第 16 章 锁

锁是用来控制访问共享资源的一种机制。大型数据库系统的一个重要特征是要保证多个并发用户一致地读写数据。若多个并发写操作要修改同样数据或多个读写操作要一致时，就要用到锁的机制了。不同数据库产品对锁的实现有很大区别。多数数据库系统(如 Sybase，Informix，SQL Server 等)使用内存管理锁，为降低内存消耗，要尽快释放锁，从而要尽快结束事务，或者把多个行锁升级为一个表锁，达到降低内存消耗的目的。Oracle 的行锁被作为行属性，不占用内存资源，不存在尽快结束事务或升级锁的需要。

本章内容主要包括：

- Oracle 的锁
- SQL Server 的锁
- 死锁

16.1 Oracle 的锁

Oracle 中的锁共有三种： DDL 锁，内部拴锁及 DML 锁。Oracle 使用多版本数据技术满足读写一致性，其读取操作不会使用锁，这是 Oracle 和 SQL Server 的一个明显区别。

DDL 锁用于保护数据对象的结构不被其他事务修改。内部拴锁用于保护数据库内部结构不被修改。这里我们只讨论 DML 和 DDL 锁。

DML 锁又分为两种：TM 锁和 TX 锁。在用户发出 DML 命令时，Oracle 会自动对其影响的记录和表加上 TX 锁及 TM 锁。TX 锁用于锁住修改的记录，防止其他事务同时修改。TM 锁用于锁住被修改的表，防止其他事务对此表执行 DDL 语句修改表的结构。

查询动态字典视图 v$lock 可以得到 TM 锁及 TX 锁的信息。其 type 列表示锁的类型。当锁的类型为 TM 时，v$lock 中的 id1 字段表示锁住的对象 id 编号(object_id)。锁的类型为 TX 时，id1 及 id2 列表示获得锁的事务信息。

下面实验，通过查询 v$lock 查看当前锁的信息。

在连接 1 中以 scott 帐号登录，对 dept 表执行 update 操作：

```
SQL> conn scott/tiger
已连接。
SQL> update dept set loc='BOSTON' where deptno=10;
```

然后在连接 2 中以 system 帐号登录，查询连接 1 的会话编号：

```
SQL> conn system/oracle
已连接。
SQL> select sid from v$session where username='SCOTT';

       SID
----------
       365
```

继续查询会话编号 140 产生的锁信息：

```
SQL> select type,id1,id2 from v$lock where sid=365;

TY        ID1        ID2
-- ---------- ----------
```

```
AE         133          0
TX      524309        631
TM       19764          0
```

执行下面查询可以验证，TM 锁中的 19764 为 dept 表的 id 编号：

```
SQL> select object_id,object_name from dba_objects where object_id=19764;

 OBJECT_ID OBJECT_NAME
---------- ------------------------------
     19764 DEPT
```

在上述步骤执行 update 操作的基础上，在连接 2 中修改 dept 表的结构，删除 loc 字段，这会在 dept 表上附加 DDL 锁：

```
SQL> alter table dept drop column loc;
alter table dept drop column loc
            *
第 1 行出现错误:
ORA-00054: 资源正忙, 但指定以 NOWAIT 方式获取资源, 或者超时失效
```

可以发现此命令不能被成功执行，这里的错误信息并未指出实际出错的原因，其实是因为连接 1 中的 update 操作对 dept 表加上了 TM 锁，与此处的 DDL 锁产生了冲突。

16.2 SQL Server 的锁

因为管理锁的机制不同，SQL Server 和 Oracle 在锁的种类方面也不同。

16.2.1 SQL Server 中锁的种类

SQL Server 中的锁有很多种，常用的锁包括下面几种：

- 共享锁
- 排他锁
- 更新锁
- 意向锁
- 架构锁

16.2.2 共享锁

多个共享锁可以同时附加在同一个资源上，这是共享的含义。共享锁一般在查询操作中使用，可以附加在表、数据页或记录上，一个资源可以由多个查询操作访问。

共享锁在数据读取完成后即释放，如在 read committed 隔离级别下，使用全表扫描方式查询表时，会对扫描到的记录附加共享锁，若是目标记录则读出，然后释放其锁，若不是目标记录，则马上释放锁，这些锁不会保持到查询完成，更不会保持到事务结束。

