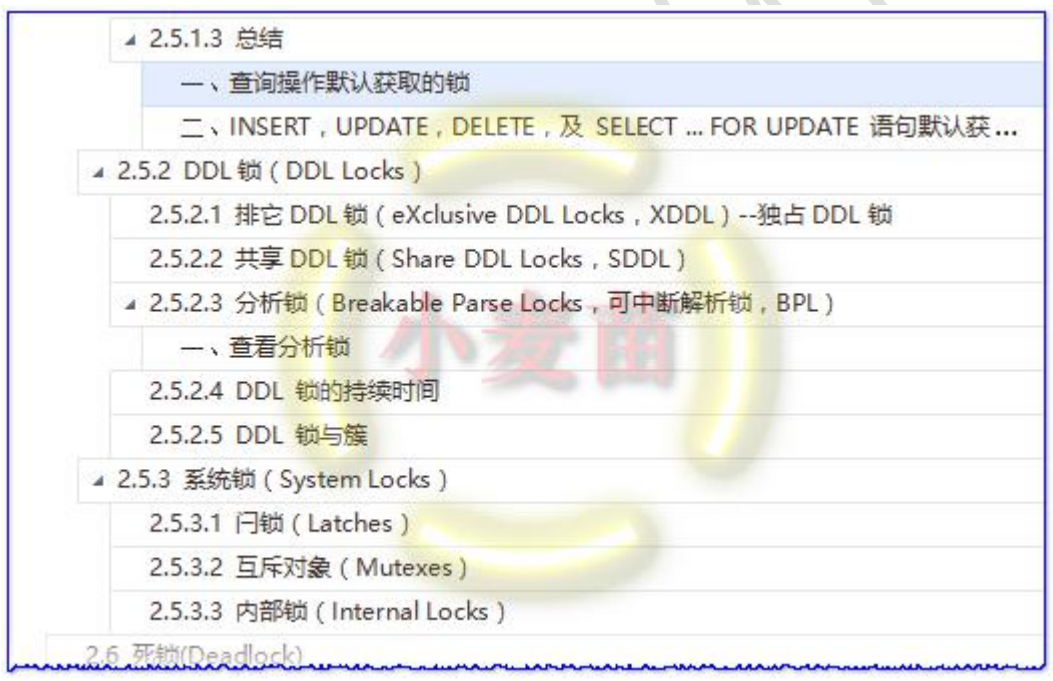
【锁】Oracle 锁系列

1.1 BLOG 文档结构图



1.2 前言部分

1.2.1 导读和注意事项

各位技术爱好者，看完本文后，你可以掌握如下的技能，也可以学到一些其它你所不知道的知识，~O(∩_∩)O~：

① **锁的概念、分类、及其模拟**

② 查询锁的视图及视图之间的关联

③ 锁的参数（DML_LOCKS、DDL_LOCK_TIMEOUT）

④ FOR UPDATE 及 FOR UPDATE OF 系列

⑤ 带 ONLINE 和不带 ONLINE 创建索引的锁情况（是否阻塞 DML 操作）

⑥ 包或存过不能编译的解决方法

⑦ ORA-08104 解决方法

Tips:

① 本文在 itpub（http://blog.itpub.net/26736162）、博客园（http://www.cnblogs.com/lhrbest）和微信公众号（xiaomaimiaolhr）上有同步更新。

② 文章中用到的所有代码、相关软件、相关资料及本文的 pdf 版本都请前往小麦苗的云盘下载，小麦苗的云盘地址见：http://blog.itpub.net/26736162/viewspace-1624453/。

