻读
Read uncommitted 未提交读	√	√	√
Read committed 读已提交	×	√	√
Repeatable Read(默认) 可重复读	×	×	√
Serializable 串行化	×	×	×

注意：事务隔离级别越高，数据越安全，但是性能越低。



并发事务带来哪些问题？怎么解决这些问题呢？MySQL的默认隔离级别是？

- 并发事务的问题：

- ① 脏读：一个事务读
- ② 不可重复读：一个
- ③ 幻读：一个事务按
- 存在，好像出现了

- 隔离级别：

- ① READ UNCOMMITT
- ② READ COMMITTED
- ③ REPEATABLE READ
- ④ SERIALIZABLE 串行

面试官：并发事务带来哪些问题？

候选人：

我们在项目开发中，多个事务并发进行是经常发生的，并发也是必然的，有可能导致一些问题

第一是脏读，当一个事务正在访问数据并且对数据进行了修改，而这种修改还没有提交到数据库中，这时另外一个事务也访问了这个数据，因为这个数据是还没有提交的数据，那么另外一个事务读到的这个数据是“脏数据”，依据“脏数据”所做的操作可能是不正确的。

第二是不可重复读：比如在一个事务内多次读同一数据。在这个事务还没有结束时，另一个事务也访问该数据。那么，在第一个事务中的两次读数据之间，由于第二个事务的修改导致第一个事务两次读取的数据可能不太一样。这就发生了在一个事务内两次读到的数据是不一样的情况，因此称为不可重复读。

第三是幻读（Phantom read）：幻读与不可重复读类似。它发生在一个事务（T1）读取了几行数据，接着另一个并发事务（T2）插入了一些数据时。在随后的查询中，第一个事务（T1）就会发现多了一些原本不存在的记录，就好像发生了幻觉一样，所以称为幻读。

面试官：怎么解决这些问题呢？MySQL的默认隔离级别是？

候选人：解决方案是对事务进行隔离

MySQL支持四种隔离级别，分别有：


第一个是，未提交读（read uncommitted）它解决不了刚才提出的所有问题，一般项目中也不用这个。第二个是读已提交（read committed）它能解决脏读的问题的，但是解决不了不可重复读和幻读。第三个是可重复读（repeatable read）它能解决脏读和不可重复读，但是解决不了幻读，这个也是mysql默认的隔离级别。第四个是串行化（serializable）它可以解决刚才提出来的所有问题，但是由于让是事务串行执行的，性能比较低。所以，我们一般使用的都是mysql默认的隔离级别:可重复读

这行数据已经

undo log和redo log的区别

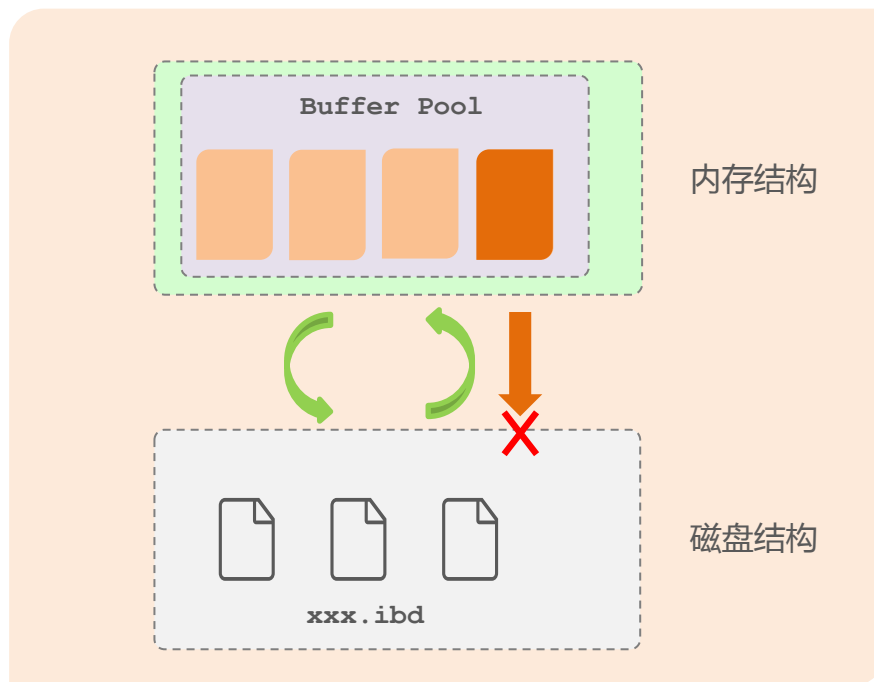


- **缓冲池 (buffer pool)** :主内存中的一个区域，里面可以缓存磁盘上经常操作的真实数据，在执行增删改查操作时，先操作缓冲池中的数据（若缓冲池没有数据，则从磁盘加载并缓存），以一定频率刷新到磁盘，从而减少磁盘IO，加快处理速度
- **数据页 (page)** :是InnoDB 存储引擎磁盘管理的最小单元，每个页的大小默认为 16KB。页中存储的是行数据



update
update
delete

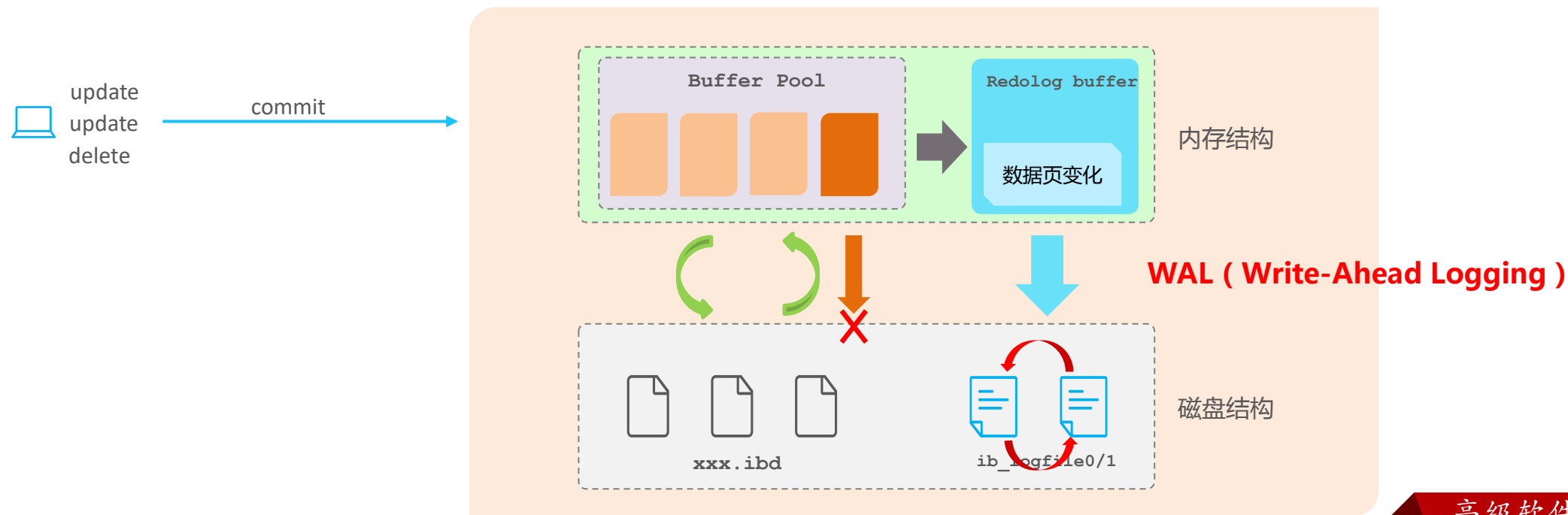
commit



redo log

重做日志，记录的是事务提交时数据页的物理修改，是**用来实现事务的持久性**。

该日志文件由两部分组成：重做日志缓冲（redo log buffer）以及重做日志文件（redo log file），前者是在内存中，后者在磁盘中。当事务提交之后会把所有修改信息都存到该日志文件中，用于在刷新脏页到磁盘，发生错误时，进行数据恢复使用。



undo log

回滚日志，用于记录数据被修改前的信息，作用包含两个：**提供回滚**和 **MVCC**(多版本并发控制)。undo log和redo log记录物理日志不一样，它是**逻辑日志**。

