**1、一二三范式（一对一，数据大，拆分表，外键唯一）**

（1）第一范式，数据库中的每一列都是不可分割的，如果实体中的某个属性有多个值时，必须进行拆分。1NF是关系模式应具备的最起码的条件，只要是关系型数据库，一定满足第一范式。

（2）第二范式，满足第二范式（2NF）必须先满足第一范式（1NF）。第二范式（2NF）要求数据库表中的每个实例或记录必须可以被唯一地区分。选取一个能区分每个实体的属性或属性组，作为实体的唯一标识，即主键，并且非主键部分要完全依赖所有主键（复合主键）。多对多，三张表，关系表两个外键

（3）第三范式，消除非主属性对主键的传递依赖。如果某一属性依赖于其他非主键数属性，而其他非主键属性又依赖于主键，则这个属性就是传递依赖于主属性。一对多，两张表，多的表加外键。

数据库设计三范式是理论上的，在实践中可能会有偏差看，最终的目的是满足客户的需求，有的时候会拿冗余换执行速度，因为表与表之间的连接越多，效率越低，存在冗余也会使得开发人员编写难度降低

**2、什么是事务**

事务是数据库区别于文件系统的重要特性之一，事务可以一条非常简单的SQL语句组成，也可以由一组复杂的SQL语句组成，事务是访问并更新数据库中各种数据项的一个程序执行单元。在事务中的操作，**要么都做修改，要么都不做**，这就是事务的主要目的。

**3、事务的特性有哪些？分别是怎么保证的？**

（1）原子性：事务是最小的执行单位，不允许分割。事务的原子性保证动作要么全部完成，要么完全不起作用；

（2）一致性：执行事务前后，数据保持一致，多个事务对同一个数据读取的结果是相同的；

（3）隔离性：并发访问数据库时，一个用户的事务不被其他事务所干扰，各并发事务之间数据库是独立的；

（4）持久性：一个事务被提交之后。它对数据库中数据的改变是持久的，即使数据库发生故障也不应该对其有任何影响。

原子性、一致性和持久性是通过数据库的redo log和undo log来完成。redo log称为重做日志，用来保证事务的原子性和持久性。undo log用来保证事务的一致性。而隔离性是通过锁实现的。

**4、三种日志的区别与作用**

redo log：恢复提交事务修改的页操作；通常是物理日志，记录的是页的物理修改操作。

uodo log：回滚记录到某个特定版本；通常是逻辑日志，根据每行记录进行记录。

bin log：用来进行Point-In-Time(PIT)的恢复及主从复制环境的建立。

5、**binlog和redolog的区别？**

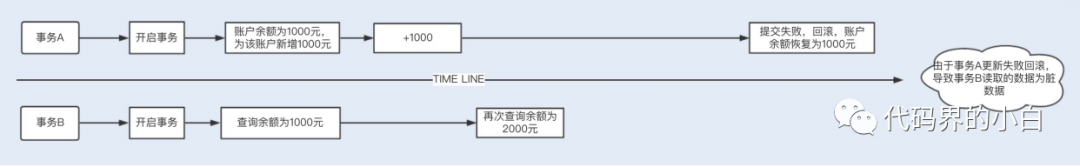
（1）重做日志是在InnoDB存储引擎层产生的，而二进制日志是在MySQL数据库上层产生的，二进制日志不仅仅针对InnoDB存储引擎，任何存储引擎都会产生二进制日志。

（2）两种日志的记录内容形式不同。二进制日志是一种逻辑日志，记录的是SQL语句；而InnoDB存储引擎层面的重做日志是物理格式日志，记录的是对于每个页的修改。

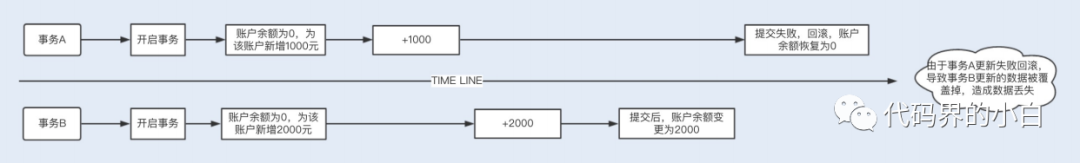
（3）写入磁盘的时间不同，二进制日志只在事务提交完成后进行一次写入，而redo log在事务进行中不断的写入。