共享锁的主要作用是避免脏读(dirty read)，脏读是读取了已更新而未提交的数据。若查询的数据正在被其他事务修改，则这些数据会被附加排他锁，从而查询操作要附加的共享锁与其冲突，要等待事务结束把排他锁释放掉才能附加，这样就避免了脏读。

为了在 SQL Server 中查看锁的情况，执行下面命令构造 dbLocks 视图：

```
create view dbLocks
as
select request_session_id as spid,
     case when resource_type='OBJECT' then
```

```
                object_name(resource_associated_entity_id)
           when resource_associated_entity_id=0 then 'n/a'
     else object_name(p.object_id)
     end as entity_name,
index_id, resource_type, resource_description as description,
     request_mode as mode, request_status as status
from sys.dm_tran_locks t left join sys.partitions p
on p.hobt_id=t.resource_associated_entity_id
where resource_database_id=db_id() and resource_type!= 'DATABASE'
```

下面我们通过说明共享锁的作用。

在连接 1 中创建测试表 t，并添加测试数据：

```
1> create table t(a int, b int)
2> go
1> insert into t values(1,10)
2> insert into t values(2,20)
3> insert into t values(3,30)
4> insert into t values(4,40)
5> go
```

在连接 1 中，执行下面 update 操作以产生排他锁：

```
1> begin tran
2> update t set b=200 where a=1
3> go
```

查询当前锁的情况：

```
1> select entity_name,resource_type,description,mode,status
2> from dbLocks
3> where entity_name='t'
4> go
entity_name     resource_type   description     mode            status
--------------- --------------- --------------- --------------- ----------
t               RID             3:8:0           X               GRANT
t               PAGE            3:8             IX              GRANT
t               OBJECT                          IX              GRANT

(3 行受影响)
```

可以发现被修改记录的 rowid 上附加了排它锁(X)，此记录所在的数据页及表 t 上附加了意向排它锁(IX)。

打开一个新连接 2，在其中进行查询：

```
1> select * from t
2> go
```

可以发现这个查询被阻塞，发生等待。回到连接 1，重新查询锁的情况：

```
1> select entity_name,resource_type,description,mode,status
2> from dbLocks
3> where entity_name='t'
4> go
entity_name     resource_type   description     mode            status
--------------- --------------- --------------- --------------- --------------
t               RID             3:8:0           X               GRANT
t               RID             3:8:0           S               WAIT
t               PAGE            3:8             IS              GRANT
t               PAGE            3:8             IX              GRANT
```

```
t               OBJECT                          IS              GRANT
t               OBJECT                          IX              GRANT

(6 行受影响)
```

可以发现查询结果多出了 3 行，连接 2 在查询记录所在的数据页及表 t 上获得了意向共享锁，执行全表扫描时，对其扫描到的第一行附加共享锁时，与已存在的排他锁冲突，从而未获得共享锁，导致这个查询被阻塞。

16.2.3 排他锁

正如其名称所示，除了 Sch-S 锁及 RangeI-N 锁以外，排他锁与任何其他锁都不兼容，执行 insert、update、delete 操作时会附加排他锁，排他锁一直保持到事务结束。如果在所访问的资源上存在其他锁，就不能再加上排他锁，同样，如果在访问的资源上已经存在排他锁，也不能在此资源上再加上其他锁。排他锁可以防止两个人同时修改相同的数据，从而解决更新丢失问题。SQL Server 在记录上附加的排他锁与 Oracle 的 TX 锁作用相同。

排他锁的示例请参考上节内容。

16.2.4 更新锁—SQL Server 真的支持行锁吗

在说明更新锁之前，先对上节示例创建的 t 表进行下面的简单操作(执行下面操作之前，请先回滚，结束之前的事务)：

在连接 1 对 t 表执行 update 操作，更新其 a=3 的记录：

```
1> begin tran
2> update t set b=300 where a=3
3> go
```

在连接 2 也对 t 表执行 update 操作，更新其 a=4 的记录：

```
1> update t set b=400 where a=4
2> go
```