- 可以认为当delete一条记录时，undo log中会记录一条对应的insert记录，反之亦然，
- 当update一条记录时，它记录一条对应相反的update记录。当执行rollback时，就可以从undo log中的逻辑记录读取到相应内容并进行回滚。

undo log可以实现事务的一致性和原子性



undo log和redo log的区别

- redo log: 记录的是数据页的物理变化，服务宕机可用来同步数据
- undo log : 记录的是逻辑日志，当事务回滚时，通过逆操作恢复原来的数据
- redo log保证了事务的持久性，undo log保证了事务的原子性和一致性

面试官：undo log和redo log的区别

候选人：好的，其中redo log日志记录的是数据页的物理变化，服务宕机可用来同步数据，而undo log不同，它主要记录的是逻辑日志，当事务回滚时，通过逆操作恢复原来的数据，比如我们删除一条数据的时候，就会在undo log日志文件中新增一条delete语句，如果发生回滚就执行逆操作；

redo log保证了事务的持久性，undo log保证了事务的原子性和一致性



undo log和redo log的区别

- redo log: 记录的是数据页的物理变化，服务宕机可用来同步数据
- undo log : 记录的是逻辑日志，当事务回滚时，通过逆操作恢复原来的数据
- redo log保证了事务的持久性，undo log保证了事务的原子性和一致性



好的，事务中的隔离性是如何保证的呢？

锁：排他锁（如一个事务获取了一个数据行的排他锁，其他事务就不能再获取该行的其他锁）

mvcc：多版本并发控制

你解释一下MVCC？

解释一下MVCC

全称 **M**ulti-**V**ersion **C**oncurrency **C**ontrol，多版本并发控制。指维护一个数据的多个版本，使得读写操作没有冲突

MVCC的具体实现，主要依赖于数据库记录中的**隐式字段**、**undo log日志**、**readView**。

id	age	name
30	3	A30

事务2	事务3	事务4	事务5
开始事务	开始事务	开始事务	开始事务
修改id为30记录，age改为3		查询id为30的记录	
提交事务			
	修改id为30记录，name改为A3		
			查询id为30的记录
	提交事务		
		修改id为30记录，age改为10	
		查询id为30的记录	
			查询id为30的记录
		提交事务	

查询的是哪个事务版本的记录

MVCC-实现原理

- 记录中的隐藏字段

id	age	name
1	1	tom
3	3	cat

id	age	name	DB_TRX_ID	DB_ROLL_PTR	DB_ROW_ID
----	-----	------	-----------	-------------	-----------

隐藏字段	含义
DB_TRX_ID	最近修改事务ID，记录插入这条记录或最后一次修改该记录的事务ID。
DB_ROLL_PTR	回滚指针，指向这条记录的上一个版本，用于配合undo log，指向上一个版本。
DB_ROW_ID	隐藏主键，如果表结构没有指定主键，将会生成该隐藏字段。

MVCC-实现原理

- undo log

回滚日志，在insert、update、delete的时候产生的便于数据回滚的日志。

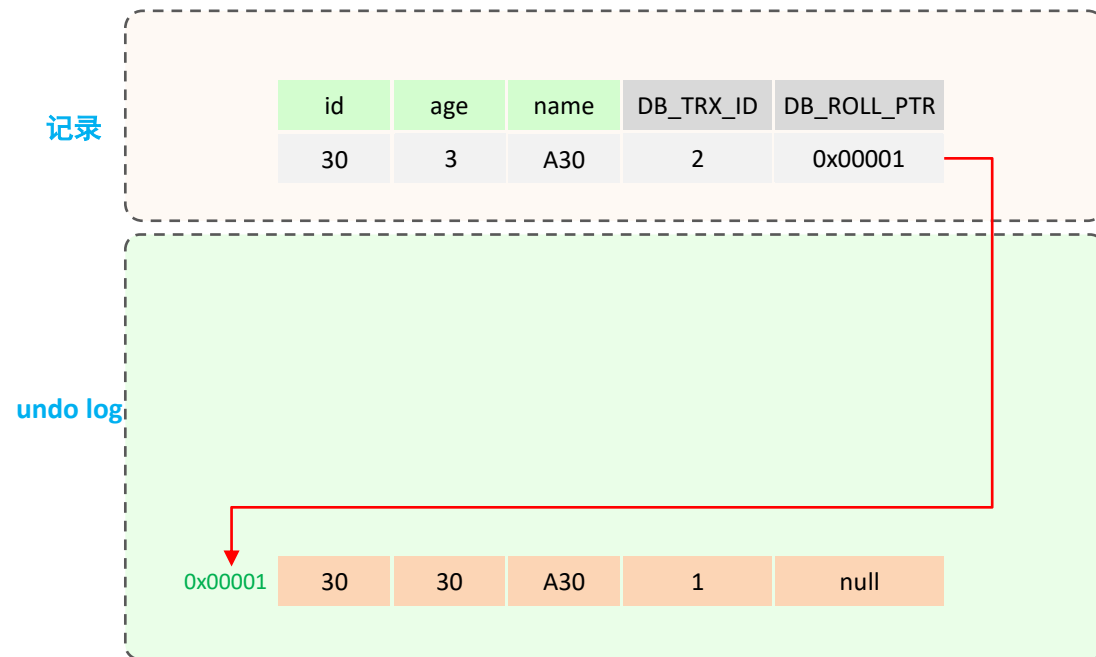
当insert的时候，产生的undo log日志只在回滚时需要，在事务提交后，可被立即删除。

而update、delete的时候，产生的undo log日志不仅在回滚时需要，mvcc版本访问也需要，不会立即被删除。

MVCC-实现原理

- undo log版本链

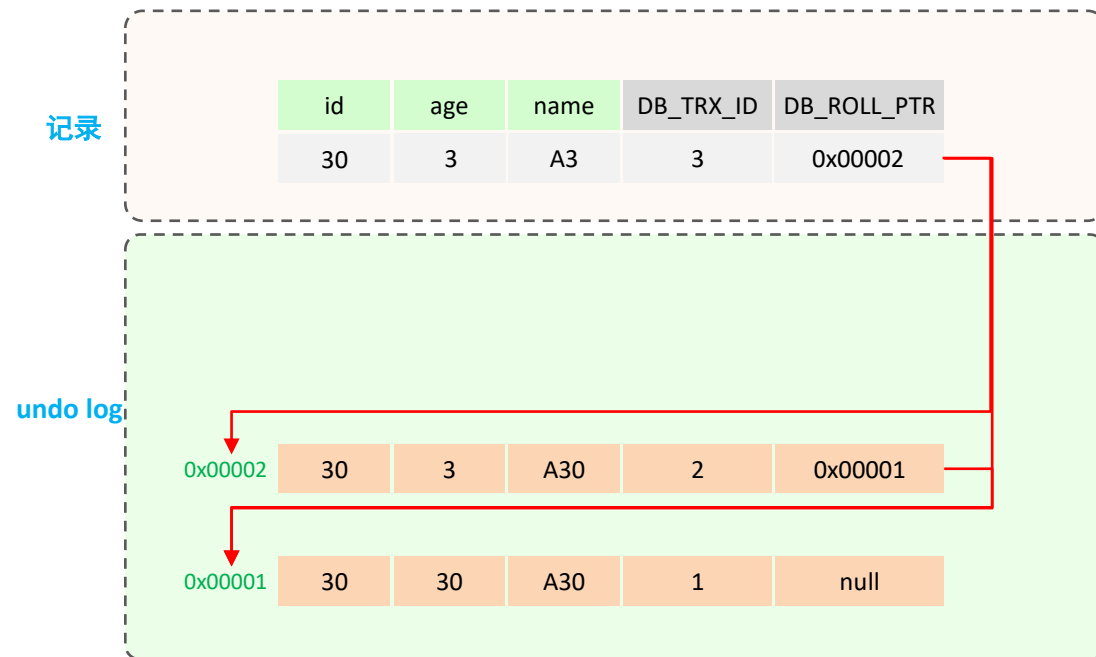
事务2	事务3	事务4	事务5
开始事务	开始事务	开始事务	开始事务
修改id为30记录, age改为3		查询id为30的记录	
提交事务			
	修改id为30记录, name改为A3		
			查询id为30的记录
	提交事务		
		修改id为30记录, age改为10	
		查询id为30的记录	
			查询id为30的记录
		提交事务	