**6、并发事务带来哪些问题？**

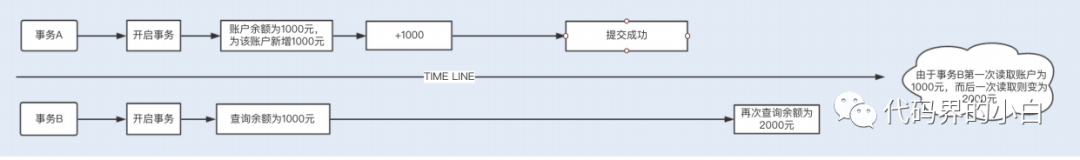
（1）**脏读（Dirty read）:** 当一个事务正在访问数据并且对数据进行了修改，而这种修改还没有提交到数据库中，这时另外一个事务也访问了这个数据，然后使用了这个数据。



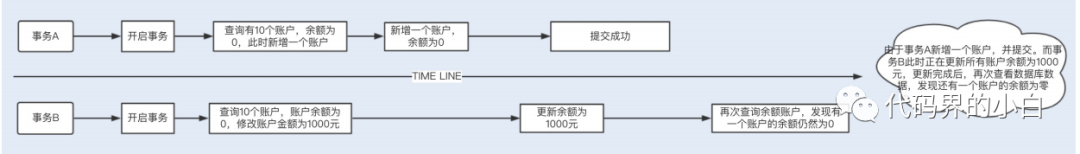
（2）**丢失修改（Lost to modify）:** 指在一个事务读取一个数据时，另外一个事务也访问了该数据，那么在第一个事务中修改了这个数据后，第二个事务也修改了这个数据。这样第一个事务内的修改结果就被丢失，因此称为丢失修改。



（3）不可重复读（Unrepeatableread）: 指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。



（4）**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。



**不可重复度和幻读区别：**不可重复读的重点是修改，幻读的重点在于新增或者删除。

**7、事务隔离级别（有哪些？MySQL的默认级别？**

（1）**未提交读**（Read Uncommitted）：最低的隔离级别，允许读取尚未提交的数据变更，可能会导致脏读、幻读或不可重复读。

（2）**已提交读**（Read Committed）：允许读取并发事务已经提交的数据，可以阻止脏读，但是幻读或不可重复读仍有可能发生。

（3）**可重复读**（Repeatable Read）：对同一字段的多次读取结果都是一致的，除非数据是被本身事务自己所修改，可以阻止脏读和不可重复读，但幻读仍有可能发生。

（4）**可序列化**（Serializable）：所有的事务依次逐个执行，这样事务之间就完全不可能产生干扰，也就是说，该级别可以防止脏读、不可重复读以及幻读。

MySQL InnoDB存储引擎的默认支持的隔离级别是**REPEATABLE-READ（可重复读）**。

**8、常见的索引有哪些？**

（1）单一索引：由一个字段组成的索引

（2）唯一索引：与普通索引类似，不同的就是：MySQL数据库索引列的值必须唯一，但允许有空值

（3）主键索引：主键上添加索引，它是一种特殊的唯一索引，不允许有空值。一般在建表的同时创建主键索引

（4）复合索引(联合索引)：多个字段共同组成一个索引

（5）全文索引：主要用来查找文本中的关键字，而不是直接与索引中的值相比较

**9、为什么使用索引？**

（1）通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性

（2）可以大大加快数据的检索速度（主要原因）

（3）可以帮助服务器避免临时排序和临时表

（4）将随机IO变为顺序IO

（5）可以加速表与表之间的连接，特别是在实现数据的参考完整性方面特别有意义。

**10、索引这么好，为什么不对表中的每一列创建一个索引呢？**

（1）当对表中的数据进行增删改的时候，索引也要动态的维护，降低了数据的维护速度

（2）索引需要占用物理空间，如果建立聚簇索引，需要很大空间

（3）创建索引和维护索引要耗费时间，会随着数据量的增加而增加

**11、索引失效**

（1）字段模糊查询采用%开头，无法找到第一个字符

（2）or操作符，or要求两个字段都要有索引，一个没有另一个也会生效

（3）使用复合索引时，左侧的字段有索引，右侧字段没有索引

（4）索引字段本身参与运算如where sal + 1 == 800，只能是sal == 799

（5）在where当中对索引列使用了函数

**11、B+树和B树的区别？**

（1）B树的每个结点都存储了key和data，B+树的data存储在叶子节点上。  
节点不存储data，这样一个节点就可以存储更多的key。可以使得树更矮，所以IO操作次数更少。

