Part 13 事务基础知识

为了保持一致性，和恢复机制。

1. 数据库事务概述

1.1 存储引擎支持情况

`SHOW ENGINES` 命令来查看当前 MySQL 支持的存储引擎都有哪些，以及这些存储引擎是否支持事务。在 MySQL 中，只有InnoDB 是支持事务的。

1.2 基本概念

事务：一组逻辑操作单元，使数据从一种状态变换到另一种状态。

事务处理的原则：保证所有事务都作为 `一个工作单元` 来执行，即使出现了故障，都不能改变这种执行方式。当在一个事务中执行多个操作时，要么所有的事务都被提交(`commit`)，那么这些修改就 `永久` 地保存下来；要么数据库管理系统将 `放弃` 所作的所有 `修改` ，整个事务回滚(`rollback`)到最初状态。

1.3 事务的ACID特性

- **原子性（atomicity）**：原子性是指事务是一个不可分割的工作单位，要么全部提交，要么全部失败回滚。
- **一致性（consistency）**：根据定义，一致性是指事务执行前后，数据从一个 `合法性状态` 变换到另外一个 `合法性状态` 。这种状态是 `语义上` 的而不是语法上的，跟具体的业务有关。比如有200元，无法转出300元，需要符合现实对余额的要求。
- **隔离型（isolation）**：事务的隔离性是指一个事务的执行 `不能被其他事务干扰` ，即一个事务内部的操作及使用的数据对 `并发` 的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。
- **持久性（durability）**：持久性是指一个事务一旦被提交，它对数据库中数据的改变就是 `永久性的` ，接下来的其他操作和数据库故障不应该对其有任何影响。持久性是通过 `事务日志` 来保证的。日志包括了 `重做日志` 和 `回滚日志` 。

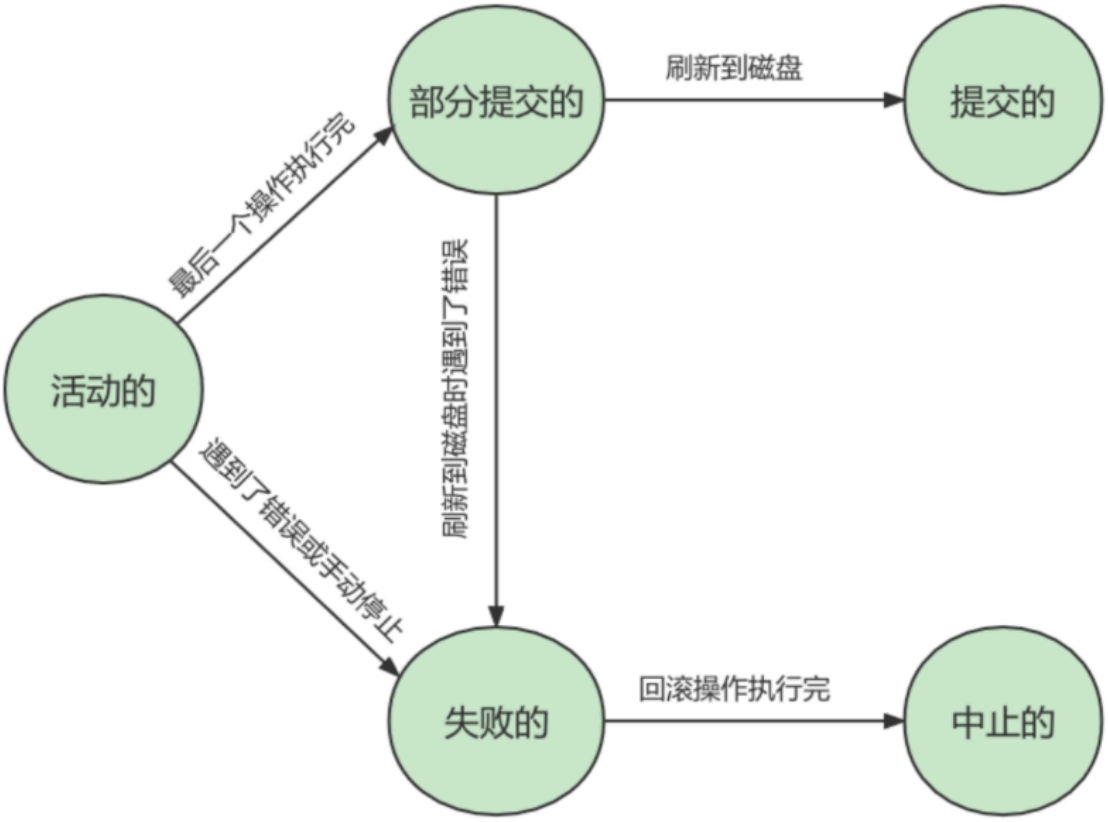
> 总结
>
> ACID是事务的四大特性，在这四个特性中，原子性是基础，隔离性是手段一致性是约束条件，而持久性是我们的目的。
>
> 数据库事务，其实就是数据库设计者为了方便起见，把需要保证 `原子性` 、 `隔离性` 、 `一致性` 和 `持久性` 的一个或多个数据库操作称为一个事务。