MVCC-实现原理

- undo log版本链

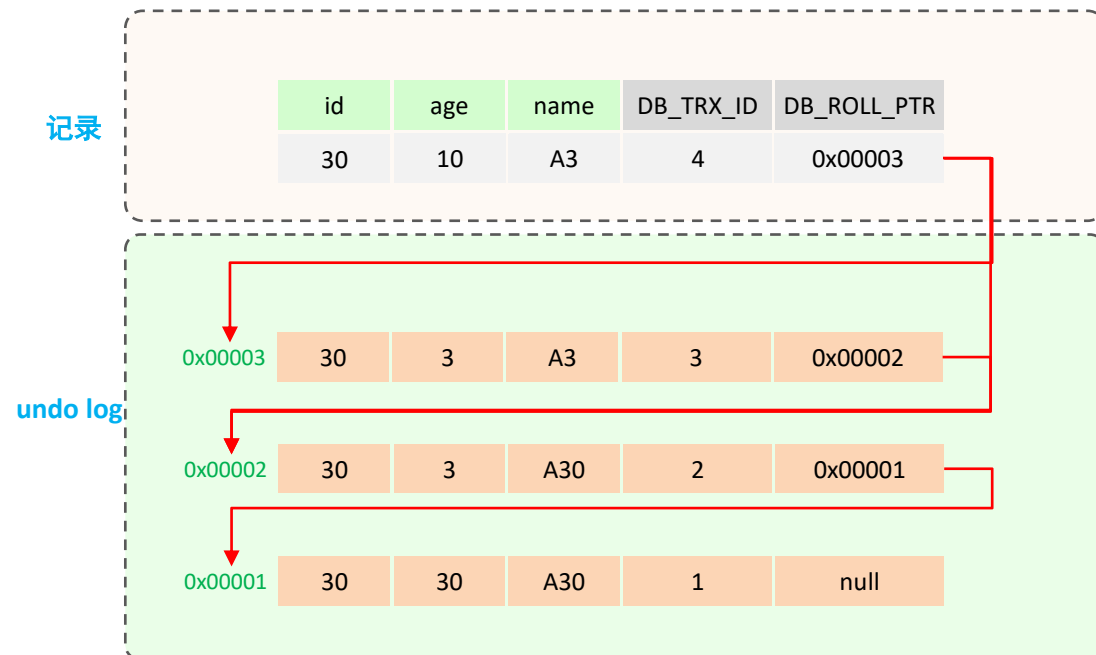
事务2	事务3	事务4	事务5
开始事务	开始事务	开始事务	开始事务
修改id为30记录, age改为3		查询id为30的记录	
提交事务			
	修改id为30记录, name改为A3		
	提交事务		查询id为30的记录
		修改id为30记录, age改为10	
		查询id为30的记录	
			查询id为30的记录
		提交事务	



MVCC-实现原理

- undo log版本链

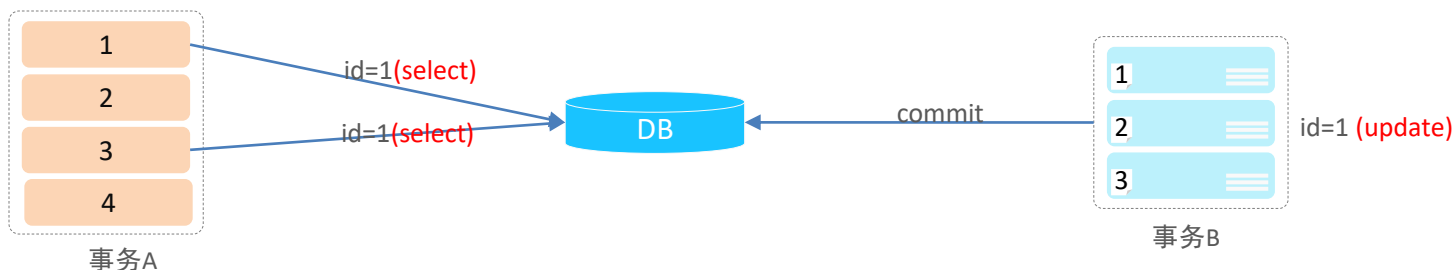
事务2	事务3	事务4	事务5
开始事务	开始事务	开始事务	开始事务
修改id为30记录, age改为3		查询id为30的记录	
提交事务			
	修改id为30记录, name改为A3		
	提交事务		查询id为30的记录
		修改id为30记录, age改为10	
		查询id为30的记录	
		提交事务	查询id为30的记录



不同事务或相同事务对同一条记录进行修改，会导致该记录的undolog生成一条记录版本链表，链表的头部是最新的旧记录，链表尾部是最早的旧记录。

MVCC-实现原理

- readview



ReadView (读视图) 是 **快照读** SQL执行时MVCC提取数据的依据，记录并维护系统当前活跃的事务（未提交的）id。

- 当前读

读取的是记录的**最新版本**，读取时还要保证其他并发事务不能修改当前记录，会对读取的记录进行加锁。对于我们日常的操作，如：

select ... lock in share mode(共享锁)，select ... for update、update、insert、delete(排他锁)都是一种当前读。

- 快照读

简单的select（不加锁）就是快照读，快照读，读取的是记录数据的可见版本，有可能是历史数据，不加锁，是非阻塞读。

- Read Committed：每次select，都生成一个快照读。
- Repeatable Read：开启事务后第一个select语句才是快照读的地方。

MVCC-实现原理

ReadView中包含了四个核心字段：

字段	含义
m_ids	当前活跃的事务ID集合
min_trx_id	最小活跃事务ID
max_trx_id	预分配事务ID，当前最大事务ID+1（因为事务ID是自增的）
creator_trx_id	ReadView创建者的事务ID

事务2	事务3	事务4	事务5
开始事务	开始事务	开始事务	开始事务
修改id为30记录，age改为3		查询id为30的记录	
提交事务			
	修改id为30记录，name改为A3		
			查询id为30的记录
	提交事务		
		修改id为30记录，age改为10	
		查询id为30的记录	
			查询id为30的记录
		提交事务	

MVCC-实现原理

- readview

trx_id : 代表是当前事务ID。

版本链数据访问规则

- ①. $\text{trx_id} == \text{creator_trx_id}$? 可以访问该版本 → 成立，说明数据是当前这个事务更改的。
- ②. $\text{trx_id} < \text{min_trx_id}$? 可以访问该版本 → 成立，说明数据已经提交了。
- ③. $\text{trx_id} > \text{max_trx_id}$? 不可以访问该版本 → 成立，说明该事务是在ReadView生成后才开启。
- ④. $\text{min_trx_id} \leq \text{trx_id} \leq \text{max_trx_id}$? 如果trx_id不在m_ids中是可以访问该版本的 → 成立，说明数据已经提交。

事务2	事务3	事务4	事务5
开始事务	开始事务	开始事务	开始事务
修改id为30记录，age改为3		查询id为30的记录	
提交事务			
	修改id为30记录，name改为A3		查询id为30的记录
	提交事务		
		修改id为30记录，age改为10	
		查询id为30的记录	
			查询id为30的记录
		提交事务	