虽然两个连接更新的不是同一条记录，但是连接 2 中的更新操作却发生了等待，SQL Server 声称支持行锁，为什么这里会发生等待呢？

答案是因为 SQL Server 在连接 2 执行 update 操作时，附加了更新锁。这可以从连接 1 中执行的下面查询确认，查询结果中的第 2 行表示正在等待获得更新锁：

```
1> select entity_name,resource_type,description,mode,status
2> from dbLocks
3> where entity_name='t'
4> go
entity_name     resource_type   description     mode            status
--------------- --------------- --------------- --------------- ---------------
t               RID             3:8:2           X               GRANT
t               RID             3:8:2           U               WAIT
t               PAGE            3:8             IU              GRANT
t               PAGE            3:8             IX              GRANT
t               OBJECT                          IX              GRANT
t               OBJECT                          IX              GRANT

(6 行受影响)
```

什么是更新锁呢？

更新锁主要是对表执行 update 操作时，在搜索目标记录过程中对记录附加的一种锁，

一般表示为 U 锁，若记录满足修改条件则把其 U 锁转换为 X 锁，若不满足修改条件，则把 U 锁释放。

U 锁与 S 锁兼容，但 U 锁与 U 锁不兼容，一行记录只能附加一个 U 锁。搜索目标记录附加 U 锁、而不是附加 S 锁的目的是为了降低发生死锁的几率。因为同一条记录可以附加多个 S 锁，若搜索满足修改条件的记录时，对记录附加 S 锁，可能会因为要同时转换为 X 锁而导致发生死锁。

如果一个连接修改一个表的记录 row_m，并对此记录附加了 X 锁，另外一个连接继续修改这个表的记录 row_n。在未使用索引的情况下，第二个连接搜索目标记录 row_n 时，会使用全表扫描对其搜索到的每一行记录附加 U 锁，当扫描至 row_m 并对其附加 U 锁时，就会与其已持有的 X 锁互斥，而被阻塞，从而发生等待。也就是说，在没有索引情况下，两个连接不能同时对同一个表执行 update、 delete 操作，即使这两个连接操作的是不同的目标记录。

解决这种等待问题的方法是对表创建索引，使得搜索目标记录时可以利用索引直接定位，不必使用全表扫描，从而避免在搜索 row_n 时对其他记录也附加 U 锁。

回滚结束连接 1 的事务后，重新进行上述实验，这次连接 1 先在 t 表的 a 列上创建非聚集索引，其他步骤相同。

```
1> create index idx_t on t(a)
2> go
1> begin tran
2> update t set b=300 where a=3
3> go
```

在连接 2 更新 a=4 的记录，这时不再等待：

```
1> update t set b=400 where a=4
2> go
```

之所以不再等待，是因为这时搜索 a=4 的记录不再需要进行全表扫描，而只要扫描索引就就可以直接访问到要更新的记录。

如果连接 2 也更新 a=3 的记录，则依然会发生等待：

```
1> update t set b=3000 where a=3
2> go
```

在连接 3 查看当前锁的情况：

```
1> select entity_name,resource_type,description,mode,status
2> from dbLocks
3> where entity_name='t'
4> go
entity_name     resource_type   description     mode            status
--------------- --------------- --------------- --------------- --------------
t               RID             3:8:2           X               GRANT
t               RID             3:8:2           U               WAIT
t               PAGE            3:8             IU              GRANT
t               PAGE            3:8             IX              GRANT
t               PAGE            3:24            IU              GRANT
t               KEY             (7c51e96a35a5)  U               GRANT
t               OBJECT                          IX              GRANT
t               OBJECT                          IX              GRANT

(8 行受影响)
```

由上述查询结果的第 2 行可以得知，发生等待的原因还是因为连接 2 要对 a=3 的记录附

加更新锁，与连接 1 附加的排他锁互斥。

如果在 a 列上创建聚集索引，这时的键值即 rowid，找到键值即找到记录，则连接 2 中的更新锁就不需要了。

在连接 1 回滚结束事务后，删除之前的非聚集索引，重新在 a 列上创建聚集索引，然后执行更新操作：

```
1> drop index t.idx_t
2> go
1> create clustered index idx_t on t(a)
2> go
1> begin tran
2> update t set b=300 where a=3
3> go
```

在连接 2 执行更新操作：

```
1> update t set b=3000 where a=3
2> go
```

发生了等待。在连接 3 查询当前锁的情况：

```
1> select entity_name,resource_type,description,mode,status
2> from dbLocks
3> where entity_name='t'
4> go
entity_name     resource_type   description     mode            status
--------------- --------------- --------------- --------------- --------------
t               PAGE            3:32            IX              GRANT
t               PAGE            3:32            IX              GRANT
t               OBJECT                          IX              GRANT
t               OBJECT                          IX              GRANT
t               KEY             (052c8c7d9727)  X               GRANT
t               KEY             (052c8c7d9727)  X               WAIT

(6 行受影响)
```

