1. 事务隔离级别
2. 概述

由于MySQL是一个C/S架构的软件，对于同一个服务器来说，可以有多个客户端与之连接，每个客户端与服务器连接后就形成一个会话，每个客户端可以在自己的会话中发送请求，一个请求语句可能是某个事务的一部分，服务器可以同时处理来自多个客户端的多个事务。由于一个事务对应着现实世界的一次状态转换，事务执行之后必须保证数据符合现实世界的规则，这就是事务的一致性。

当有多个事务要同时访问同一个数据时，我们需要通过某种手段来让这些事务按照顺序一个个地单独执行，或者最终执行结果和单独执行一样，也就是各个事务是隔离执行的，互不干涉，这就是事务的隔离性。最简单的方式是按照顺序执行，直接让所有事务在单线程中执行，这种方式称为串行执行，效率太低；也可以限制在某个事务访问某个数据时，对其他试图访问相同数据的事务进行限制，让它们排队，这种方式称为可串行化执行。

1. 事务并发的一致性问题

脏写和脏读一起记忆，一个是对未提交事务修改过的数据进行写，另一个是进行读；不可重复读和幻读一起记忆，都是对未提交事务读取的数据进行修改。

1. 脏写

如果一个事务修改了另一个未提交事务修改过的数据，则称为脏写现象。如何理解记忆？脏代表着一个未提交事务修改过的数据，写代表着另一个事务对该数据进行修改。一个事务修改了这个数据，另一个数据又修改了这个数据，那么两个事务最后的结果都是不一样的。

1. 脏读

如果一个事务读到了另一个未提交事务修改过的数据，则称为脏读现象。严格的解释是T1事务首先修改了数据的值，然后T2事务又读取了未提交事务T1对于x修改后的值，之后T1中止T2提交，那么T2读到了一个不存在的值。

1. 不可重复读

如果一个事务修改了另一个未提交事务读取的数据，意味着发生了不可重复读。也就是说一个事务在另一个事务修改前后对同一个数据读到了不同的结果。

1. 幻读

如果一个事务先根据某些搜索条件查询出了一些记录，在该事务未提交时，另一个事务写入了一些符合那些搜索条件的记录（INSERT, UPDATE, DELETE），就意味着发生了幻读操作。因为前一个事务在再次搜素时，结果与第一次搜索是不一致的。虽然说这个概念看起来和不可重复读是差不多的，但是幻读是针对搜索记录的，不可重复读针对的是读取记录。

1. sql标准中的四种隔离级别

前面介绍的四种一致性问题的严重性排序如下：

脏写 > 脏读 > 不可重复读 > 幻读

对于串行执行这种隔离性最高的方式，带来的结果就是性能最差，我们可以放弃一些隔离性以换取一部分的性能，这就是设立隔离级别的初衷。隔离级别越高，问题越少，性能越差；隔离级别越低，问题越多，性能越好。

1. 未提交读

最低的隔离级别，可能发生最多的问题，如脏读，不可重复读，幻读。从字面 意思上来理解，能够读取未提交的事务的数据，就是脏读，连脏读都解决不了，更别说后面严重程度比较轻的问题了。

1. 已提交读

解决了脏读问题，可能发生不可重复读，幻读。从字面意思上来看，能够读取已提交事务的数据，说明脏读问题解决了，还有后续的问题。

1. 可重复读

解决了脏读问题，可重复读问题，还可能发生幻读问题。

1. 可串行化

解决了脏读问题，可重复读问题，幻读问题。使用可串行化的方式执行，隔离性最强，性能够最低。

为什么脏写问题每一个隔离级别都没有提到呢？因为脏写对于一致性影响过于严重，无论哪种隔离级别都不允许脏写的发生。

1. Mysql中支持的四种隔离级别

不同的数据库厂商对于SQL标准中的四种隔离级别的支持都不一样，mysql虽然支持四种级别，但是对于可重复读级别，可以很大程度上禁止幻读现象发生，默认级别是可重复读。

1. 设置事务的隔离级别

set global transaction isolation level serializable;

在set关键字后面放置 global关键字，在全局范围内产生影响，只对执行完该语句之后的会话起作用，对于当前已经存在的会话是无效的。

set session transaction isolation level serializable;

在set关键字后面放置 session关键字，在会话范围内有效，对当前会话的所有后续事务有效。

如果两个关键字都不使用，就只对当前会话的下一个即将开启的事务有效，后续事务恢复到之前的隔离级别。

1. 修改服务器默认隔离级别

修改启动项 transaction-isolation，或者修改系统变量transaction\_isolation，对于这个系统变量，有三个作用范围，GLOBAL, SESSION和仅作用于下一个事务。

SET GLOBAL transaction\_isolation=…

SET SESSION var\_name = xxx

SET var\_name = xxx

1. MVCC
2. 版本链

对于使用InnoDB存储引擎的表来说，它的聚簇索引记录都包含两个必要的隐藏列，trx\_id属性表示对该条记录进行修改的事务id，roll\_pointer指向上一次对该记录进行修改的undo日志，这样该记录的所有修改undo日志都可以串成一个链表，称为版本链，版本链的头结点就是当前记录的最新值，可以通过它找到该记录修改前的信息。