不同的隔离级别，生成ReadView的时机不同：

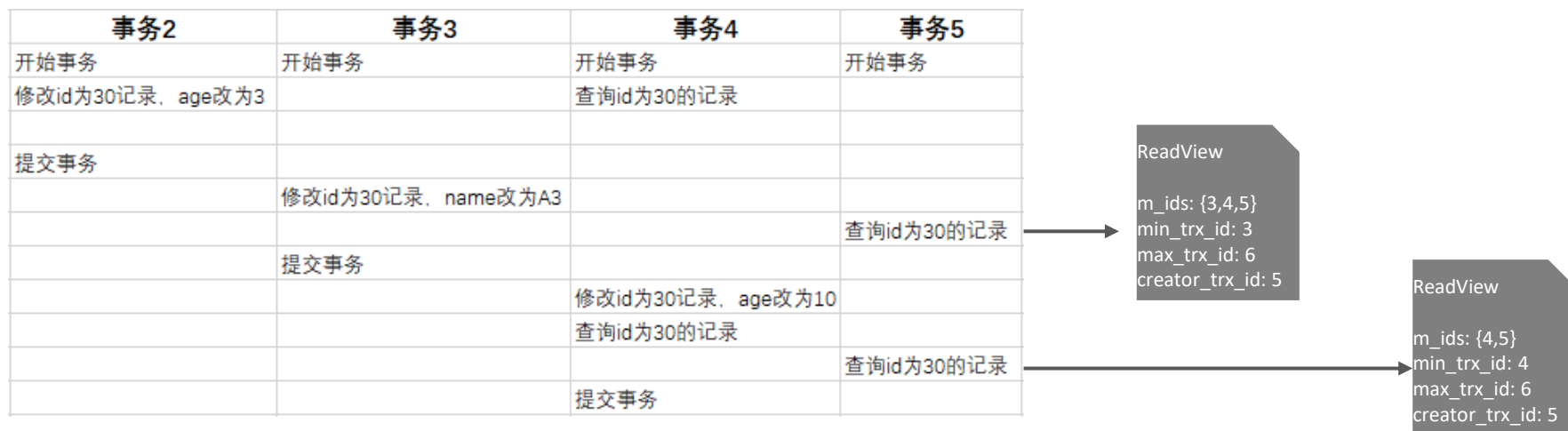
- READ COMMITTED：在事务中每一次执行快照读时生成ReadView。
- REPEATABLE READ：仅在事务中第一次执行快照读时生成ReadView，后续复用该ReadView。

MVCC-实现原理

- readview

RC隔离级别下，在事务中每一次执行快照读时生成ReadView。

RC



MVCC-实现原理

● readview

事务2	事务3	事务4	事务5
开始事务	开始事务	开始事务	开始事务
修改id为30记录, age改为3		查询id为30的记录	
提交事务			
	修改id为30记录, name改为A3		查询id为30的记录
	提交事务		
		修改id为30记录, age改为10	
		查询id为30的记录	
		提交事务	查询id为30的记录

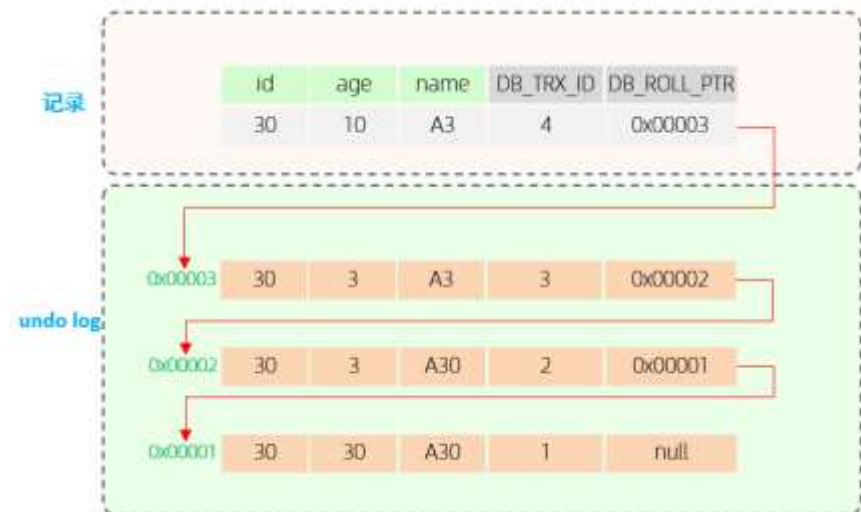
ReadView

m_ids: {3,4,5}
min_trx_id: 3
max_trx_id: 6
creator_trx_id: 5

ReadView

m_ids: {4,5}
min_trx_id: 4
max_trx_id: 6
creator_trx_id: 5

RC



①. ⁵ $trx_id == creator_trx_id$? 可以访问该版本 ➡ 成立, 说明数据是当前这个事务更改的。

②. ³ $trx_id < min_trx_id$? 可以访问该版本 ➡ 成立, 说明数据已经提交了。

③. ⁶ $trx_id > max_trx_id$? 不可以访问该版本 ➡ 成立, 说明该事务是在ReadView生成后才开启。

④. ³ $min_trx_id \leq trx_id \leq max_trx_id$? 如果⁶ trx_id 不在^{3,4,5} m_ids 中是 可以访问该版本的 ➡ 成立, 说明数据已经提交。

MVCC-实现原理

● readview

事务2	事务3	事务4	事务5
开始事务	开始事务	开始事务	开始事务
修改id为30记录, age改为3		查询id为30的记录	
提交事务			
	修改id为30记录, name改为A3		查询id为30的记录
	提交事务		
		修改id为30记录, age改为10	
		查询id为30的记录	
		提交事务	查询id为30的记录