如上面结果最后一行所示，这时连接 2 直接在 a=3 的记录上附加排他锁。

16. 2.5 意向锁

意向锁是在对记录进行修改时，对其上层对象(page 或 table)附加的一种锁，具体示例请参考前面内容，mode 字段值为 IX 或 IU 的即为意向排它锁或意向更新锁。

16.2.6 架构锁

架构锁包括两种：

- Sch-M(Schema Modification)
- Sch-S(Schema-Stability)

对象结构被修改的时候，即执行 DDL 操作时，在此对象上会附加 Sch-M 锁，架构锁与任何其他锁都不兼容，目的是禁止在此对象上执行任何其他操作。Sch-M 锁与 Oracle 的 DDL 锁的功能相似。

当解析一个查询命令时，会对所涉及的对象附加 Sch-S 锁，目的是避免其他用户对其执行 DDL 操作，它只与 Sch-M 锁不兼容。

16.2.7 锁的升级

SQL Server 使用内存结构管理锁(每个锁使用 96 字节)，为了避免因行锁的个数太多而耗费内存，当一个表上的行锁数目达到一定限度时，这些行锁会升级为一个表锁，使得管理锁的内存大大降低。要注意的是，行锁升级的结果总是表锁，不会升级为页锁或数据库锁。Oracle 中行锁不占用内存资源，不存在锁升级的需要。

我们通过下面实验来观察锁的升级情况。

删除 t 表后，对其重建，并添加 8000 条记录：

```
1> set nocount on
2> go
1> drop table t
2> go
1> create table t(a int, b int)
2> go
1> insert into t values(1,1)
2> go 8000
```

通过下面实验可以看出，当表中被锁住的行数达到 6235 时，这些行锁就会升级为一个表锁(SQL Server 官方文档中的数字为 5000，与这里的测试结果不同)：

```
1> begin tran
2> update top(6234) t set b=b+1
3> go

(6234 行受影响)
```

查询当前锁的个数(这里的结果包括记录上的排他锁及其他锁)：

```
1> select count(*) from dbLocks where entity_name='t'
2> go

-----------
        6249
```

回滚结束以上事务后，再次开始一个事务，并更新 t 表中的 6235 条记录：

```
1> rollback
2> go
1> begin tran
2> update top(6235) t set b=b+1
3> go

(6235 行受影响)
```

查询锁的情况，可以发现行锁发生了升级，现在只有一个表锁了：

```
1> select entity_name,resource_type,description,mode,status
2> from dbLocks
3> where entity_name='t'
4> go
entity_name     resource_type   description     mode            status
--------------- --------------- --------------- --------------- --------------
t               OBJECT                          X               GRANT

(1 行受影响)
```

16.2.8 read uncommitted 隔离级别与锁

Oracle 的查询都不会产生锁，我们只需要讨论 SQL Server 的不同隔离级别下执行的查询产生锁的情况。

在 read uncommitted 隔离级别下，select 操作不再附加共享锁，这与在表上附加提示 nolock 的效果相同。下面通过实验证实此结论。

在连接 1 中回滚结束之前事务后，执行下面命令修改其第一行记录：

```
1> begin tran
2> update top(1) t set b=b+1
3> go

(1 行受影响)
```

查询当前锁的信息：

```
1> select entity_name,resource_type,description,mode,status
2> from dbLocks
3> where entity_name='t'
4> go
entity_name     resource_type   description     mode            status
--------------- --------------- --------------- --------------- ---------------
t               PAGE            3:64            IX              GRANT
t               RID             3:64:0          X               GRANT
t               OBJECT                          IX              GRANT

(3 行受影响)
```

在连接 2 设置 read uncommitted 隔离级别后，再重新执行上述查询：

```
1> set transaction isolation level read uncommitted
2> go
1> select top(1) * from t
2> go
a           b
----------- -----------
          1           2

(1 行受影响)
```