③ 若网页文章代码格式有错乱，请下载 pdf 格式的文档来阅读。

④ 在本篇 BLOG 中，代码输出部分一般放在一行一列的表格中。其中，需要特别关注的地方我都用灰色背景和粉红色字体来表示，比如在下边的例子中，thread 1 的最大归档日志号为 33，thread 2 的最大归档日志号为 43 是需要特别关注的地方；而命令一般使用黄色背景和红色字体标注；对代码或代码输出部分的注释一般采用蓝色字体表示。

```
  List of Archived Logs in backup set 11
  Thrd Seq     Low SCN    Low Time            Next SCN   Next Time
  ---- ------- ---------- ------------------- ---------- ---------
  1    32      1621589    2015-05-29 11:09:52 1625242    2015-05-29 11:15:48
  1    33      1625242    2015-05-29 11:15:48 1625293    2015-05-29 11:15:58
  2    42      1613951    2015-05-29 10:41:18 1625245    2015-05-29 11:15:49
  2    43      1625245    2015-05-29 11:15:49 1625253    2015-05-29 11:15:53
[ZHLHRDB1:root]:/>lsvg -o
T_XLHRD_APP1_vg
rootvg
[ZHLHRDB1:root]:/>
00:27:22 SQL> alter tablespace idxtbs read write;
====》2097152*512/1024/1024/1024=1G
```

本文如有错误或不完善的地方请大家多多指正，ITPUB 留言或 QQ 皆可，您的批评指正是我写作的最大动力。

1.2.2 本文简介

有网友一直催着说发一些锁系列的文章，其实小麦苗一直对锁这块也没有彻底去研究过，今年写书里边写到了锁的内容，干脆就彻底把这一块整理了一下，现在分享给大家，若有错误，还请大家及时指正。

文章很多内容来源于网络或 Concepts 的内容，若有侵权还请联系小麦苗删除。

第 2 章 锁

2.1 锁的基本概念

锁的定义：锁（lock）机制用于管理对共享资源的并发访问，用于多用户的环境下，可以保证数据库的完整性和一致性。锁是防止访问相同资源的事务之间的破坏性交互的机制。既可以是用户对象(例如表或行)，也可以是对用户不可见的系统对象(例如共享数据结构和数据字典行)。

锁的解释：当多个用户并发地存取数据时，在数据库中就会产生多个事务同时存取同一数据的情况。若对并发操作不加控制就可能会读取和存储不正确的数据，破坏数据库的完整性和一致性。当事务在对某个数据对象进行操作前，先向系统发出请求，对其加锁。加锁后事务就对该数据对象有了一定的控制。

锁的作用：在并发事务之间防止破坏性的交互作用，不需要用户的动作，自动使用最低的限制级别，在事务处理期间保持。

数据库是一个多用户使用的共享资源。当多个用户并发地存取数据时，在数据库中就会产生多个事务同时存取同一数据的情况。若对并发操作不加控制就可能会读取和存储不正确的数据，破坏数据库的一致性。

锁（lock）是防止访问相同资源（例如表或数据行等用户对象，或内存中的共享数据结构及数据字典等对用户不可见的系统对象）的事务产生破坏性交互的机制。

在任何情况下，Oracle 都能够自动地获得执行 SQL 语句所必须的所有锁，无需用户干预。Oracle 会尽可能地减少锁产生的影响，从而最大程度地保证数据的并发访问能力，并确保数据一致性及错误恢复。同时，Oracle 也支持用户手工加锁的操作。

Oracle 从来不会升级锁，但是它会执行锁转换（lock conversion）或锁提升（lock promotion）。

A **lock** is a mechanism that prevents **destructive interactions**, which are interactions that incorrectly update data or incorrectly alter underlying data structures, between transactions accessing shared data. Locks play a crucial row in maintaining database concurrency and consistency.

锁是一种机制，用来防止多个共同访问共享数据的事务之间的破坏性交互，包括不正确地更新数据或不正确地更改基础数据结构。锁在维护数据库并发性和一致性当中扮演着一个关键的角色。

2.1.1 并发和并行

并发(concurrency)和并行(parallel)。并发意思是在数据库中有超过两个以上用户对同样的数据做修改，而并行的意思就是将一个任务分成很多小的任务，让每一个小任务同时执行，最后将结果汇总到一起。所以说，锁产生的原因就是并发，并发产生的原因是因为系统和客户的需要。

