7.1事务的概念与性质

◆ 7.1.1. 事务的定义

事务是单个用户或应用程序执行的、用来读取或更新数据库内容的单个或多个操作。

◆**说明：**

- 事务是数据库操作的逻辑单位。
- 事务应该将数据库从一种一致性状态转换到另一种一致性状态。尽管当事务处于进程中时，数据库的一致性遭可能会到破坏。
- **一致性状态**是指数据的**正确性、有效性**和**相容性**。

7.1.1. 事务的定义

- DBMS无法知道哪些数据库操作被组合在一起构成一个独立的逻辑事务，因此，必须提供一种方法使用户能够定义**事务的边界**。
- 在SQL中，定义**事务的开始**有显式方式 和 隐式方式：
  - **显式方式**，事务以**BEGIN TRANSACTION开始**；
  - **隐式方式**，DBMS按默认规定自动划分事务，一般就是以某个select或insert等SQL开始。

  采用哪种方式，取决于所用的DBMS产品中对SQL的语法规定。
- 用**Commit 或 Rollback** 为结尾表示**事务结束**。
  - **Commit**表示事务正常结束。
  - **Rollback**表示事务异常终止。

7.1.2 事务的性质

事务的**ACID**特性：

- **原子性**（Atomicity）

  事务中包括的所有操作要么都做，要么都不做。

- **一致性**（Consistency）

  事务执行的结果必须是使数据库从一个一致性状态变到另一个一致性状态。

- **隔离性**（Isolation）

  事务之间的执行是相互独立的，不能被其他事务干扰，未完成事务的中间结果对于其他事务是不可见的。

- **持久性**（Durability）

  一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。以后的故障也不会对其执行结果有任何影响。

7.2 并发可能导致的三类问题

- **串行的定义**

  事务一个一个地依次执行，一个事务结束了，另一个事务才能开始运行，称为串行执行。

- **并发的定义**

  在单处理机系统中，事务中的操作轮流交叉运行的过程，称为并发。

7.2 并发可能导致的三类问题

并发控制的定义

为了提高数据库的利用率，通常允许多个用户同时使用数据库系统，即并发地访问共享数据库。**管理数据库上的并发操作，使之不相互干扰的过程，称为并发控制。**

特点：在同一时刻并发运行的事务数可达数百上千个。

- 若所有用户都是读取数据，并发控制就非常简单了，因为相互之间不可能产生干扰。
- 但若多个用户中**至少有一个**用户要对**更新**数据库，就可能导致数据库出现不一致的状态。
- 虽然两个事务本身都是正确的，但交互操作却可能导致不正确得结果。

(1) 丢失更新

一个用户对数据库所做的成功修改操作被另一个用户的更新操作覆盖了。

Time	bal_x初始值为100	T_1（取10元）	T_2（存100元）	bal_x
t_1			begin_transaction	100
t_2		begin_transaction	read(bal_x)	100
t_3		read(bal_x)	$bal_x = bal_x + 100$	100
t_4		$bal_x = bal_x - 10$	write(bal_x)	200
t_5		write(bal_x)	commit	90
t_6		commit		90

原因：T_1的操作结果覆盖了T_2提交的结果，导致T_2的修改被丢失。

避免的方法：禁止事务T_1读取bal_x的值，直到T_2完成更新。

（2）未提交依赖问题----读脏数据

- 事务A修改某一数据，还未将其写回磁盘，就被另一事务B读取了该数据。
- 随后，事务A由于某种原因被撤销，此时被事务A修改过的数据就恢复原值。
- 而事务B读到的数据就与数据库中的数据不一致，称为**“脏”数据**，即不正确的数据。
- 总之，**“脏”数据**是未提交的、随后又被撤销了的数据。

（3）不一致分析问题

- 事务**A**从数据库中读取多个数据值，当另一事务**B**却在其执行过程中修改了其中的某些值，这时就会出现**不一致分析问题。**

Time	T_5	T_6	bal_x	bal_y	bal_z	sum
t_1		begin_transaction	100	50	25	
t_2	begin_transaction	sum = 0	100	50	25	0
t_3	read(bal_x)	read(bal_x)	100	50	25	0
t_4	$bal_x = bal_x - 10$	sum = sum + bal_x	100	50	25	100
t_5	write(bal_x)	read(bal_y)	90	50	25	100
t_6	read(bal_z)	sum = sum + bal_y	90	50	25	150
t_7	$bal_z = bal_z + 10$		90	50	25	150
t_8	write(bal_z)		90	50	35	150
t_9	commit	read(bal_z)	90	50	35	150
t_{10}		sum = sum + bal_z	90	50	35	185
t_{11}		commit	90	50	35	185

三个账户正确的总和为175，结果却是185.

