1. 考察触发器关联更新的基本知识，以及通过非法语句的执行退出触发器的小技巧。对于交易金额不超过 500 的条件，有的同学很全面地想到了±500 的限制。

2.考察触发器删/改记录，自动更新其他表的基本知识，在“SET balance = balance - OLD.amount + NEW.amount”，有的同学会把两个步骤放到两个触发器中，分别在 before 和 after 中执行，会对性能造成影响，还是尽量减少触发器为好。

3.考察查询触发器信息的知识，SELECT * FROM `information_schema`.`TRIGGERS`;和 show triggers 都是正确的。

4.考察的是手动提交事务的知识，通过 begin/start transaction 和 commit 的组合，显式地提交事务。有的同学会漏了 start transaction

5.考察对于事务和回滚的理解，大家回答的都很准确。当 autocommit=OFF 时，仅当 commit 或者 rollback 之后一次事务才能生效，rollback 后，事务回滚，对两张表的改变没有生效。

6.考察的是数据库的设计和算法知识迁移的能力。题目的本意是希望同学建立尽可能少的表，通过增加少数字段的形式，将节点间的父子关系存储起来。有一小部分同学，选择了每一个层次开一张数据表的做法，这样在需要查询子节点和后代节点的场景下，就需要在多张表上反复的切换，也不便于层级的扩展。另一部分同学通过增加 parent_id 字段的方式存储父节点，优点是查找父节点很方便，缺点是查找子节点需要递归遍历。第三种做法是通过字符串存储父系路径的所有节点 ID，通过字符串解析的方式遍历一次即可知道其位置，性能高，算法直观，缺点是需要做字符串的解析。最后一种做法是通过设置左右

值，保存 DFS 先序遍历的结果，通过比较左右值即可获得所有的后代节点，缺点是增删一个节点，需要调整和他相关的所有节点的左右值。

7.将数据表中某个子节点移到另一个节点之下，考察同学对增删改节点导致的树形变化的处理能力。