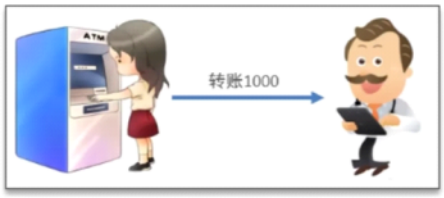
事务

2024年4月5日 12:16

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。

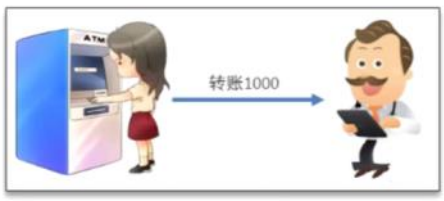


id	name	money
1	张三	2000
2	李四	2000

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

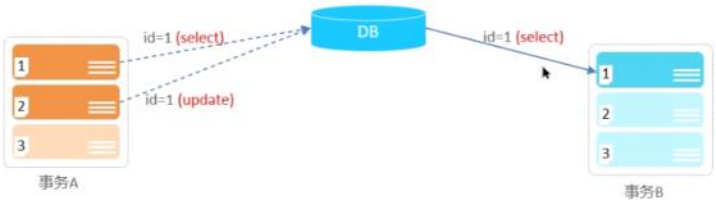
在转账的过程中，持久性体现在事务提交后，要把数据持久化（可以说是落盘操作）

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

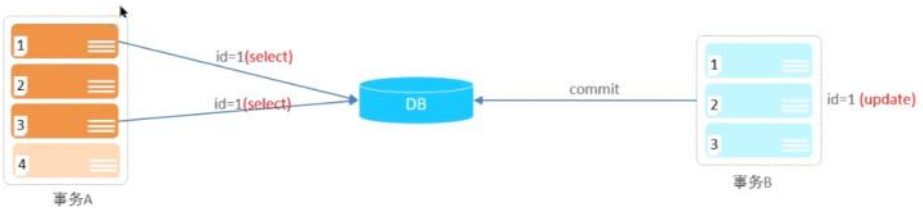
并发事务问题

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"。



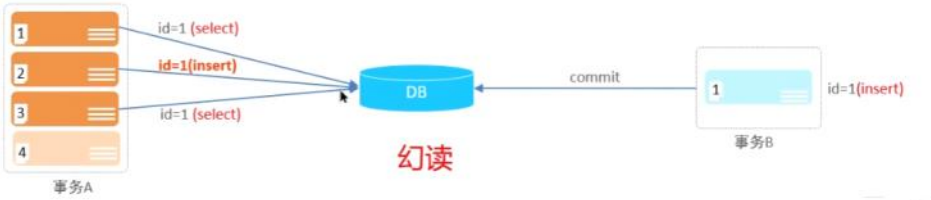
并发事务问题

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：一个事务读到另外一个事务还没有提交的数据。

② 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

③ 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"。

- **隔离级别：**

① READ UNCOMMITTED 未提交读　　脏读、不可重复读、幻读

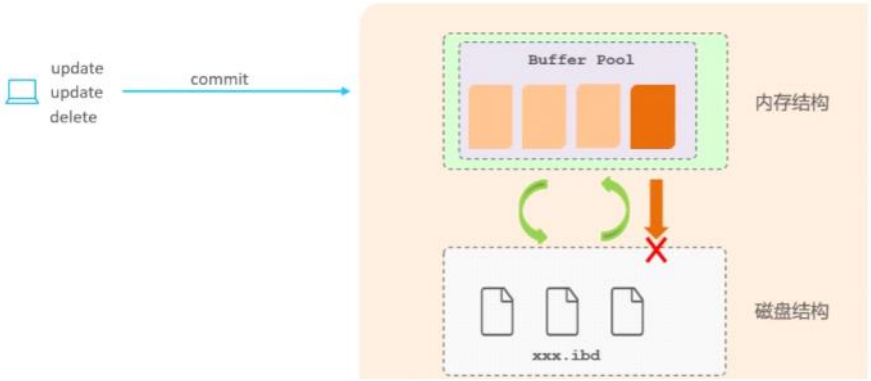
② READ COMMITTED 读已提交　　不可重复读、幻读

③ REPEATABLE READ 可重复读　　幻读

④ SERIALIZABLE　串行化

undo log和redo log的区别

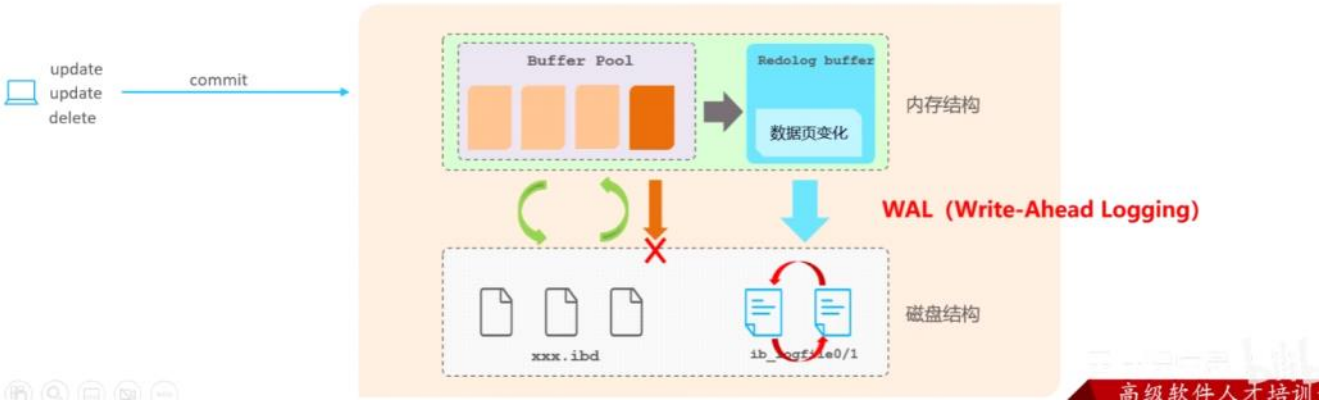