我们发现，这时不会发生等待的情况，如果回到连接 1 重新查询锁的情况，与之前查询结果相同，不会出现共享锁。

16.2.9 read committed 隔离级别与锁

与 read uncommitted 隔离级别不同，设置 read committed 隔离级别后，执行 select 操作时，会附加共享锁。

在连接 1 结束事务，重建 t 表并添加记录：

```
1> create table t(a int, b int)
2> go
1> insert into t values(1,10)
2> insert into t values(2,20)
3> insert into t values(3,30)
4> insert into t values(4,40)
5> insert into t values(5,50)
6> insert into t values(6,60)
7> go
```

在连接 1 重新开始一个事务更新 a=1 的记录：

```
1> begin tran
2> update t set b=100 where a=1
3> go
```

在连接 2 中重新设置隔离级别为默认的 read committed：

```
1> set transaction isolation level read committed
2> go
```

执行下面查询，查询条件为 a=4：

```
1> select * from t where a=4
2> go
```

可以发现，这时发生了等待。

在连接 1 查询当前锁的情况：

```
1> select entity_name,resource_type,description,mode,status
2> from dbLocks
3> where entity_name='t'
4> go
entity_name     resource_type   description     mode            status
--------------- --------------- --------------- --------------- ---------------
t               PAGE            3:304           IX              GRANT
t               PAGE            3:304           IS              GRANT
t               RID             3:304:0         X               GRANT
t               RID             3:304:0         S               WAIT
t               OBJECT                          IX              GRANT
t               OBJECT                          IS              GRANT

(6 行受影响)
```

由以上结果的第 4 行，可以得知等待的原因是因为要对 rowid 为 1:350:0 的行附加共享锁，这与已存在的 X 锁冲突。

仔细查看，会发现，要附加 S 锁的行不是连接 2 要查询的行，而是连接 1 中被修改的行。因为没有索引，只能对表 t 执行全表扫描搜索要查询的记录，对扫描到的记录先附加 S 锁，而不只对要查询的记录附加 S 锁，若扫描到的记录已经附加了排他锁，则连接 2 会发生等待，这与更新锁的效果相似。

若在 a 列上创建了非聚集索引，而且连接 2 中的查询用到了这个索引，则这时不再需要进行全表扫描，这也不会发生等待。若 a 列上创建了聚集索引，则索引叶节点数据即表的数据，连接 2 中的查询必定会用到这个索引，从而也不会发生等待。

在连接 1 执行回滚结束事务，然后在 a 列上创建索引后，重新执行更新操作：

```
1> create index idx_t on t(a)
2> go
1> begin tran
2> update t set b=100 where a=1
3> go
```

在连接 2 执行查询，并附加索引提示，强制其使用索引：

```
1> select * from t with (index(idx_t)) where a=4
2> go
```

可以发现，这时连接 2 不会发生等待情况。如果连接 1 是创建的聚集索引，则这里的索

引提示就不需要了，连接 2 的查询肯定会用到这个索引。

执行上述实验步骤时，要注意，如果先执行连接 2 中的查询，则不会查询到共享锁的信息，这是因为在查询执行完成后，锁就释放了，不会保持到事务结束，这与连接 1 中执行 update 时，产生的 X 锁一直保持到事务结束不同。

16.2.10 repeatable read 隔离级别与锁

设置 repeatable read 隔离级别下，在事务结束之前，查询操作对查询到的数据会一直加锁，从而其他事务不能对其执行 update。这与 read committed 隔离级别的情况显然不同，在 read committed 隔离级别，查询结束，附加到行的共享锁就释放了，而不会等到事务结束。

继续使用之前的 t 表进行下面的实验过程。

回滚结束连接 1 中的事务，然后在连接 1 中设置 repeatable read 隔离级别：

```
1> set transaction isolation level repeatable read
2> go
```

然后开始事务，并执行查询：

```
1> begin tran
2> select * from t where a between 3 and 6
3> go
a           b
----------- -----------
          3          30
          4          40
          5          50
          6          60

(4 行受影响)
```

在连接 1 查询锁的情况：

```
1> select entity_name,resource_type,description,mode,status
2> from dbLocks
3> where entity_name='t'
4> go
entity_name     resource_type   description     mode            status
--------------- --------------- --------------- --------------- --------------
t               PAGE            3:304           IS              GRANT
t               RID             3:304:4         S               GRANT
t               RID             3:304:2         S               GRANT
t               OBJECT                          IS              GRANT
t               RID             3:304:5         S               GRANT
t               RID             3:304:3         S               GRANT

(6 行受影响)
```