2.1.2 使用锁

在单用户数据库中，锁不是必需的，因为只有一个用户在修改信息。但是，当多个用户在访问和修改数据时，数据库必须提供一种方法，以防止对同一数据进行并发修改。锁实现了以下重要的数据库需求：

- **一致性**

一个会话正在查看或更改的数据不能被其它会话更改，直到用户会话结束。

- **完整性**

数据和结构必须按正确的顺序反映对他们所做的所有更改。数据库通过其锁定机制，提供在多个事务之间的数据并发性、一致性、和完整性。锁定将自动执行，并且不需要用户操作。

执行 SQL 语句时，Oracle 数据库自动获取所需的锁。例如，在数据库允许某个会话修改数据之前，该会话必须先锁定数据。锁给予该会话对数据的独占控制权，以便在释放该锁之前，任何其它事务都不可以修改被锁定的数据。

因为数据库的锁定机制与事务控制紧密地绑定在一起，应用程序设计人员只需要正确地定义事务，而数据库会自动管理锁定。

2.1.3 锁模式（Lock Modes）--共享和排它

Oracle 数据库自动使用最低适用的限制级别，来提供最高程度的数据并发，但还能提供非常安全的数据完整性。限制级别越低、则有更多的可用数据供其他用户访问。相反，限制级别越高，则其它事务为获取其所需的锁类型就将遭受更多的限制。

在多用户的数据库系统中，Oracle 使用两种模式的锁：

名称	简介	何时使用
共享锁（Share Lock，S 锁，读锁）	S 锁是可以查看但无法修改和删除的一种数据锁。若事务 T 对数据对象 A 加上 S 锁，则事务 T 只能读 A；其它事务只能再对 A 加 S 锁，而不能加 X 锁，直到 T 释放 A 上的 S 锁。这就保证了其它事务可以读 A，但在 T 释放 A 上的 S 锁之前不能对 A 做任何修改。	当执行 SELECT 时，数据库会自动使用 S 锁
排它锁（eXclusive Lock，X 锁，独占锁，写锁，互斥锁）	如果事务 T 对数据 A 加上 X 锁后，则其它事务不能再对 A 加任任何类型的锁。获得 X 锁的事务既能读数据，又能修改数据。	执行 INSERT、UPDATE、DELETE 时数据库会自动使用 X 锁

2.1.4 锁的持续时间

事务内各语句获得的锁在事务执行期内有效，以防止事务间破坏性的相互干扰，例如：脏读取（dirty read），无效地更新（lost update），以及其它并发事务中具有破坏性的 DDL 操作。如果某个事务中的 SQL 语句对数据进行了修改，只有在此事务提交后开始的事务才能看到前者修改的结果。

当用户提交（commit）或撤销（undo）一个事务后，Oracle 将释放此事务内各个 SQL 语句获得的锁。当用户在事务内回滚到某个保存点（savepoint）后，Oracle 也会释放此保存点后获得的锁。只有当前没有等待被锁资源的事务才能获得可用资源的锁。等待事务不会对可用资源加锁而是继续等待，直至拥有其所等待资源的事务完成提交或回滚。

2.2 显式锁定和隐式锁定

有两种类型：显式锁定和隐式锁定。Oracle 锁被自动执行，并且不要求用户干预的锁为隐式锁。对于 SQL 语句隐式锁是必须的，依赖被请求的动作。隐式锁定除 SELECT 外，对所有的 SQL 语句都发生。用户也可以手动锁定数据，这是显式锁定。

隐式锁定：这是 Oracle 中使用最多的锁。通常用户不必声明要对谁加锁，Oracle 自动可以为操作的对象加锁，这就是隐式锁定。

显式锁定：用户可以使用命令明确的要求对某一对象加锁。显式锁定很少使用。

2.2.1 显式锁定

LOCK TABLE 没有触发行锁，只有 TM 表锁。

```
LOCK TABLE TABLE_NAME IN ROW SHARE MODE NOWAIT;   --2：RS
LOCK TABLE TABLE_NAME IN SHARE UPDATE MODE;  --2：RS
LOCK TABLE TABLE_NAME IN ROW EXCLUSIVE MODE NOWAIT; --3：RX
LOCK TABLE TABLE_NAME IN SHARE MODE; --4：S
LOCK TABLE TABLE_NAME IN SHARE ROW EXCLUSIVE MODE;  --5：SRX
LOCK TABLE TABLE_NAME IN EXCLUSIVE MODE NOWAIT; --6：X
```