- **缓冲池（buffer pool）**:主内存中的一个区域，里面可以缓存磁盘上经常操作的真实数据，在执行增删改查操作时，先操作缓冲池中的数据（若缓冲池没有数据，则从磁盘加载并缓存），以一定频率刷新到磁盘，从而减少磁盘IO，加快处理速度
- **数据页（page）**:是InnoDB 存储引擎磁盘管理的最小单元，每个页的大小默认为 16KB。页中存储的是行数据



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的内容并进行回滚。

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 Multi-Version Concurrency Control，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。

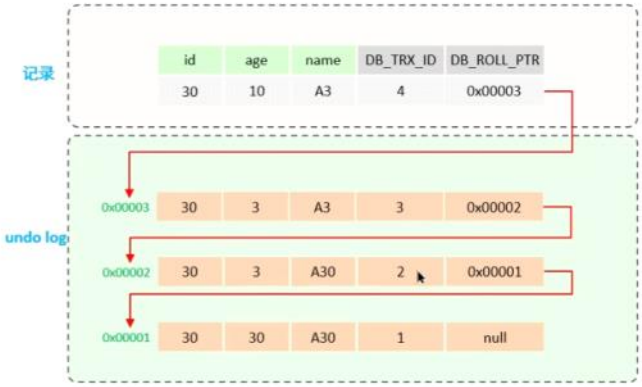
当insert的时候，产生的undo log日志只在回滚时需要，在事务提交后，可被立即删除。

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

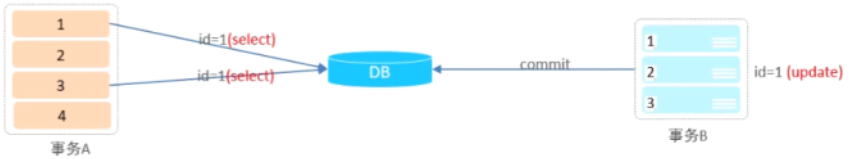
- undo log版本链

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	



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. 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. trx_id < min_trx_id ? 可以访问该版本 → 成立，说明数据已经提交了。
- ③. 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. min_trx_id <= trx_id <= 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。