1.4 事务的状态

我们现在知道 `事务` 是一个抽象的概念，它其实对应着一个或多个数据库操作，MySQL根据这些操作所执行的不同阶段把 `事务` 大致划分成几个状态：

- 活动状态
- 部分提交

- 失败
- 中止
- 提交



2. 如何使用事务

使用事务有两种方式，分别为 显式事务 和 隐式事务 。

2.1 显式事务

- 步骤1：显示开启一个事务
  - ```
    # 方式一
    mysql> BEGIN;
    # 方式二
    # 后面可以跟修饰符：READ ONLY，READ WRITE，WITH CONSISTENT SNAPSHOT(一致性读)
    mysql> START TRANSACTION;
    ```
- 步骤2：一系列事务中的操作（主要是DML，不含DDL）
- 步骤3：提交事务 或 中止事务（即回滚事务）
    ```

```
# 提交事务。当提交事务后，对数据库的修改是永久性的。
mysql> COMMIT;

# 回滚事务。即撤销正在进行的所有没有提交的修改
mysql> ROLLBACK;

# 将事务回滚到某个保存点。
mysql> ROLLBACK TO [SAVEPOINT]
```

2.2 隐式事务

MySQL中有一个系统变量 autocommit ， 控制自动提交功能。

```
mysql> SHOW VARIABLES LIKE 'autocommit';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| autocommit | ON |
+---------------+-------+
1 row in set (0.01 sec)
```

关闭自动提交功能：

- 显式的的使用 START TRANSACTION 或者 BEGIN 语句开启一个事务。这样在本次事务提交或者回滚前会暂时关闭掉自动提交的功能。
- 把系统变量 autocommit 的值设置为 OFF 。

2.3 隐式提交数据的情况

- 数据定义语言（Data definition language，缩写为：DDL）
- 隐式使用或修改mysql数据库中的表
- 事务控制或关于锁定的语句
  - 当我们在一个事务还没提交或者回滚时就又使用 `START TRANSACTION` 或者 `BEGIN` 语句开启了另一个事务时，会 `隐式的提交` 上一个事务。
  - 当前的 `autocommit` 系统变量的值为 `OFF` ，我们手动把它调为 `ON` 时，也会 `隐式的提交` 前边语句所属的事务。
  - 使用 `LOCK TABLES` 、 `UNLOCK TABLES` 等关于锁定的语句也会 `隐式的提交` 前边语句所属的事务。

2.4 使用举例1：提交与回滚

当我们设置 autocommit=0 时，不论是否采用 START TRANSACTION 或者 BEGIN 的方式来开启事务，都需要用 COMMIT 进行提交，让事务生效，使用 ROLLBACK 对事务进行回滚。

