CH18 Concurrency Control并发控制

一、Serial and Serializable Schedules

1. 多事务执行方式

- 串行执行
- 交叉并发
- 同时并发（并行）

2. 并发导致的数据异常

- 脏读：读取其他事务未提交数据，且其他事务错误回滚
- 不可重复读：前后多次读取，数据内容不一致（其他事务更新）
- 幻读：按照相同查询条件两次查询数据不一致（其他事务插入）

事务隔离等级：

	脏读	不可重复读	幻读
READ_UNCOMMITTED（允许读未提交内容）	允许	允许	允许
READ_COMMITTED（事务提交以后才可读）	不允许	允许	允许
REPEATABLE_READ（事务开始读取后禁止更新）	不允许	不允许	允许
SERIALIZABLE（完全串行化）	不允许	不允许	不允许

3. 概念

- 并发控制：保证事务执行时数据库的一致性的过程
- 调度器：从事务中获取read/write操作，选择执行他们或者推迟他们

- 调度：按时间先后顺序执行操作
- 串行化调度：不存在交叉并发
- 可串行化调度：存在一个串行化调度，使得二者执行结果相同。

4. 记号

$r_i(X), w_i(X)$

$T_1 : r_1(A); w_1(A); r_1(B); w_1(B);$

$S = r_1(A)w_1(A)r_2(A)w_1(B)r_3(A)$

二、Conflict-Serializability

1. 冲突操作

不同事物对一个相同元素：一读一写或两写都会引发冲突，不可交换顺序。

2. 冲突等价

如果一个调度能通过一些不冲突交换转换为另一个调度，则称这两个调度冲突等价。

3. 冲突可串行化

一个调度是S冲突可串行化的当且仅当存在一个串行调度S'和S冲突等价。

冲突可串行化调度一定是一个**可串行化调度。**

反之不一定成立。（只在特例中存在）

4. 前序图

- 节点：事务
- 边：$T_i \to T_j$，存在冲突操作$W_i(A)$和$W_j(A)$（或其他冲突操作）

判断是否冲突可串行化 $\Longleftrightarrow$ 判断前序图是否无环

三、Enforcing Serializability by Locks

1. 锁

为确保冲突可串行化，使用锁。

对于调度中的每一个操作，要么将**事务**阻塞，等到时机合适释放，要么执行语句。

2. 记号

$l_i(X); u_i(X);$

3. 规则

> 任何事务在读/写操作之前需要申请锁，随后在合适时机释放锁。
>
> 在一个事务对一个元素的$l(X)$和$u(X)$之间，不允许出现其他事务对X申请锁。

4. 二阶段锁（2PL）

对每个事务而言，存在一个分解线，使得上锁语句只出现在分解线之前，解锁语句只出现在分解线之后。

保证冲突可串行化。

四、Locking System with Serveral Lock Modes

1. 二阶段锁的问题

禁止了并发读，影响效率。

2. 双模式锁

- Shared locks（共享锁）
- Exclusive locks（排它锁）

对任意元素X来说，要么存在一个排它锁，要么存在数个共享锁。

3. 记号

$sl_i(X); xl_i(X); u_i(X)$

***$u_i(X)$为解除事务T_i对X所有的锁的语句。

4. 规则

> 读操作之前必须存在共享锁或排它锁，写操作之前必须存在排它锁。
>
> 二阶段事务，所有的上锁过程在任意解锁过程之前
>
> 共享锁和排它锁无法同时存在（同一事务可以存在升级锁）

5. 兼容性矩阵

注意：持有和请求的不是同一个事务！！

（下）持有 请求（右）	Lock S	Lock X
Lock S	Yes	No
Lock X	No	No

6. 升级锁

事务一开始先申请共享锁，当事务准备写时，申请排它锁（Upgrade）

$ul_i(X)$：只有升级锁可以再次申请排它锁，共享锁无法升级。

	S	X	U
S	Yes	No	Yes
X	No	No	No
U	No	No	No

7. 解决死锁

> 破坏死锁条件（预防，不适用于DBMS）
>
> 超时法，事务等待图法（诊断与解除）

超时：事务执行时间超过时限，认为死锁发生（可能误判）

等待图：有向图表示等待关系，若发生循环等待，认为死锁发生

解除：选择一个处理死锁代价最小的事务，将其撤销，释放该事务持有的锁，使其他事务继续运行。