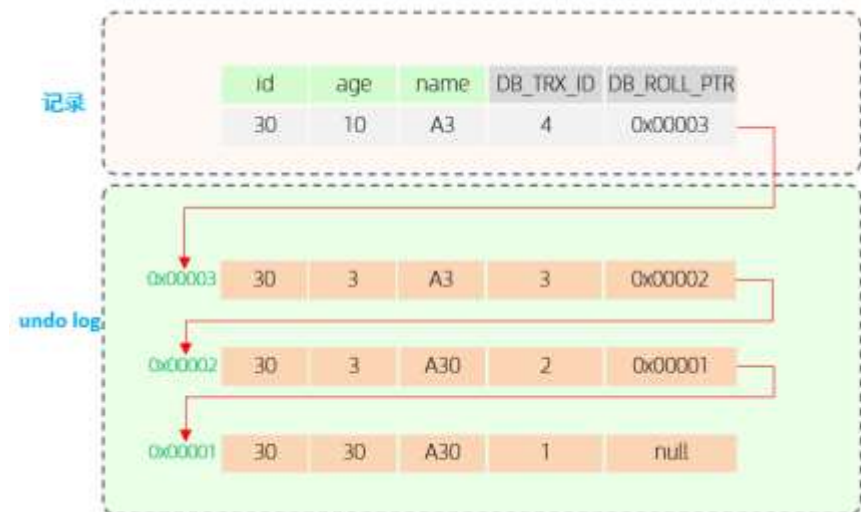
ReadView

m_ids: {3,4,5}
min_trx_id: 3
max_trx_id: 6
creator_trx_id: 5

ReadView

m_ids: {4,5}
min_trx_id: 4
max_trx_id: 6
creator_trx_id: 5

RC



①. ⁵ $trx_id == creator_trx_id$? 可以访问该版本 ➡ 成立, 说明数据是当前这个事务更改的。

②. ⁴ $trx_id < min_trx_id$? 可以访问该版本 ➡ 成立, 说明数据已经提交了。

③. ⁶ $trx_id > max_trx_id$? 不可以访问该版本 ➡ 成立, 说明该事务是在ReadView生成后才开启。

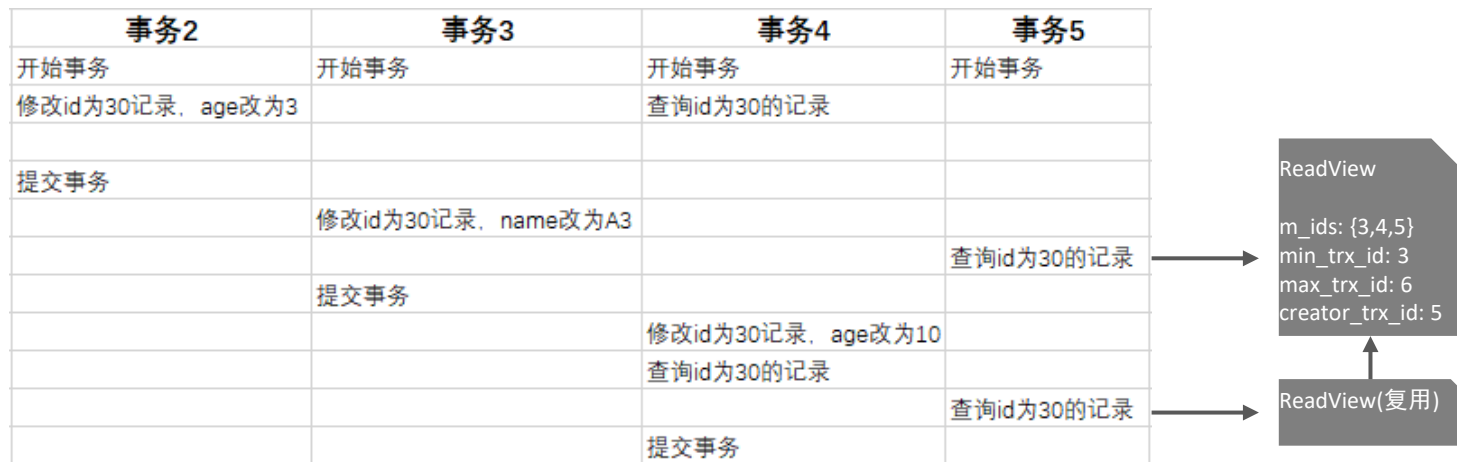
④. ⁴ $min_trx_id \leq trx_id \leq max_trx_id$? 如果⁶ trx_id 不在^{4,5} m_ids 中是 可以 访问该版本的 ➡ 成立, 说明数据已经提交。

MVCC-实现原理

- readview

RR隔离级别下，仅在事务中第一次执行快照读时生成ReadView，后续复用该ReadView。

RR





好的，事务中的隔离性是如何保证的呢？(你解释一下MVCC)

MySQL中的多版本并发控制。指维护一个数据的多个版本，使得读写操作没有冲突

● 隐藏字段：

① trx_id(事务id)，记录

② roll_pointer(回滚指针)

● undo log：

① 回滚日志，存储老版本

② 版本链：多个事务并行

● readView解决的是一

➢ 根据readView

➢ 不同的隔离级别

面试官：事务中的隔离性是如何保证的呢？(你解释一下MVCC)

候选人：事务的隔离性是由锁和mvcc实现的。

其中mvcc的意思是多版本并发控制。指维护一个数据的多个版本，使得读写操作没有冲突，它的底层实现主要是分为了三个部分，第一个是隐藏字段，第二个是undo log日志，第三个是readView读视图

隐藏字段是指：在mysql中给每个表都设置了隐藏字段，有一个是trx_id(事务id)，记录每一次操作的事务id，是自增的；另一个字段是roll_pointer(回滚指针)，指向上一个版本的事务版本记录地址

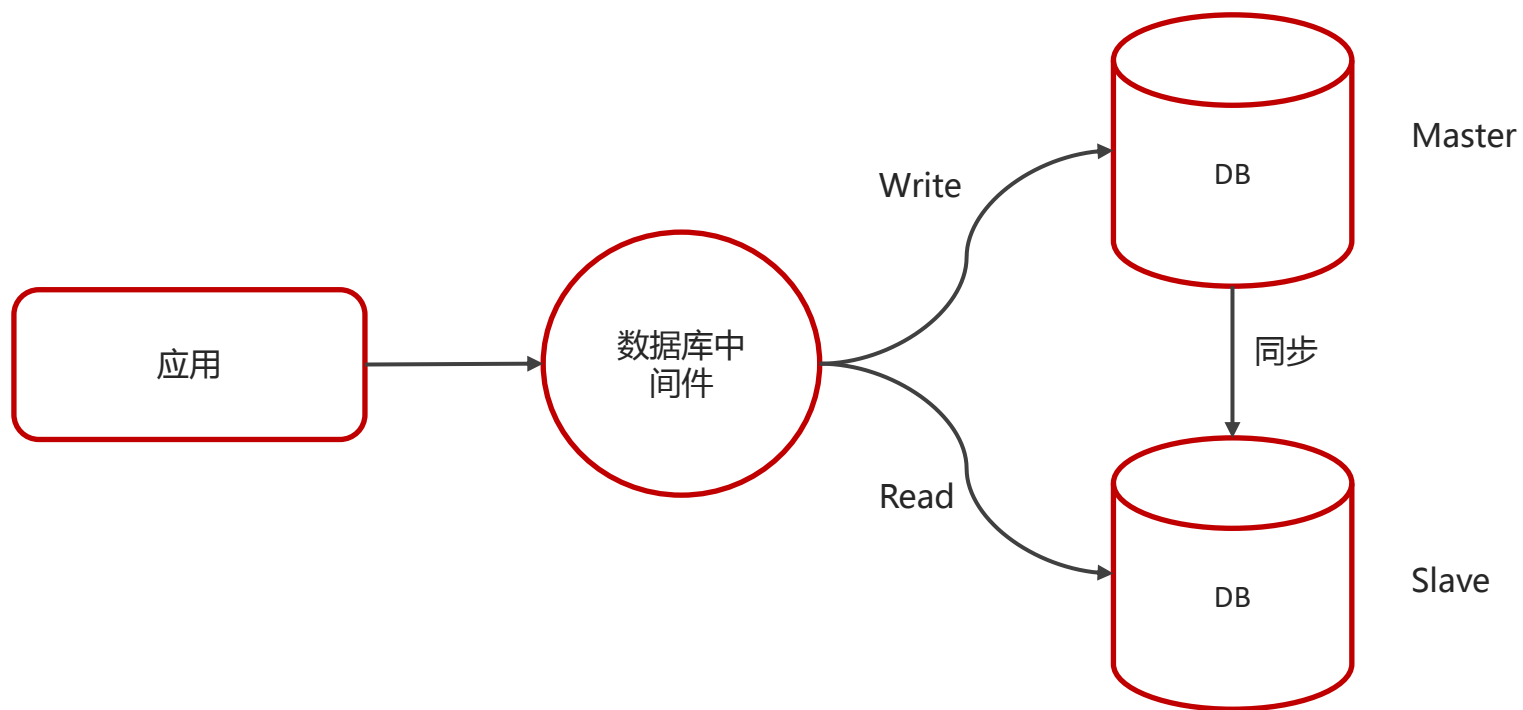
undo log主要的作用是记录回滚日志，存储老版本数据，在内部会形成一个版本链，在多个事务并行操作某一行记录，记录不同事务修改数据的版本，通过roll_pointer指针形成一个链表

readView解决的是一个事务查询选择版本的问题，在内部定义了一些匹配规则和当前的一些事务id判断该访问那个版本的数据，不同的隔离级别快照读是不一样的，最终的访问的结果不一样。如果是rc隔离级别，每一次执行快照读时生成ReadView，如果是rr隔离级别仅在事务中第一次执行快照读时生成ReadView，后续复用

