oracle锁机制

1.存在原因：**并发+数据一致性==>锁**

事务隔离级别也是基于锁实现的。

2.基本锁类型：

**排它锁**（Exclusive Locks，即X锁）：no read ，no write ;X+NULL（任何事务都不能再对当前数据对象加任何类型的锁）

**共享锁**（Share Locks，即S锁）：read only;S+S（表示其它事务只能对当前数据加S锁，而不能加X锁）

3.按保护对象的锁分类：

DML锁（data locks，**数据锁**），用于实现并发存取并保护**数据**的完整性；

DDL锁（dictionary locks，**字典锁**），用于保护**数据库对象**的结构，如表、索引等的结构定义；

**内部锁和闩**（internal locks and latches），保护**数据库内部结构**。比如数据库解析了一条查询语句并生成了最优化的执行计划，它将把这个执行计划“latche”在library cache中然后供其它session使用。

4.DML锁（最常见）分类

TM锁：**表级锁**（用来保证表的结构不被用户修改）

TX锁：**事务锁或行级锁**

TX锁的实现：在oracle的每行数据上，都有一个**标志位**来表示该行数据是否被锁定。数据行上锁标志一旦被置位，就表明该行数据被加X锁，oracle在数据**行上没有S锁**。

执行语句获取锁顺序：🡪对A表N行操作的DML语句🡪申请A表**TM锁**🡪获取后申请**TX锁**，锁标志位进行**置位**到N。（这样在事务加锁前检查TX锁相容性时就不用再逐行检查锁标志，而只需检查TM锁模式的相容性即可，大大提高了系统的效率）

TM锁分类：**共享锁（S锁）、排它锁（X锁）、行级共享锁（RS锁）、行级排它锁（RX）、共享行级排它锁（SRX锁）**

|  |  |  |  |
| --- | --- | --- | --- |
| 锁模式 | 锁描述 | 允许锁模式 | SQL操作 |
| 1 | NULL | RS、RX、S、SRX、X | **select \* from table\_name...** |
| 2 | RS | RS、RX、S、SRX | **lock table table\_name in row share mode** |
| 3 | RX | RS、RX | **insert into table\_name ...**  **update table\_name ...**  **delete from table\_name ...**  **select \* from table\_name for update**  **lock table table\_name in row exclusive mode** |
| 4 | S |  | **lock table table\_name in share mode** |
| 5 | SRX |  | Lock share row exclusive |
| 6 | X |  | Alter table、Drop able、Drop index、Truncate table 、Lock exclusive |

oracle数据库TM锁的相容矩阵（Y=YES，N=NO）：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **T1/T2** | **S** | **X** | **RS** | **RX** | **SRX** | **-** |
| **S** | Y | N | Y | N | N | Y |
| **X** | N | N | N | N | N | Y |
| **RS** | Y | N | Y | Y | Y | Y |
| **RX** | N | N | Y | Y | N | Y |
| **SRX** | N | N | Y | N | N | Y |
| **-** | Y | Y | Y | Y | Y | Y |

5、死锁

1）死锁发生在两个session同时锁住了对方正请求的资源的情况下

演示情况如下：

create table tp1(id number(8,0),name varchar2(20));

create table tp2(id number(8,0),name varchar2(20));

*--id=(1,2,3) name =null*

*--Session1 更新 tp1*

SQL> update tp1 set name='session1' where id=1;

*--Session2更新tp2*

SQL> update tp2 set name='session2' where id=2;

*--session2又更新tp1中session1正在更新的行*

SQL> update tp1 set name='session2' where id=1;

此时session2阻塞

*--session1又更新tp2中session2正在更新的行*

SQL> update tp2 set name='session1' where id=2;

此时session1也被阻塞且session2报错：

ORA-00060: 等待资源时检测到死锁

ORA-00060: deadlock detected while waiting for resource

如果此时session1不commit或rollback，session2将一直被阻塞。

2）还有一种情况发生死锁是由没有索引的外键引起的，oracle会在下面两种情况给整个子表加锁：

当更新父表主键的时候，如果子表的外键没有索引则会给整个子表加锁。还有就是删除父表记录的时候，同样也会给子表加锁。

比如这样做：

*--session1*

create table p(x number primary key not null );*--父表*

create table c(x number references p);*--子表*

insert into p values(1);

insert into p values(2);

commit;

insert into c values(2);

*--session2*

delete from p where x=1;

此时session2立刻被阻塞，此时其它session（如session3）都不能够再对c表做insert、delete及update操作，

如果执行这样的操作则会立刻被阻塞并当提交或回滚session1时session2就会提示发现死锁：

*--session3*

SQL> insert into c values(1);

被阻塞

*--session1*

SQL> rollback;

*--session2*

SQL> delete from p where x=1;

delete from p where x=1

\*

第 1 行出现错误:

ORA-00060: 等待资源时检测到死锁

实际上此时并没有死锁，但是当其它sesion（如session3）在更新c表之前占有了其它资源，而此时session1又去请求这个资源时就会造成死循环从而导致死锁。

如果给c表的外键加了索引则可以避免死锁：

*--session1*

create index idx\_c on c (x);

insert into c values(1);

*--session2*

SQL> delete from p where x=1;

此时还是会被阻塞

*--session3*

SQL> insert into c values(1);

同样被阻塞

*--session1*

SQL> rollback;

*--session2提示违反完整性约束而不会产生死锁*

SQL> delete from p where x=1;

delete from p where x=1

\*

第 1 行出现错误:

ORA-02292: 违反完整约束条件 (ICEBERG.SYS\_C005335) - 已找到子记录

建议：能够在程序里实现数据完整性约束就尽量不要使用主键和外键关联。