（2）树的所有叶结点构成一个有序链表，可以按照关键码排序的次序遍历全部记录，由于数据顺序排列并且相连，所以便于区间查找和搜索。而B树则需要进行每一层的递归遍历。相邻的元素可能在内存中不相邻，所以缓存命中性没有B+树好。

**12、MySQL索引主要使用的数据结构**

（1）哈希索引：底层是哈希表，当需求为查询的时候，选择哈希索引，查询性能最快，其余大部分场景，选择B Tree索引

（2）BTree索引：主要采用B树中的B+树，但是对于主要的两种存储引擎（MyISAM和InnoDB）的实现方式是不同的

**13、为什么MySQL数据库使用B+树而不使用B树？**

（1）B+树的磁盘读写代价更低：B+树的内部节点并没有指向关键字具体信息的指针，因此其内部节点相对B树更小，如果把所有同一内部节点的关键字存放在同一盘块中，那么盘块所能容纳的关键字数量也越多，一次性读入内存的需要查找的关键字也就越多，相对IO读写次数就降低了。

（2）B+树的查询效率更加稳定：由于非终结点并不是最终指向文件内容的结点，而只是叶子结点中关键字的索引。所以任何关键字的查找必须走一条从根结点到叶子结点的路。所有关键字查询的路径长度相同，导致每一个数据的查询效率相当。

（3）B+树更便于遍历：由于B+树的数据都存储在叶子结点中，分支结点均为索引，方便扫库，只需要扫一遍叶子结点即可，但是B树因为其分支结点同样存储着数据，我们要找到具体的数据，需要进行一次中序遍历按序来扫，所以B+树更加适合在区间查询的情况。

（4）B+树更适合基于范围的查询：B树在提高了IO性能的同时并没有解决元素遍历的效率低下的问题，正是为了解决这个问题，B+树应运而生。B+树只需要去遍历叶子节点就可以实现整棵树的遍历。而且在数据库中基于范围的查询是非常频繁的，而B树不支持这样的操作或者说效率太低。

**14、MyISAM和InnoDB实现B树索引方式的区别是什么？**

（1）MyISAM，B+Tree叶节点的data域存放的是数据记录的地址到sesx，在索引检索的时候，首先按照B+Tree搜索算法搜索索引，如果指定的key存在，则取出其data域的值，然后以data域的值为地址读取相应的数据记录，这被称为“非聚簇索引”

（2）InnoDB，其数据文件本身就是索引文件，相比MyISAM，索引文件和数据文件是分离的，其表数据文件本身就是按B+Tree组织的一个索引结构，树的节点data域保存了完整的数据记录，这个索引的key是数据表的主键，因此InnoDB表数据文件本身就是主索引，这被称为“聚簇索引”或者聚集索引，而其余的索引都作为辅助索引，辅助索引的data域存储相应记录主键的值而不是地址，这也是和MyISAM不同的地方。

**15、Hash索引和B+树索引区别是什么？**

（1）B+树可以进行范围查询，Hash索引不能

（2）B+树支持联合索引的最左侧原则，Hash索引不支持

（3）B+树支持排序，Hash索引不支持

（4）Hash在等值查询上比B+树效率更高

（5）B+树可以使用like进行模糊查询，而Hash索引不能

**16、什么是最左前缀原则？**

MySQL中的索引可以以一定顺序引用多列，这种索引叫作联合索引。在组合索引树中，最底层的叶子节点按照第一列a列从左到右递增排列，但是b列和c列是无序的，b列只有在a列值相等的情况下小范围内递增有序，而c列只能在a，b两列相等的情况下小范围内递增有序。