RC：每一次执行快照读时生成ReadView

RR：仅在事务中第一次执行快照读时生成ReadView，后续复用

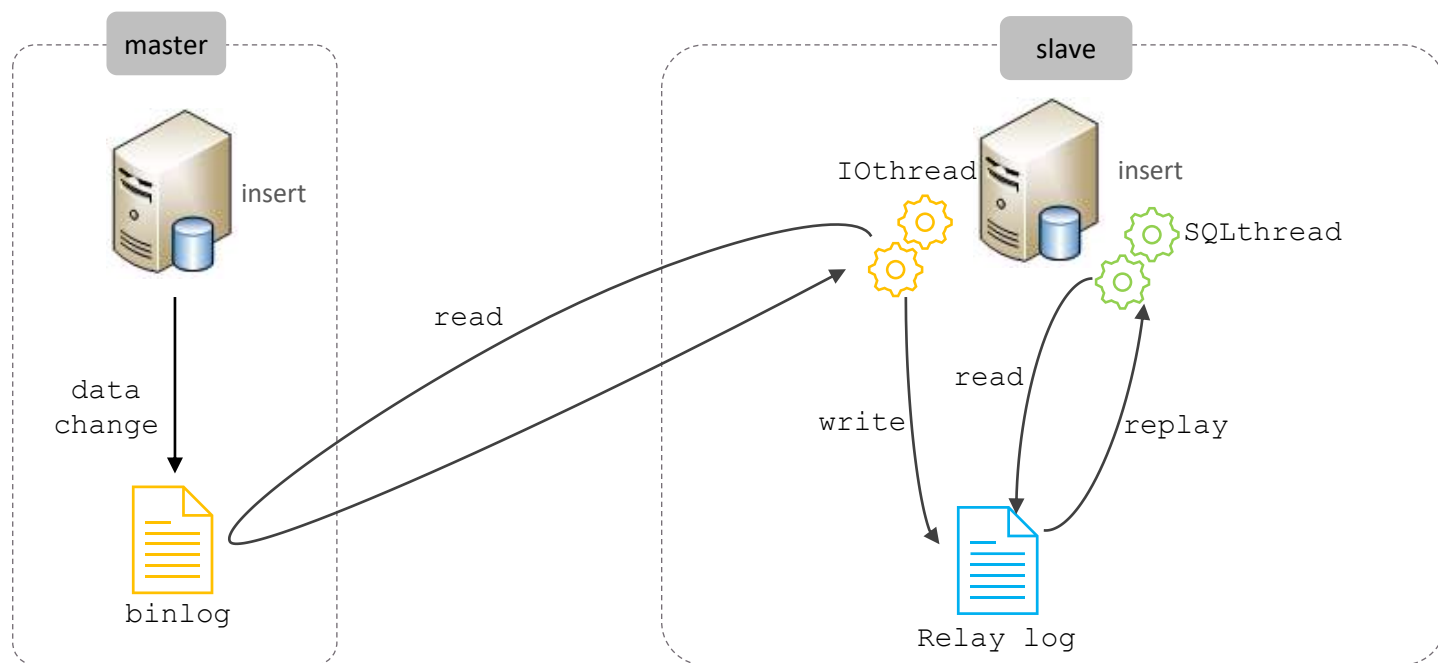
MySQL主从同步原理



主从同步原理

MySQL主从复制的核心就是二进制日志

二进制日志（BINLOG）记录了所有的 DDL（数据定义语言）语句和 DML（数据操纵语言）语句，但不包括数据查询（SELECT、SHOW）语句。



复制分成三步：

1. Master 主库在事务提交时，会把数据变更记录在二进制日志文件 Binlog 中。
2. 从库读取主库的二进制日志文件 Binlog，写入到从库的中继日志 Relay Log。
3. slave重做中继日志中的事件，将改变反映它自己的数据。

主从同步原理



MySQL主从复制的核心就是二进制日志binlog(DDL (数据定义语言) 语句和 DML (数据操纵语言) 语句)

- ① 主库在事务提交时，会把数据变更记录在二进制日志文件 Binlog 中。
- ② 从库读取主库的二进制日志文件 Binlog，写入到从库的中继日志 Relay Log。
- ③ 从库重做中继日志中的事件，将改变反映它自己的数据

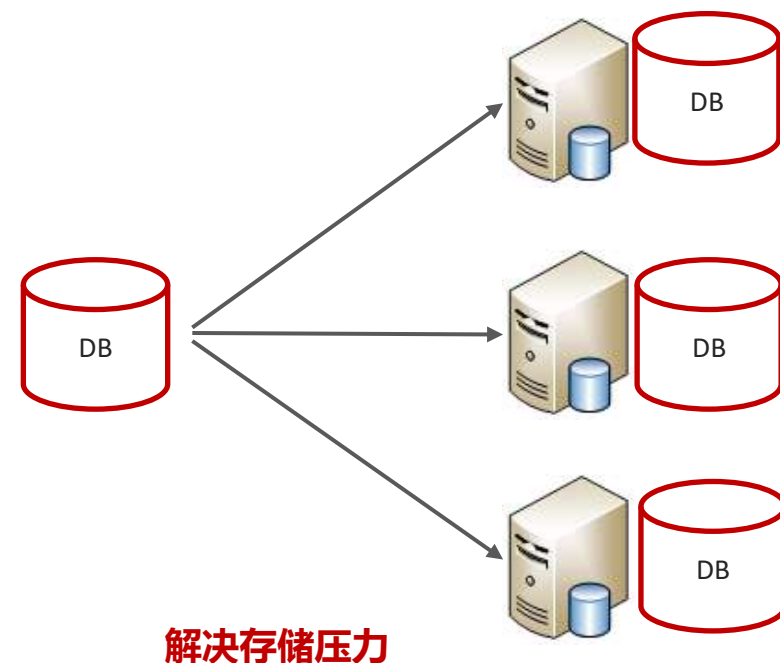
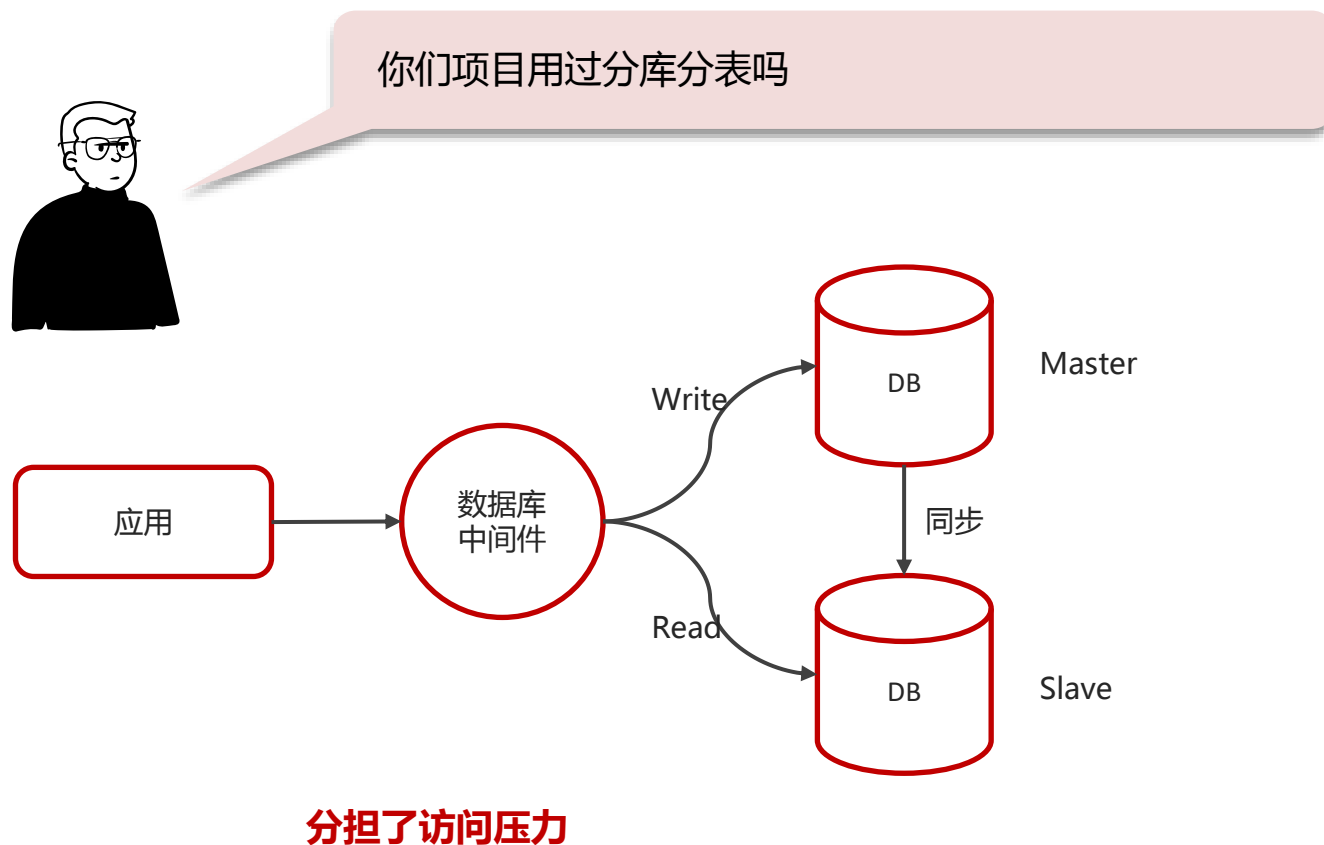
面试官：说一下主从同步的原理？