可以发现，连接 1 查询到的 4 条记录都被附加了 S 锁。

在 repeatable read 隔离级别下，SQL Server 会对查询结果附加共享锁，保持到事务结束，这样，其他连接就不能对这些记录进行 update、delete 等操作了。

对于这些查询结果之外的记录，其他连接对其进行 update 或 delete 时，则不受影响。另外注意，对查询结果之外的记录执行 update 操作时，有可能使其满足查询条件。

对 t 表进行 insert 操作，只会对新添加的记录附加 X 锁，与这些共享锁不会发生互斥作用，这样，对表 t 的 insert 操作不会发生等待的情况。

由上述分析可知，当另外连接的 insert 或 update 操作所在的事务提交后，如果在

repeatable read 隔离级别的事务中进行第二次查询，结果有可能比第一次多出一些。但第一次查询结果中的记录不会被其他连接修改，在第二次查询中也不会改变。

16.2.11 serializable 隔离级别与锁

讨论 serializable 隔离级别与锁的关系，要分为两种情况：

- 表上无索引
- 表上有索引

serializable 隔离级别下，当查询条件列上不存在索引时，事务中的查询涉及的表被附加表级共享锁，与查询时，附加 holdlock 提示的效果相同。下面进行验证。

在连接 1 中执行回滚结束之前的事务后，设置 serializable 隔离级别：

```
1> set transaction isolation level serializable
2> go
```

在连接 1 开始一个事务，并执行下面查询：

```
1> begin tran
2> select * from t where a=1
3> go
a           b
----------- -----------
          1          10

(1 行受影响)
```

在连接 1 查看当前锁的情况：

```
1> select entity_name,resource_type,description,mode,status
2> from dbLocks
3> where entity_name='t'
4> go
entity_name     resource_type   description     mode            status
--------------- --------------- --------------- --------------- --------------
t               OBJECT                          S               GRANT

(1 行受影响)
```

我们发现在无索引的情况下，连接 1 的查询导致对 t 表附加了 S 锁，而表上的 S 锁与 IX 锁互斥，执行 update、insert、delete 操作都要对表附加 IX 锁，这样，对表 t 执行这些操作时，都会发生等待，换句话说，表中所有的数据都不允许修改，也不允许对表添加新数据。

在无索引的情况下，如果对表 t 执行 update 或 delete 操作，则会对表 t 附加 X 锁，执行 insert 操作，对表附加 IX 锁，对添加的新记录附加 X 锁，请读者自行验证，这里不再赘述。

下面查看表 t 存在索引的情况。

在连接 1 中执行回滚结束之前的事务后，在 t 表的 a 列创建非聚集索引：

```
1> create index idx_t on t(a)
2> go
```

在连接 1 中执行 commit 命令提交之前的事务，重新执行上述查询，并强制使用新建的索引 idx_t：

```
1> begin tran
2> go
1> select * from t with (index(idx_t)) where a=3
2> go
```

```
a           b
----------- -----------
          3          30

(1 行受影响)
```

查看当前锁的情况：

```
1> select entity_name,resource_type,description,mode,status
2> from dbLocks
3> where entity_name='t'
4> go
entity_name     resource_type   description     mode            status
--------------- --------------- --------------- --------------- --------------
t               RID             3:312:2         S               GRANT
t               PAGE            3:312           IS              GRANT
t               PAGE            3:320           IS              GRANT
t               KEY             (5e1bcbc683af)  RangeS-S        GRANT
t               KEY             (e2016e92fb4c)  RangeS-S        GRANT
t               OBJECT                          IS              GRANT

(6 行受影响)
```

如果删除上述非聚集索引，在 a 列创建聚集索引，则同样的过程，会得到下面的锁：

```
1> select entity_name,resource_type,description,mode,status
2> from dbLocks
3> where entity_name='t'
4> go
entity_name     resource_type   description     mode            status
--------------- --------------- --------------- --------------- --------------
t               KEY             (1a39e6095155)  RangeS-S        GRANT
t               PAGE            3:328           IS              GRANT
t               KEY             (052c8c7d9727)  RangeS-S        GRANT
t               OBJECT                          IS              GRANT

(4 行受影响)
```

由上述结果，我们发现，当 a 列上存在索引时，连接 1 的查询导致在索引的两个键值上附加了键范围共享锁(键范围共享锁的 description 列值表示键散列值)，这时附加锁的不是整个表了，而是与查询条件的 a 值相关的一个较小范围，只要其他连接修改的记录不在这个禁止范围，都是允许的。