避免的方法：禁止T6 在T5完成之前读取账户x和z的余额。

（3）不可重复读 或 幻读

不一致分析问题还有另外两种情况:

- **不可重复读（读----修改）**

  是指事务T1读取数据项A的值后，事务T2 修改了A的值，当T1再　次读取A的值时，无法再现前一次读取结果。

- **幻读（读----插入）**

  事务T1执行某查询操作后，事务T2插入新的元组，过一段时间T1再次执行此查询时，发现结果集中包含了新元组。

7.3 可串行化调度
(Serializable Schedule)

◆ **调度的定义：**事务的执行次序称为**调度**，它同时满足以下条件：

- 调度必须包含并发事务中的所有操作。
- 调度必须**保持**每个事务内各操作之间的**相对次序**。

◆ 对多个事务处理的两种方法：

- **串行调度：**事务的依次执行称为串行调度。
- **并发调度：**利用分时的方法，同时处理多个事务，称为**事务的并发调度**。

可串行化(Serializable)调度

- 定义：当事务集合的并发调度结果与这些事务按某个串行调度的执行结果相同时，称这样的**并发调度为可串行化调度。**
- 可串行化调度被认为是**正确的并发调度。**
- 为提高并发程度，就是要找到**可串行化的并发调度。**

封　锁

- 事务T在对某个数据对象（例如表、记录等）进行**读、写操作之前**，必须先向并发控制管理器**发出请求**，对其加X锁或S锁；
- 只有**获得了封锁**后，事务才能**继续进行**，否则就必须等待，直到现有的锁被释放为止。

锁的相容矩阵

T_1 \ T_2	X	S	–
X	N	N	Y
S	N	Y	Y
–	Y	Y	Y

S锁只能与S锁共存，因为读操作是不冲突的，
因此同一时刻可以有多个事务对同一数据项加读锁。

7.4.2 两段锁协议 (2PL)

- 若事务中所有的加锁操作都在事务的第一个解锁操作之前进行，则说这个事务遵循两段锁协议。
- 所有事务必须分**两个阶段**对数据项**加锁**和**解锁**：
  - **扩展阶段（加锁阶段, Growing phase ）**

    事务可以申请获得任何数据项上的任何类型的锁，但是不能释放任何锁.
  - **收缩阶段（解锁阶段, Shrinking phase ）**

    事务可以释放任何数据项上的任何类型的锁，但是不能再申请任何锁.

锁的使用方式

- 协议**并不要求同时获得**所有的锁.
- 通常，事务先获得一些锁，进行相应的处理，然后再根据需要请求其他的锁。
- 事务在到达不再需要任何新锁的阶段前，它不会释放任何锁。
- 事务在对一个数据项进行**操作之前**，**必须先获得**对该数据项的**锁**。所加锁的类型取决于访问的类型（写或读）.

定理

- **定理：**若并发事务都遵守两段锁协议，则这些事务的任何并发调度都是可串行化的。
- 事务遵守两段锁协议是可串行化调度的**充分条件，而不是必要条件**。
- 即使不遵守两段锁协议，并发调度也可能是可串行化的。
- 若并发事务的一个调度是可串行化的，不一定所有事务都符合两段锁协议

7.4.3 死锁

- 两段锁协议可以有效地解决并发调度的可串行化问题，但也带来一个**新问题 ：死锁**。
- **死锁**：当两个（或多个）事务相互等待对方所拥有的锁被释放时，所产生的僵持局面。
- **这是因为每个事务都不能及时解除被它封锁的数据。**