候选人：

嗯，好的。

MySQL主从复制的核心就是二进制日志，二进制日志记录了所有的 DDL语句和 DML语句
具体的主从同步过程大概的流程是这样的：

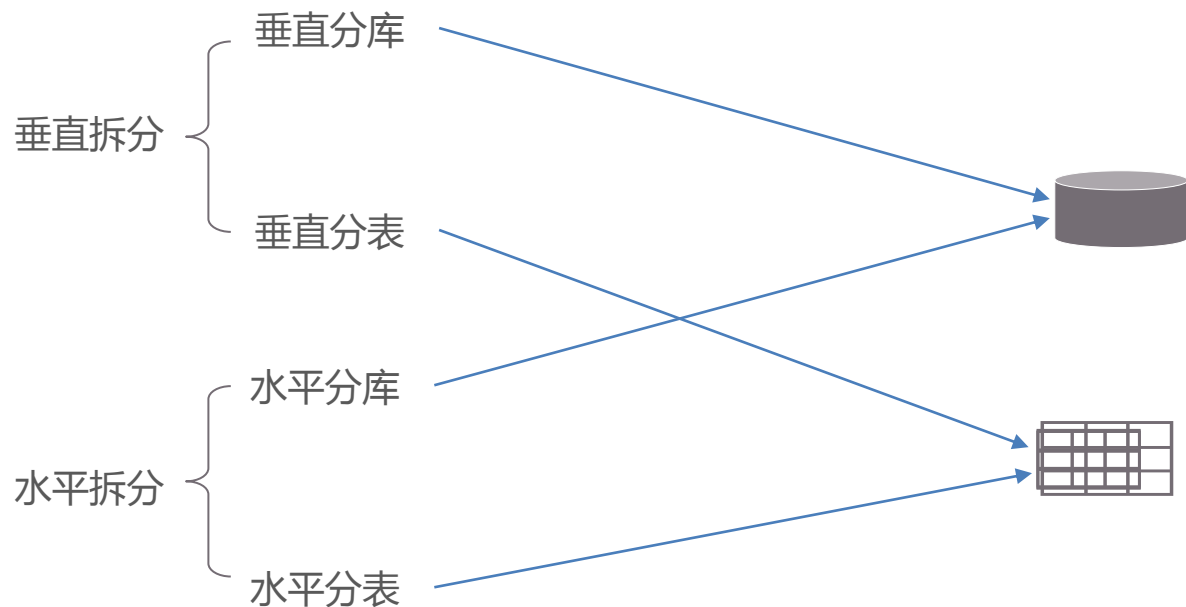
1. Master 主库在事务提交时，会把数据变更记录在二进制日志文件 Binlog 中。
2. 从库读取主库的二进制日志文件 Binlog，写入到从库的中继日志 Relay Log。
3. slave重做中继日志中的事件，将改变反映它自己的数据。



分库分表的时机：

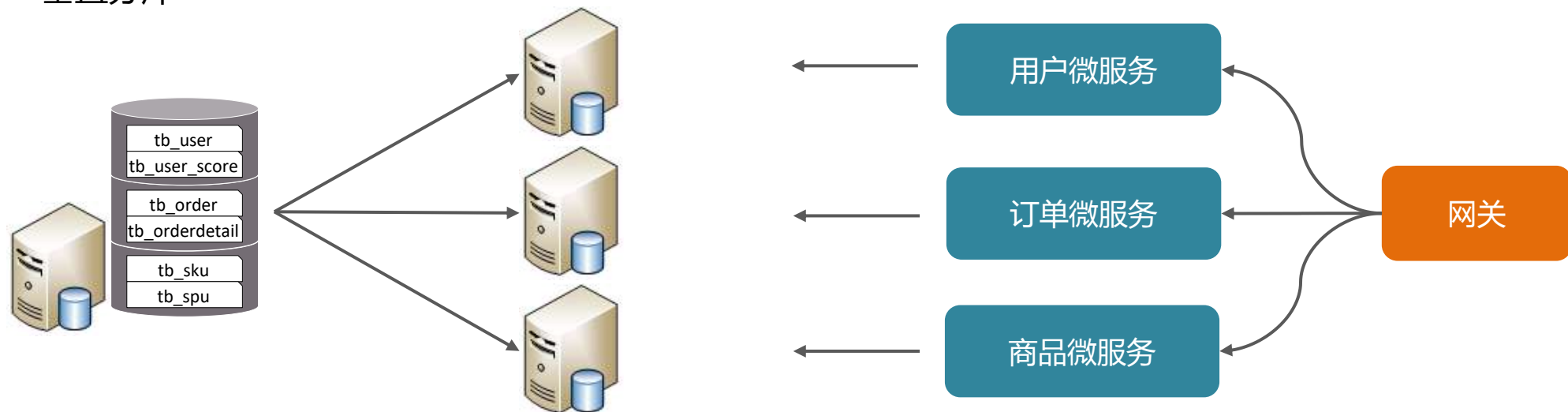
- 1, **前提**，项目业务数据逐渐增多，或业务发展比较迅速 单表的数据量达**1000W**或**20G**以后
- 2, 优化已解决不了性能问题（主从读写分离、查询索引...）
- 3, IO瓶颈（磁盘IO、网络IO）、CPU瓶颈（聚合查询、连接数太多）