16.2.13 SQL Server 查询不使用锁的几种情况

除了对操作的数据资源加锁的方法外，从 SQL Server 2005 版本开始，还可以使用多版本数据技术保证读写一致性，这种情况下，查询操作可以不必使用锁。

查询操作不使用任何锁的情形有以下三种：

- 隔离级别设置为 read uncommitted。此时不需要保证数据读写一致性。
- 数据库开启 read_committed_snapshot 参数后，在 read committed 隔离级别查询时，不再对表以及表中的数据页、记录附加任何锁，用多版本数据技术保证读写一致性。
- 数据库开启 allow_snapshot_isolation 参数后，可以设置 snapshot 隔离级别，此级别下的查询不会对表附加任何锁，用多版本数据技术保证读写一致性。

详细设置方法请参考上一章相关内容。

16.3 死锁

锁可以解决并发操作中的丢失更新问题，但使用不合适，也会引起死锁问题。

死锁是两个或多个事务同时处于等待状态，每个事务都在等待另一个事务释放对某个资源锁定后才能继续自己的操作。

下表所示操作步骤可以说明死锁的产生过程，图中的 R1 和 R2 分别表示两种需要锁定的资源，加锁时采用排他方式。

表 16-1 模拟死锁产生

时间	事务 A	事务 B
t1	锁定 R1	
		锁定 R2
t2	欲锁定 R2	
t3	等待	欲锁定 R1
t4	等待	等待

下面我们在 Oracle 中模拟死锁的产生过程，假定两个事务需要修改 dept 表的 10 号部门和 20 号部门的地址，很明显，这两行记录相当于上图中需要锁定的资源 R1 和 R2，且锁定方式为排他。

启动两个连接，在其中修改 dept 表中的 10 号和 20 号部门的地址。

在连接 1 中修改 dept 表中 10 号部门的地址：

```
SQL> --conn1
SQL> update dept set loc='BOSTON' where deptno=10;
已更新 1 行。
```

以上操作对 10 号记录附加上了排他锁。

在连接 2 中修改 dept 表中 20 号部门的地址：

```
SQL> --conn2
SQL> update dept set loc='CHICAGO' where deptno=20;
已更新 1 行。
```

以上操作对 20 号记录附加上了排他锁。

回到连接 1，修改 20 号记录：

```
SQL> --conn1
SQL> update dept set loc='NEW YORK' where deptno=20;
```

因为 20 号记录已被连接 2 锁定，连接 1 的本次修改操作发生等待。

回到连接 2，修改 10 号记录：

```
SQL> --conn2
SQL> update dept set loc='DALLAS' where deptno=10;
```

因为 10 号记录已被连接 1 锁定，连接 2 的本次修改操作发生等待。

两个连接都在等待，这时就发生了死锁。

发生死锁后，对资源锁定的等待形成了一个环形结构，要解除死锁，只要破坏掉环形结构中的任意一个环节即可。

Oracle 会自动探测到死锁的发生，并撤销其中一个事务中引起锁的操作，以使其可以执行用户的命令。在此示例中，Oracle 撤销了连接 1 的修改操作，给出如下错误信息：

```
SQL> --conn1
```

```
SQL> update dept set loc='NEW YORK' where deptno=20;
update dept set loc='NEW YORK' where deptno=20
       *
第 1 行出现错误:
ORA-00060: 等待资源时检测到死锁
```

撤销连接 1 的修改操作后，用户可以在连接 1 执行新的命令了。要注意的是，Oracle 检测到死锁后，只是撤销了引起死锁的这个操作，并未回滚整个事务。

在 SQL Server 中执行上述过程也会像 Oracle 一样产生死锁，SQL Server 也会自动探测到死锁的存在，但会把其中一个事务作为牺牲品对其执行回滚操作，而不是只撤销此事务中引起死锁的那个操作。

下面是 SQL Server 探测到死锁时的报错信息：

```
消息 1205，级别 13，状态 51，服务器 LAW_X240，第 1 行
事务(进程 ID 52)与另一个进程被死锁在 锁 资源上，并且已被选作死锁牺牲品。请重新运行该事务。
```

Oracle 的查询操作不使用锁，修改数据时附加行锁，与 SQL Server 相比，一般较少发生死锁。