T1	T2
LOCK S(B)	
READ (B)	
	LOCK S(A)
	READ (A)
XLOCK(A)	
WAIT…	
	XLOCK(B)
	WAIT…

（1）超时

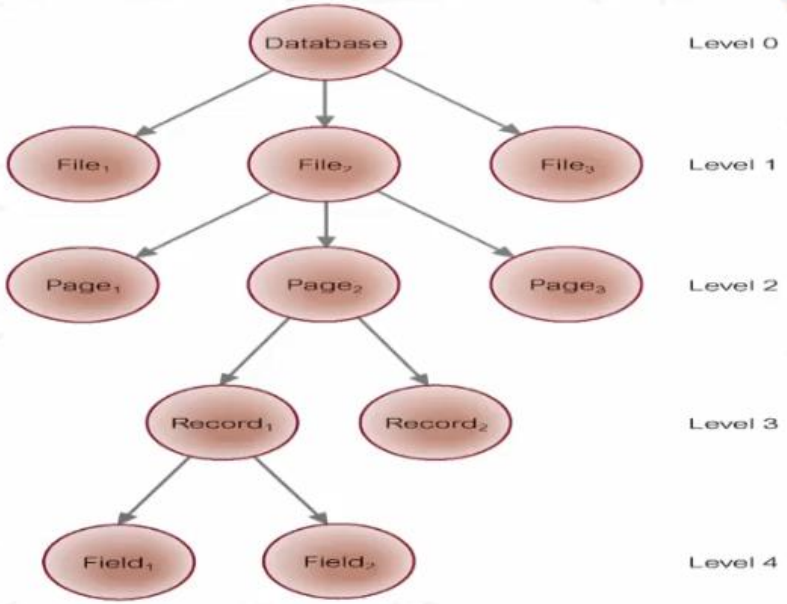
- 判断标准：如果一个事务的等待时间超过了规定的时限，就认为发生了死锁。
- 优点：实现简单
- 缺点：
  - （1）若时限设置得太短，有可能误判死锁；
  - （2）若时限设置得太长，死锁发生后不能及时发现

（2）死锁检测和恢复

- 通常构造**事务等待图 (wait-for graph，WFG)** 表示事务间的等待情况:
  - 事务等待图是一个有向图G=(T，U)
  - T为**结点**的集合，每个结点表示正运行的**事务**
  - U为**边**的集合，每条边表示**事务等待**的情况
  - 若T1等待T2，则在T1和T2之间划一条有向边，从T1指向T2
  - 如果发现图中**存在回路**（Topological Sorting），则表示系统中**出现了死锁。**
- 死锁检测的频率：周期性地（比如每隔数秒）生成事务等待图，检测其中是否有回路存在。

多粒度封锁

- 在一个系统中同时支持多种封锁粒度，供不同的事务选择。
- 以**树形结构**来表示多级封锁粒度。
- 根结点是整个数据库，表示最大的数据粒度。
- 数据库的子结点为文件。
- 叶结点表示最小的数据粒度。
- 如果对一个结点加锁，则该结点所有子孙结点也被加锁 。
- 若另一个事务**请求**对**已加锁节点**的任意**子孙**节点加锁，则DBMS先检查从根节点到请求节点的层次路径，确定是否有其祖先节点被加锁，再决定是否同意对请求节点加锁。



选择封锁粒度原则

- 封锁粒度与系统的并发度和并发控制的开销密切相关。
  - 封锁的**粒度越大**，数据库所能够封锁的数据单元就越少，并发度就越小，系统开销也越小；
  - 封锁的**粒度越小**，并发度越高，但系统开销也就越大
- 选择封锁粒度：
  - 需要处理少量元组的事务：以记录为封锁单元
  - 只处理大量元组的事务：以页为封锁单位

7.6 隔离级别

- **隔离级别**表明一个事务在与其他事务**并发**执行时**允许交互的程度**，或是能**容忍干扰的程度**。
- SQL标准规定有4个隔离级别：
  - **Serializable** —（可串行化，默认设置）
  - **Repeatable read**（可重复读）
  - **Read committed**（已提交读）
  - **Read uncommitted**（未提交读）
- 4个隔离级别从上到下依次从高到低。
- Serializable隔离级别最高，只有它是安全的，保证可产生可串行调度，使事务的ACID 性质得以实现。
- **隔离级别越高，干扰越少，并发程度越低；**
- **隔离级别越低，干扰越多，并发程度越高；**

SQL中的事务隔离级别

隔离级别
低
↓
高

级别	丢失更新	读脏数据	不可重复读	幻读
READ UNCOMMITTED	No	Maybe	Maybe	Maybe
READ COMMITTED	No	No	Maybe	Maybe
REPEATABLE READ	No	No	No	Maybe
SERIALIZABLE	No	No	No	No

并发程度
高
↓
低

No表示“不会产生此问题”； Maybe表示“可能产生此问题”