拆分策略



垂直拆分

- 垂直分库



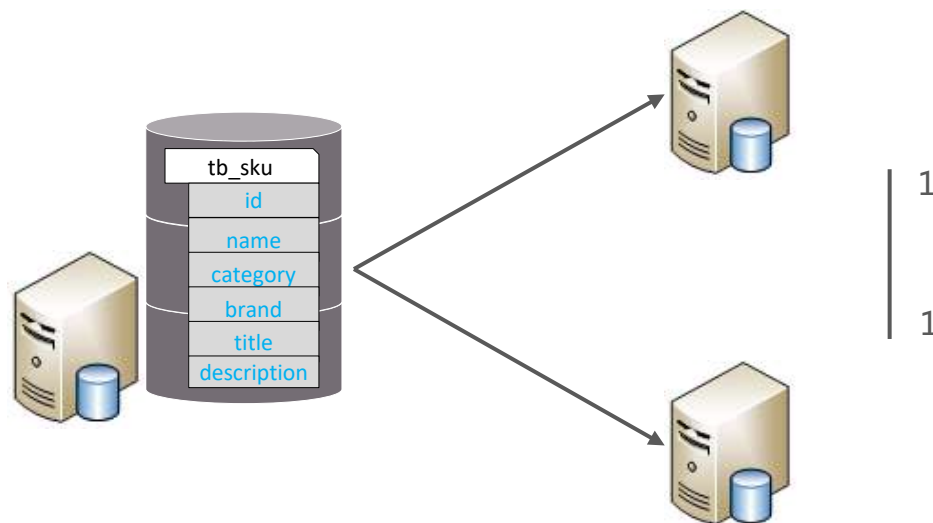
垂直分库：以表为依据，根据业务将不同表拆分到不同库中。

特点：

1. 按业务对数据分级管理、维护、监控、扩展
2. 在高并发下，提高磁盘IO和数据量连接数

垂直拆分

- 垂直分表



拆分规则：

- 把不常用的字段单独放在一张表
- 把text，blob等大字段拆分出来放在附表中

垂直分表：以字段为依据，根据字段属性将不同字段拆分到不同表中。

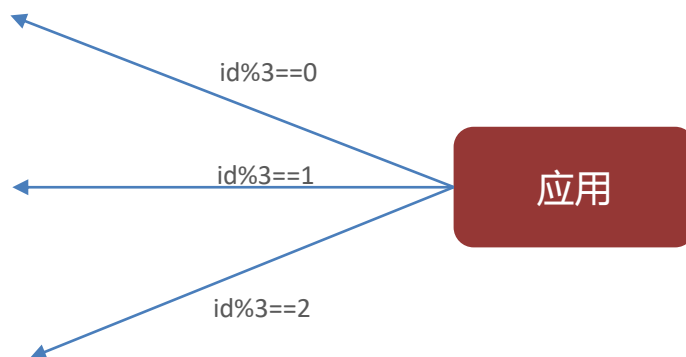
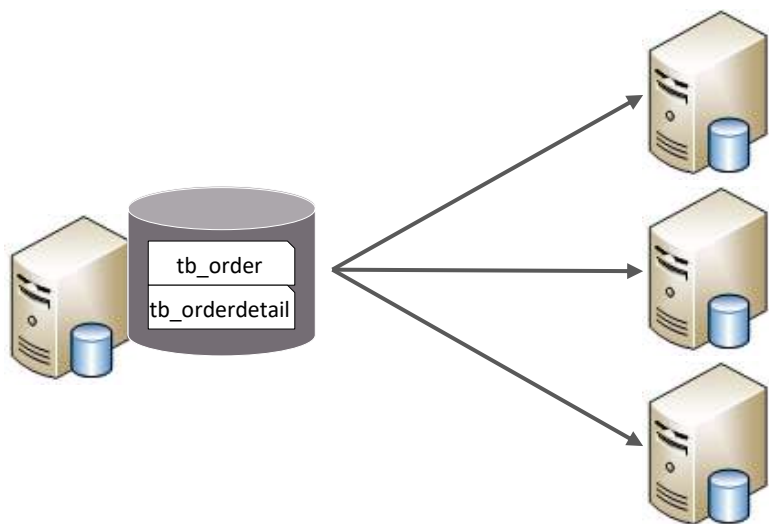
特点：

1，冷热数据分离

2，减少IO过渡争抢，两表互不影响

水平拆分

- 水平分库



水平分库：将一个库的数据拆分到多个库中。

特点：

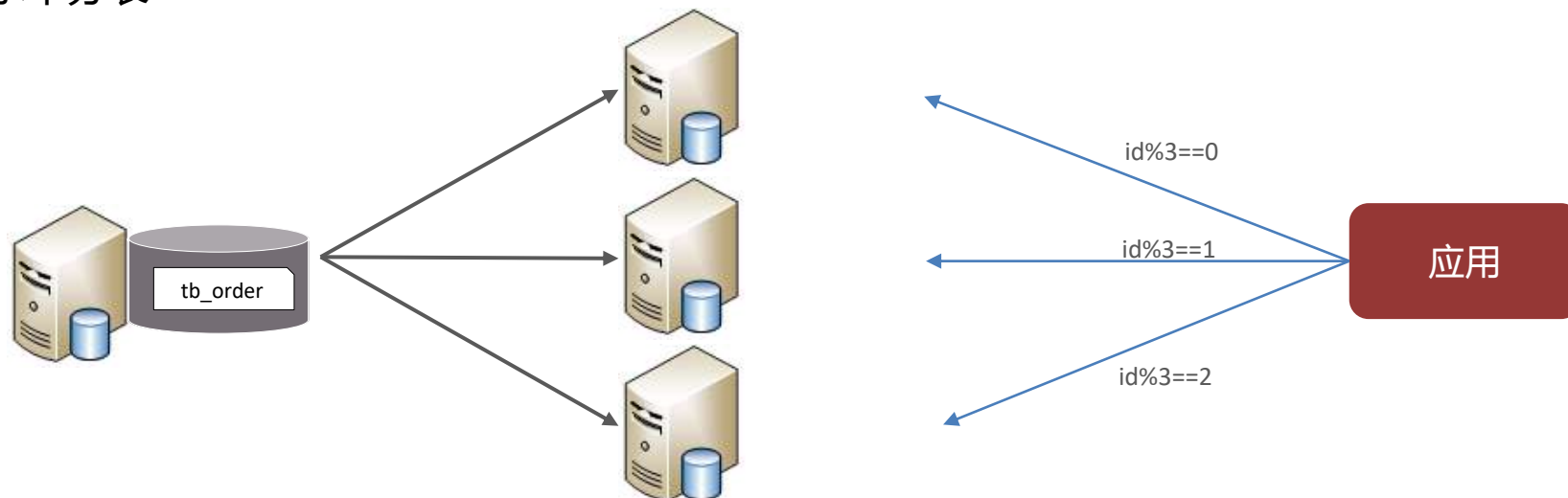
1. 解决了单库大数量，高并发的性能瓶颈问题
2. 提高了系统的稳定性和可用性

路由规则

- 根据id节点取模
- 按id也就是范围路由，节点1(1-100万),节点2(100万-200万)
- ...

水平拆分

- 水平分表



水平分表：将一个表的数据拆分到多个表中(可以在同一个库内)。

特点：

1. 优化单一表数据量过大而产生的性能问题;
2. 避免IO争抢并减少锁表的几率;

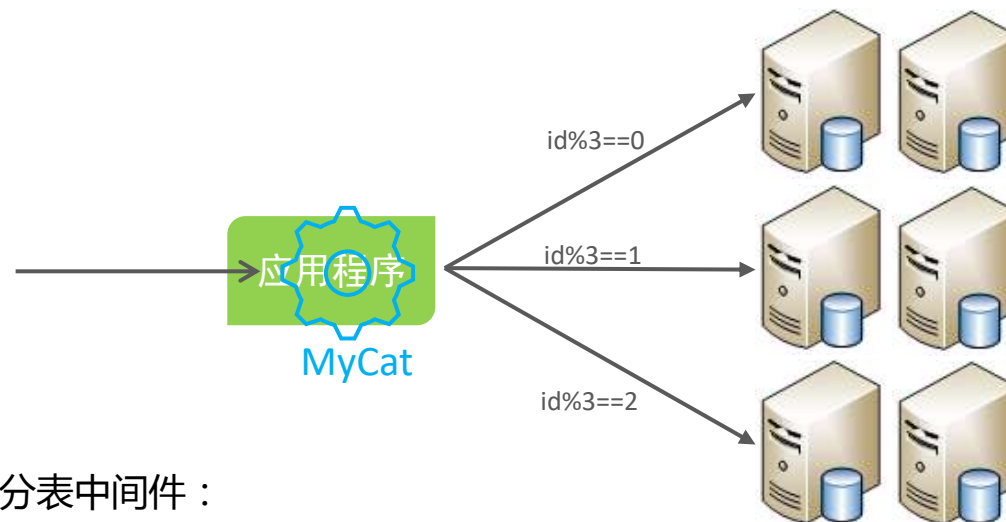
分库分表的策略有哪些

- 新的问题和新的技术



分库之后的问题：

- 分布式事务一致性问题
- 跨节点关联查询
- 跨节点分页、排序函数
- 主键避重



分库分表中间件：

- sharding-sphere
- mycat



你们项目用过分库分表吗

- 业务介绍

- 1, 根据自己简历上的项目, 想一个数据量较大业务 (请求数多或业务累积大)
- 2, 达到了什么样的量级 (单表1000万或超过20G)

- 具体拆分策略

- 1, 水平分库, 将一个库的数据拆分到多个库中, 解决海量数据存储和高并发的问题
- 2, 水平分表, 解决单表存储和性能的问题
- 3, 垂直分库, 根据业务进行拆分, 高并发下提高磁盘IO和网络连接数
- 4, 垂直分表, 冷热数据分离, 多表互不影响

} sharding-sphere、mycat

mysql 搜索

综合 视频 99+ 番剧 0 影视 0 直播 4 专栏 99+ 话题 0 用户 24

综合排序 最多点击 最新发布 最多弹幕 最多收藏 更多筛选

10天精通 MySQL
讲的特别深入的那种!
307.4万 8.4万 29:52:21
黑马程序员 MySQL数据库入门到精通，从mysql安装到mysql高级、...
UP 黑马程序员 · 2022-1-18

每天学建模一小时，提高闲暇收入!
广告 米塔在线

MySQL8.0
<基础+高级>
一套通关 晋升大牛
66.2万 1.1万 29:42:16
黑马程序员MySQL知识精讲+mysql实战案例_零基础mysql数据库入门到...
UP 黑马程序员 · 2021-12-15

黑马讲师：涛哥



传智教育旗下高端IT教育品牌