当我们设置 autocommit=1 时，每条 SQL 语句都会自动进行提交。 不过这时，如果你采用 START TRANSACTION 或者 BEGIN 的方式来显式地开启事务，那么这个事务只有在 COMMIT 时才会生效，在 ROLLBACK 时才会回滚。

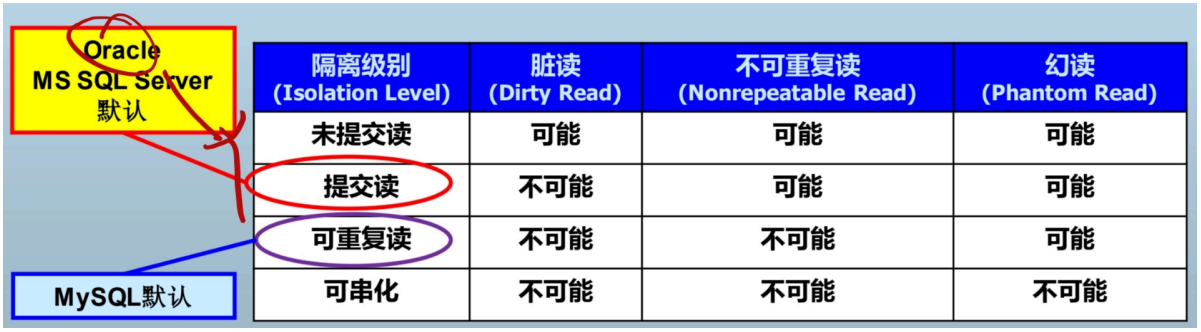
2.5 使用举例2：测试不支持事务的engine

INNODB支持事务，MyISAM不支持事务。

2.6 使用举例3：SAVEPOINT

在事务结束之前，如果在某处设置savepoint，可以通过`ROLLBACK TO savpoint1`回滚到savepoint处。

3. 事务隔离级别



隔离级别 (Isolation Level)	脏读 (Dirty Read)	不可重复读 (Nonrepeatable Read)	幻读 (Phantom Read)
未提交读	可能	可能	可能
提交读	不可能	可能	可能
可重复读	不可能	不可能	可能
可串化	不可能	不可能	不可能

3.1 数据并发问题

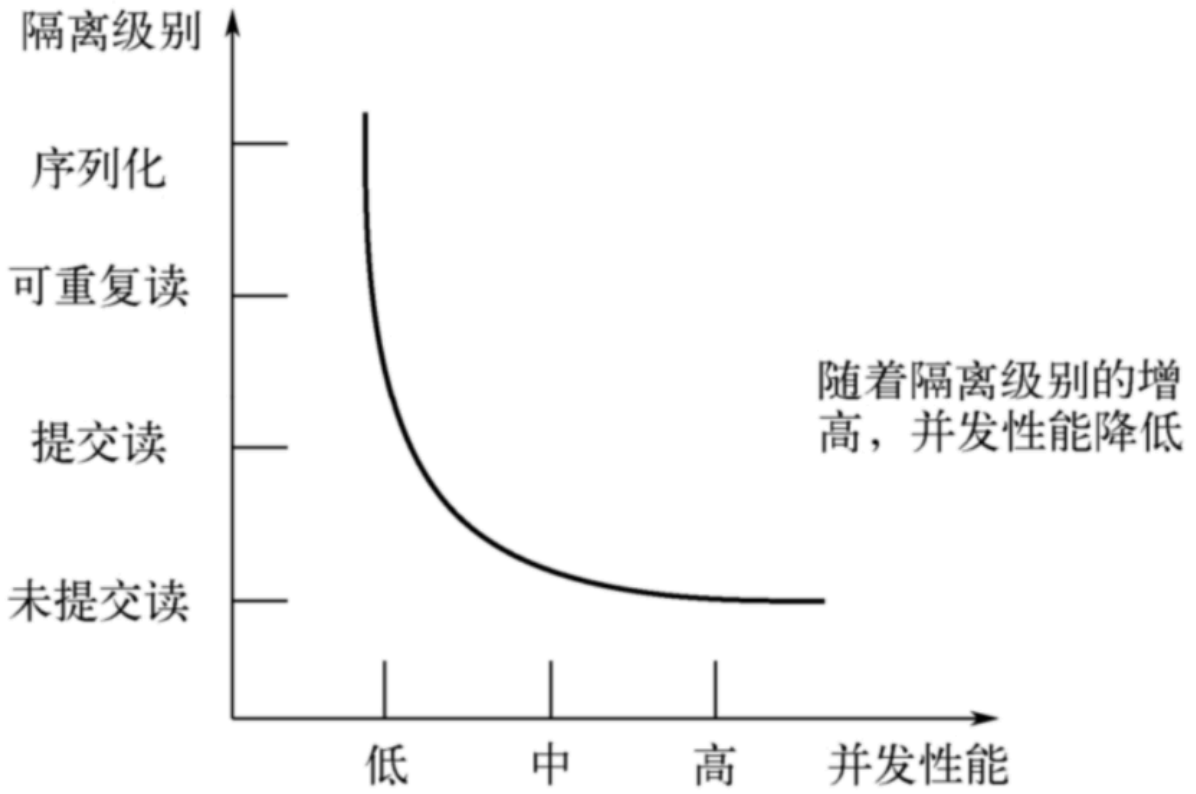
- 脏写（Dirty Write）
  - 对于两个事务 Session A、Session B，如果事务Session A 修改了另一个未提交事务Session B 修改过 的数据，那就意味着发生了脏写。
- 脏读（Dirty Read）
  - 对于两个事务 Session A、Session B，Session A 读取了已经被 Session B 更新但还没有被提交的字段。之后若 Session B 回滚，Session A 读取 的内容就是临时且无效的。
- 不可重复读（Non-Repeatable Read）
  - 对于两个事务Session A、Session B，Session A 读取 了一个字段，然后 Session B 更新 了该字段。 之后Session A 再次读取 同一个字段， 值就不同 了。那就意味着发生了不可重复读。
- 幻读（Phantom）
  - 对于两个事务Session A、Session B。Session A从一个表中读取了一个字段，然后 Session B 在该表中插入了一些新的行。 之后，如果 Session A 再次读取同一个表，就会多出几行。那就意味着发生了幻读。

3.2 SQL中的四种隔离级别

按严重性：脏写 > 脏读 > 不可重复读 > 幻读

- READ UNCOMMITTED： 未提交读
- READ COMMITTED： 提交读
- REPEATABLE READ： 可重复读
- SERIALIZABLE： 可串行化

不同的隔离级别有不同的现象，并有不同的锁和并发机制，隔离级别越高，数据库的并发性能就越差，4种事务隔离级别与并发性能的关系如下：



3.3 如何设置事务的隔离级别

通过下面的语句修改事务的隔离级别：