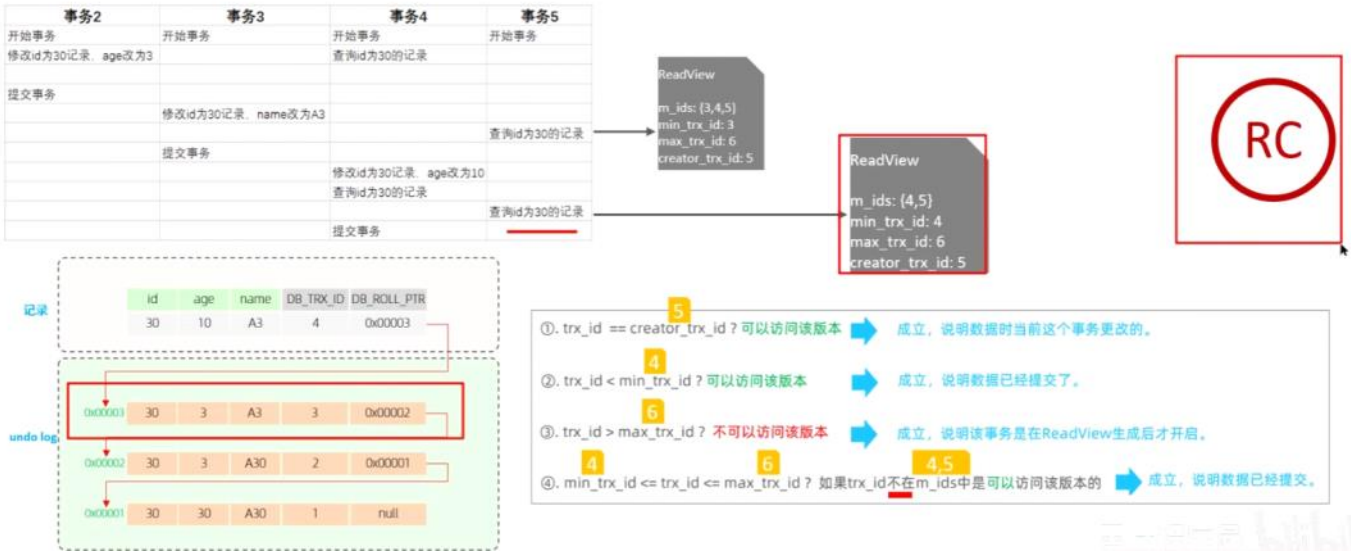
事务2	事务3	事务4	事务5
开始事务	开始事务	开始事务	开始事务
修改id为30记录，age改为3		查询id为30的记录	
提交事务			
	修改id为30记录，name改为A3		
			查询id为30的记录
	提交事务		
		修改id为30记录，age改为10	
		查询id为30的记录	
			查询id为30的记录
		提交事务	

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



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。

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)，记录每一次操作的事务id，是自增的

② roll_pointer(回滚指针)，指向上一个版本的事务版本记录地址

- **undo log**：

① 回滚日志，存储老版本数据

② 版本链：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
  - 根据readView的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

    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

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。

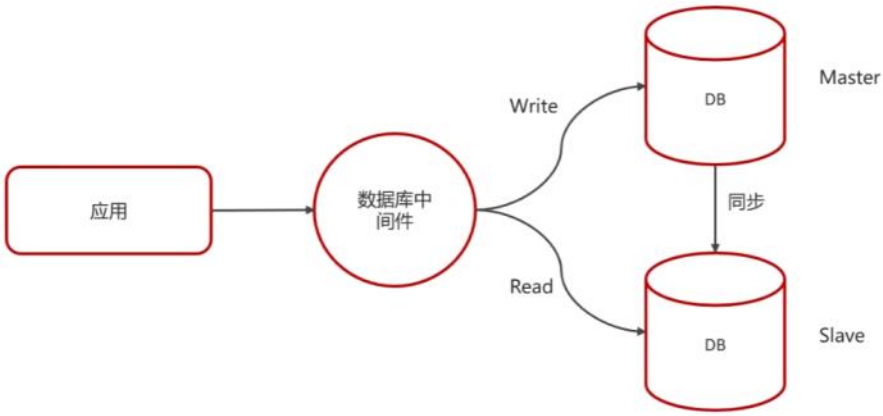
其中mvcc的意思是多版本并发控制。指维护一个数据的多个版本，使得读写操作没有冲突，它的底层实现主要是分为了三个部分，第一个是隐藏字段，第二个是undo log日志，第三个是readView读视图