2.2.2 隐式锁定

隐式锁定：

```
Select * from table_name……
Insert into table_name……
Update table_name……
Delete from table_name……
Select * from table_name for update
```

2.3 悲观锁和乐观锁

名称	描述	应用场景
悲观锁（Pessimistic Lock）	顾名思义，很悲观。每次去读数据的时候，都认为别的事务会修改数据，所以，每次在读数据的时候都会上锁，防止其它事务读取或修改这些数据，这样导致其它事务会被阻塞，直到这个事务结束。	数据更新比较频繁的场合
乐观锁（Optimistic Lock）	顾名思义，很乐观，每次去拿数据的时候都认为别人不会修改，所以，不会上锁，但是在更新的时候会判断在此期间别人有没有去更新这个数据。乐观锁一般通过增加时间戳字段来实现。	数据更新不频繁，查询比较多的场合，这样可以提高吞吐量

2.3.1 悲观锁

锁在用户修改之前就发挥作用：

```
Select ..for update（nowait)
Select * from tab1 for update
```

用户发出这条命令之后，oracle 将会对返回集中的数据建立行级封锁，以防止其他用户的修改。

如果此时其他用户对上面返回结果集的数据进行 dml 或 ddl 操作都会返回一个错误信息或发生阻塞。

1：对返回结果集进行 update 或 delete 操作会发生阻塞。

2：对该表进行 ddl 操作将会报：Ora-00054:resource busy and acquire with nowait specified.

原因分析

此时 Oracle 已经对返回的结果集上加了排它的行级锁，所有其他对这些数据进行的修改或删除操作都必须等待这个锁的释放，产生的外在现象就是其它的操作将发生阻塞，这个这个操作 commit 或 rollback.

同样这个查询的事务将会对该表加表级锁，不允许对该表的任何 ddl 操作，否则将会报出 ora-00054 错误：:resource busy and acquire with nowait specified.

会话 1：

```
SYS@lhrdb S1> create table t_lock_lhr as select rownum as id,0 as type from dual connect by rownum <=3;

Table created.

SYS@lhrdb S1> select * from t_lock_lhr where id=2 and type =0 for update nowait;

       ID      TYPE
---------- ----------
        2         0
```

会话 2：

```
SYS@lhrdb S2> select * from t_lock_lhr where id=2 and type=0 for update nowait;
select * from t_lock_lhr where id=2 and type=0 for update nowait
           *
ERROR at line 1:
ORA-00054: resource busy and acquire with NOWAIT specified or timeout expired
```

会话 1：

```
SYS@lhrdb S1> update t_lock_lhr set type=1 where id=2 and type=0;

1 row updated.

SYS@lhrdb S1> commit;

Commit complete.


SYS@lhrdb S1> select * from t_lock_lhr where id=2;

       ID      TYPE
---------- ----------
        2         1
```

会话 2：

```
SYS@lhrdb S2> select * from t_lock_lhr where id=2 and type=0 for update nowait;

no rows selected
```

2.3.2 乐观锁

乐观的认为数据在 select 出来到 update 进取并提交的这段时间数据不会被更改。这里面有一种潜在的危险就是由于被选出的结果集并没有被锁定，是存在一种可能被其他用户更改的可能。因此 Oracle 仍然建议是用悲观封锁，因为这样会更安全。

会话 1：

```
SYS@lhrdb S1> select id,type,ora_rowscn from t_lock_lhr where id = 3;

       ID       TYPE ORA_ROWSCN
---------- ---------- ----------
        3          0   37698547
```

会话 2：

```
SYS@lhrdb S2> select id,type,ora_rowscn from t_lock_lhr where id = 3;

       ID       TYPE ORA_ROWSCN
---------- ---------- ----------
        3          0   37698547
```