**最左前缀匹配**：使用组合索引查询时，mysql会一直向右匹配直至遇到范围查询(>、<、between、like)就停止匹配。

**20、什么是锁，锁的作用是什么？**

锁是数据库系统区别文件系统的一个关键特性，锁机制用于管理对共享资源的并发访问，保持数据的完整性和一致性。

**21、数据库中有哪些锁？lock和latch的区别？**

回答：数据库中有表锁和行锁等

**lock锁**：锁的对象是事务，用于锁定数据库中的对象，如表、页、行等，并且lock锁一般在commit或rollback后释放，有死锁机制。

**latch锁**：一般称为轻量级锁，要求锁定的时间必须非常短，在InnoDB中又可以分为mutex(互斥量)和rwlock(读写锁)。目的是用来保证并发线程操作临界资源的正确性，并且通常没有死锁检测的机制。

**22、InnoDB存储引擎中的锁有哪些类型？**

有共享锁、排他锁、意向锁、一致性非锁定读和一致性锁定读。

行级锁：

共享锁(S Lock)：运行事务读一行数据。

排他锁(X Lock)：允许事务删除或更新一行数据。

表级锁：

意向锁：分为意向共享锁(IS Lock)和意向排他锁(IX Lock)

意向共享锁(IS Lock)：事务想要获得一张表中某几行的共享锁。

意向排他锁(IX Lock)：事务想要获得一张表中某几行的排他锁。

一致性非锁定读：

指InnoDB存储引擎通过**多版本控制**的方式来读取当前执行时间数据库中行的数据。如果读取的行正在执行DELETE或UPDATE操作，这时读取操作不会因此等待行上锁的释放，相反的，InnoDB存储引擎会读取一个快照数据。

一致性锁定读：

InnoDB存储引擎对于SELECT语句支持两种一致性锁定读的操作：select ... for update和select ... lock in share mode

**23、什么是一致性非锁定读（MVCC）？**

最早的数据库系统，只有读读之间可以并发，读写，写读，写写都要阻塞。引入多版本之后，只有写写之间相互阻塞，其他三种操作都可以并行，这样大幅度提高了InnoDB的并发度。

在并发读写数据库时，可以做到在读操作时不用阻塞写操作，写操作也不用阻塞读操作，提高了数据库并发读写的性能同时还可以解决脏读，幻读，不可重复读等事务隔离问题，但不能解决更新丢失问题 MVCC只在读取已提交和可重复 读两种隔离级别下有作用。

实现方式有乐观锁和悲观锁。

**24、MVCC能解决什么问题？**

数据库并发有三种场景，分别是：

（1）读-读：不存在任何问题，不需要并发控制

（2）读-写：有线程安全问题，可能会造成事务隔离性问题，可能遇到脏读，幻读，不可重复读问题

（3）写-写：有线程安全问题，可能存在更新丢失问题

**17、乐观锁和悲观锁的区别？**

**悲观锁**：总是假设最坏的情况，每次去取数据的时候都认为别人会修改，所以每次在取数据的时候都会上锁，这样别人想拿这个数据就会阻塞直到它拿到锁（共享资源每次只给一个线程使用，其它线程阻塞，用完后再把资源转让给其它线程）。

**适用于写比较多的场景**。

**乐观锁**：总是假设最好的情况，每次去取数据的时候都认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有去更新这个数据，可以使用版本号机制和CAS算法实现。**适用于读较多写较少的场景**。

**18、乐观锁的实现方式**

（1）版本号机制：一般是在数据表中加上一个数据版本号version字段，表示数据被修改的次数，当数据被修改时，version值会加一。当线程A要更新数据值时，在读取数据的同时也会读取version值，在提交更新时，若刚才读取到的version值为当前数据库中的version值相等时才更新，否则重试更新操作，直到更新成功。

（2）CAS算法：涉及三个操作数，需要读写的内存值V、进行比较的值A和拟写入的新值B，当且仅当V符合A时，CAS通过原子方式用新值B更新V，否则不会执行任何操作，一般是个自旋锁。

**19、CAS带来的问题**

（1）ABA问题：如果一个变量V初次读取的时候是A值，并且在准备赋值的时候检查到它仍然是A值，那我们就能说明它的值没有被其他线程修改过了吗？很明显是不能的，因为在这段时间它的值可能被改为其他值，然后又改回A，那CAS操作就会误认为它从来没有被修改过。

（2）自旋CAS（也就是不成功就一直循环执行直到成功）如果长时间不成功，会给CPU带来非常大的执行开销。

（3）CAS只对单个共享变量有效，当操作涉及跨多个共享变量时CAS无效。