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

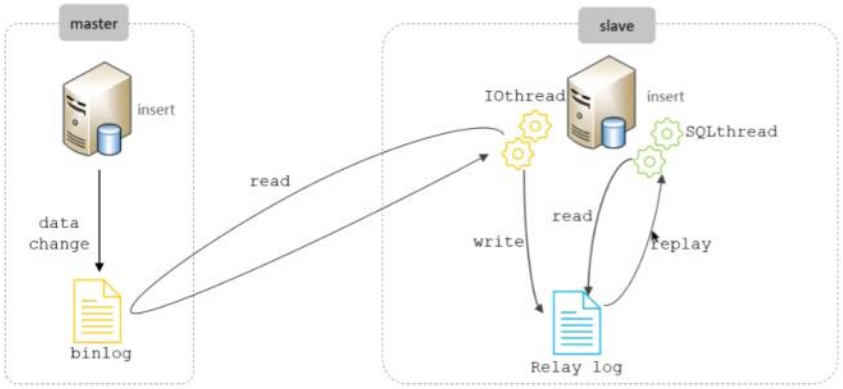
MySQL主从同步原理



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？

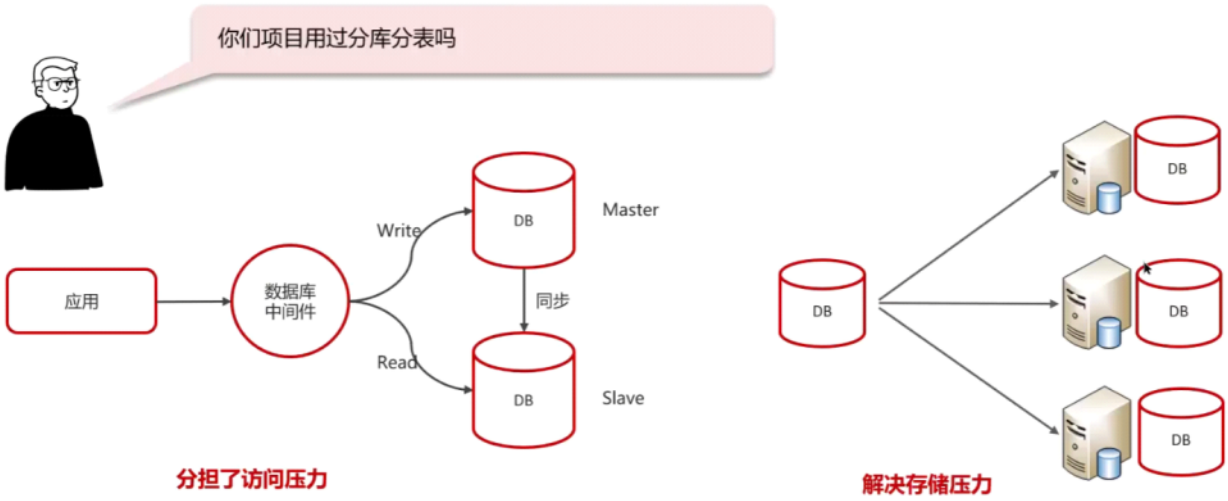
候选人：

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。



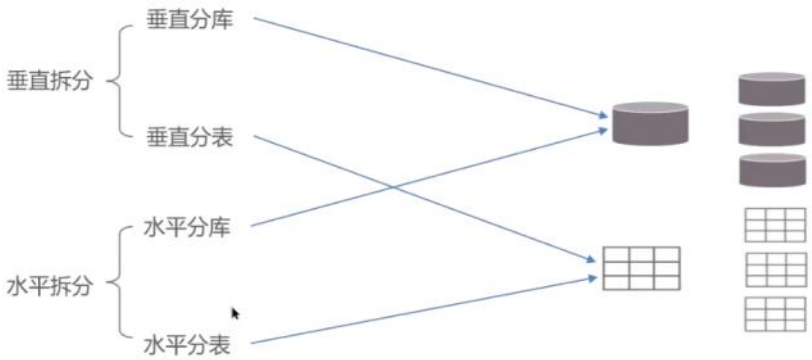
分库分表的时机：

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...）

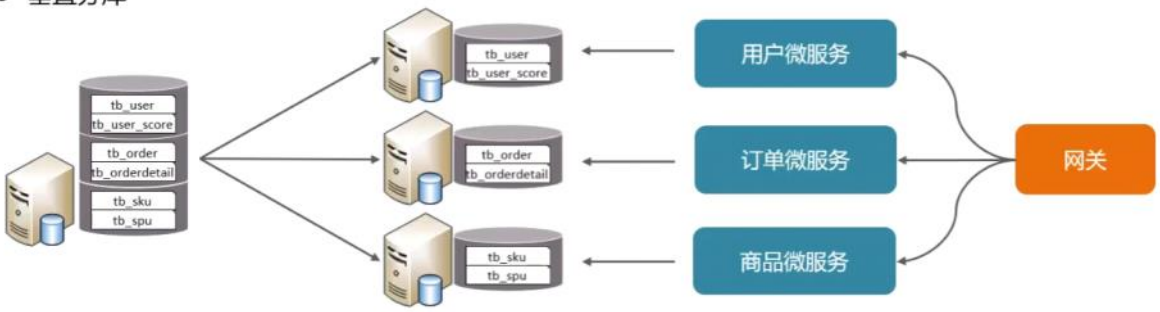
3，IO瓶颈（磁盘IO、网络IO）、CPU瓶颈（聚合查询、连接数太多）

拆分策略



垂直拆分

- 垂直分库



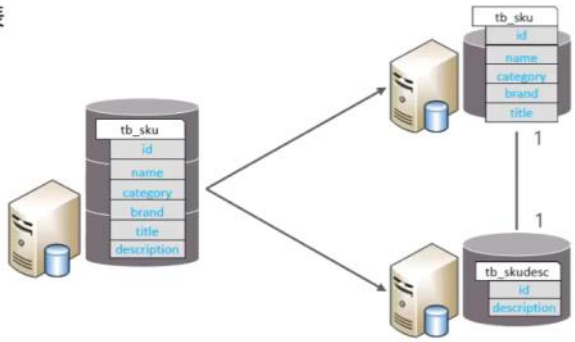