```
SET [GLOBAL|SESSION] TRANSACTION ISOLATION LEVEL 隔离级别;
# 或者
SET [GLOBAL|SESSION] TRANSACTION_ISOLATION = '隔离级别';

#其中，隔离级别格式:
> READ UNCOMMITTED
> READ COMMITTED
> REPEATABLE READ
> SERIALIZABLE
```

关于设置时使用GLOBAL或SESSION的影响：

- GLOBAL：在全局范围影响

```
SET GLOBAL TRANSACTION ISOLATION LEVEL SERIALIZABLE;
#或
SET GLOBAL TRANSACTION_ISOLATION = 'SERIALIZABLE';
# 当前已经存在的会话无效，只对执行完该语句之后产生的会话起作用
```

- SESSION：在会话范围影响

```
SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
#或
SET SESSION TRANSACTION_ISOLATION = 'SERIALIZABLE';
# 对当前会话的所有后续的事务有效；如果在事务之间执行，则对后续的事务有效；该语句可以在已经开启的事务中间执行，但不会影响当前正在执行的事务
```

4. 事务的常见分类

从事务理论的角度来看，可以把事务分为以下几种类型：

- 扁平事务（Flat Transactions）
  - 事务类型中最简单但使用最频繁的事务。 在扁平事务中，所有的操作都处于同一层次，由BEGIN/START TRANSACTION开始事务，由COMMIT/ROLLBACK结束，且都是原子的，要么都执行，要么都回滚。
- 带有保存点的扁平事务（Flat Transactions with Savepoints）
  - 除了支持扁平事务支持的操作外，允许在事务执行过程中回滚到同一事务中较早的一个状态。
- 链事务（Chained Transactions）
  - 在提交一个事务时，释放不需要的数据对象，将必要的处理上下文隐式地传给下一个要开始的事务。
- 嵌套事务（Nested Transactions）
  - 由一个顶层事务（top-level transaction）控制着各个层次的事务。顶层事务之下嵌套的事务被称为子事务（subtransaction），其控制每一个局部的变换。
- 分布式事务（Distributed Transactions）
  - 一个在分布式环境下运行的扁平事务，因此需要根据数据所在位置访问网络中的不同节点。