另外，每个版本节点还保存了生成该版本时对应的事务id，我们会利用这个记录的版本链来控制并发事务访问相同记录的行为，这种机制称为多版本并发控制（MVCC）。

1. ReadView

对于不同的隔离级别，事务可以读取到记录的不同版本，核心问题是如何判断版本链中的哪个版本才是当前事务可见的。InnoDB中使用ReadView(一致性视图)来解决这个问题。视图中包含四个重要内容：

1. m\_ids

在生成ReadView时，当前系统中活跃的读写事务的事务id列表，即未提交的事务。

1. min\_trx\_id

在生成ReadView时，当前系统中活跃的读写事务中最小的事务id，也就是m\_ids的最小值。

1. max\_trx\_id

在生成ReadView时，系统应该分配给下一个事务的事务id。

1. creator\_trx\_id

生成该ReadView的事务的事务id。

1. 根据ReadView判断当前记录的版本可见性
2. 如果被访问版本的trx\_id属性值与ReadView中的creator\_trx\_id相同，说明当前事务访问的是它自己修改过的记录，所有该版本记录可以被当前事务访问。
3. 如果被访问版本的trx\_id属性小于ReadView中的min\_trx\_id属性，表名生成该版本的事务在当前事务前已经提交了，所以该版本可以被当前事务访问。
4. 如果被访问版本的trx\_id属性值大于或等于ReadView中的max\_trx\_id值，表名生成该版本的事务在当前事务生成ReadView之后才开启，所以该版本不可以被当前事务访问。
5. 如果被访问版本的trx\_id属性值在ReadView的min\_trx\_id和max\_trx\_id之间，则需要判断trx\_id属性是否在m\_ids列表中，如果在，说明生成该版本的事务该活跃，没有提交，因此该版本不能被访问，否则可以访问。

如果某个版本的数据对当前事务不可见，就顺着版本链找下一个版本的数据，并继续执行上面的步骤来判断记录的可见性，如果最后一个版本页不可见，则查询结果不包含该记录。

1. 生成ReadView的时机

已提交读隔离级别在每次读取数据前都会生成一个ReadView，即会不停地更新ReadView，它会根据当前事务的提交情况来更新ReadView中的事务活跃列表与其他参数，这会导致在当前事务中每次读取的结果可能都会不一样，也就是不可重复读问题。

而对于可重复读隔离级别，只会在第一次执行查询语句时生成一个ReadView，每次都是复用第一次的ReadView，这样其中的事务活跃列表等参数都不会变化，那么当前事务从版本链中读取的结果就都是一致的，解决了不可重复读问题。

1. 二级索引和MVCC

前面都是根据主键在聚簇索引记录中查找，如果查询语句使用二级索引来查询，如何判断可见性呢？

步骤如下：

1. 二级索引页面的Page Header中有一个属性，每当对该页面中的记录执行增删改操作时，如果执行该操作的事务的事务id大于该属性的值，就会更新该值为事务id，也就是说该属性代表着修改该二级索引页面的最大事务id是什么。当某个SELECT语句访问某个二级索引记录时，首先查看当前的ReadView的min\_trx\_id是否大于该页面的该属性值，如果是，说明该属性对应的事务已经提交，该页面的所有记录都对该ReadView可见，否则就得执行回表，再判断可见性。
2. 利用主键回表之后，得到对应的聚簇索引再按照之前的方式找到可见的第一个版本，判断该版本中相应的二级索引列的值是否与利用该二级索引列查询时的值相同。如果是则将该记录发送给客户端，否则跳过。
3. 关于purge

之前说insert undo日志在事务提交之后就可以释放掉了，而update undo日志还需要支持MVCC不能立即删除，一个事务写的一组undo日志中都有一个History链表节点，当一个事务提交之后，就会把这个事务执行过程中产生的一组update undo日志插入到history链表的头部。这些update undo日志的空间应该在什么时候被释放掉呢？

我们应该在合适的时候将update undo日志以及仅仅标记为删除的记录彻底删除掉，这个操作称为 purge。这些日志是为了MVCC而保存的，那么只要系统中最早产生的ReadView不再访问它们了，它们的使命也就结束了。只要我们保证在生成ReadView时某个事务已经提交，那么该ReadView肯定就不需要访问该事务运行过程中产生的undo日志了，因为该ReadView已经可以查看到该事务所改动记录的最新版本了。

在一个事务提交时，会为这个事务生成一个名为事务no的值，表示事务提交的顺序，history链表也是按照事务提交的顺序来排列各组undo日志的，在生成一个ReadView也会包含比当前系统中最大的事务no还大1的值。系统中所有的ReadView按照创造时间连成链表，当执行purge操作时，就把系统中最早生成的ReadView取出来，如果不存在ReadView就新建一个，再从History链表中取出事务no值较小的各组undo日志，如果一组undo日志的事务no值小于最早的ReadView，则释放他们的空间。

这里要注意，对于可重复读隔离级别，由于会一直复用最初的ReadView，加入这个事务运行了很久一直没有提交，那么最早生成的ReadView会一直不释放，系统中的update undo日志会越来越多，浪费系统性能。