垂直分库：以表为依据，根据业务将不同表拆分到不同库中。

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。

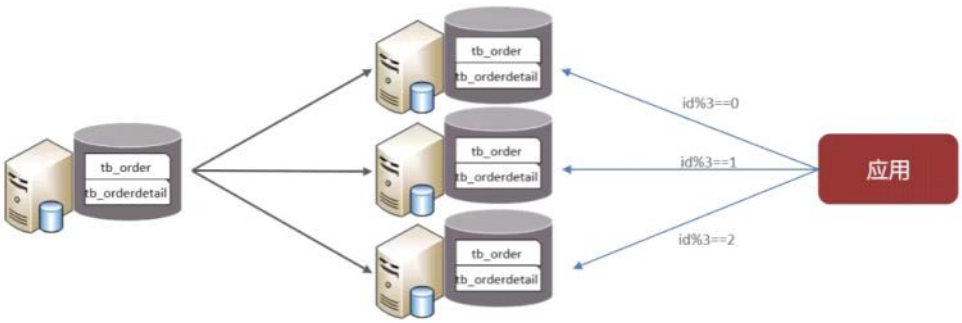
特点：

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：

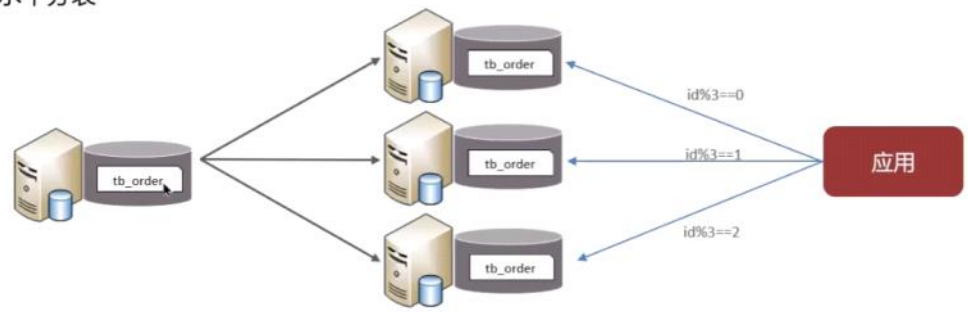
1. 解决了单库大数量，高并发的性能瓶颈问题
2. 提高了系统的稳定性和可用性

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



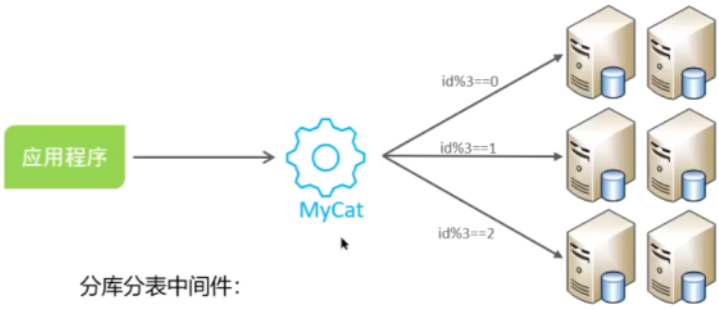
水平分表：将一个表的数据拆分到多个表中(可以在同一个库内)。

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

} sharding-sphere、mycat

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