会话 1：

```
SYS@lhrdb S1> update t_lock_lhr set type=1 where ora_rowscn=37698547 and id = 3;

1 row updated.

SYS@lhrdb S1> commit;

Commit complete.
SYS@lhrdb S1> select id,type,ora_rowscn from t_lock_lhr where id = 3;

       ID       TYPE ORA_ROWSCN
---------- ---------- ----------
        3          1   37698591
```

会话 2：

```
SYS@lhrdb S2> update t_lock_lhr set type=1 where ora_rowscn=37698547 and id =3;

0 rows updated.

SYS@lhrdb S2> select id,type,ora_rowscn from t_lock_lhr where id = 3;

       ID       TYPE ORA_ROWSCN
---------- ---------- ----------
        3          1   37698591
```

2.3.3 更新丢失问题的解决方法

更新丢失是指多个用户通过应用程序访问数据库时，由于查询数据并返回到页面和用户修改完毕点击保存按钮将修改后的结果保存到数据库这个时间段（即修改数据在页面上停留的时间）在不同用户之间可能存在偏差，从而最先查询数据并且最后提交数据的用户会把其他用户所作的修改覆盖掉。

解决方法如下：

悲观锁定	保存前怀查询值并验证数据尚未修改，试图更新之前把行锁住，SELECT…FORUPDATE 疑数据被 NOWAIT，然后更新数据。修改过。			
乐观锁定	认为数据不会被其他用户修改，修改屏幕上的信息而不要锁	1.使用版本列的乐观锁定	增加 NUMBER 或 TIMESTAMP 或 DATE 列。每次修改行时 检查数据库中这一列的值与最初读出的值是否匹配 匹配的话修改数据且通过触发器要负责递增 NUMBER，DATE/TIMESTAMP。	增加一个时间戳列，可以知道最后修改时间。
		2.使用校验和的乐观锁定	用基数据本身来计算一个"虚拟的"版本列。生成散列值进行比较。	数据库独立性好，从 CPU 使用和网络传输方面来看，资源开销量大。
		3.使用 ORA_ROWSCN 的乐观锁定	建立在内部 ORACLE 系统时钟（SCN）基础上，建表时，启用 ROWDEPENDENCIES，防止整个数据块的 ORA_ROWSCN 向前推进。可以用 SCN_TO_TIMESTAMP(ORA_ROWSCN) 将 SCN 转换为墙上时钟时间。	将原先的悲观锁机制修改为乐观锁来控制并发，可以使用 ORA_ROWSCN，这样可以无需增加新列。也可以通过 SCN_TO_TIMESTAMP 来获取最后修改时间。

2.4 锁转换和锁升级（Lock Conversion and Escalation）

数据库在必要时执行锁转换。在锁转换中，数据库自动将较低限制的表锁转换为较高限制的其它锁定。一个事务在该事务中所有执行插入、更新、或删除的行上持有行独占锁。因为行锁是在最高程度限制下获得的，因此不要求锁转换，也不执行锁转换。锁转换不同于锁升级，锁升级发生在当某个粒度级别持有许多锁（例如行），数据库将其提高到更高粒度级别（例如表）。如果一个用户锁定了一个表中的许多行，则某些数据库自动将行锁升级到单个表锁。锁的数量减少了，但被锁定的东西却增加了。

Oracle 数据库永远不会升级锁。锁升级极大地增加了死锁的可能性。假定一个系统尝试升级事务 1 中的锁，但因为事务 2 持有该锁，故不能成功。如果事务 2 在它可以继续操作之前也需要在相同的数据上进行锁升级，则将发生一个死锁。

ORACLE 的锁是 block 里面实现的,SQLSERVER,DB2 是内存里面实现的.内存实现有资源消耗问题,当内存不足会引发锁升级,但是 ORACLE 不会发生锁升级。

事务拥有在此事务内被插入（insert），更新（update），删除（delete）的数据行的排它行级锁（exclusive row lock）。对于数据行来说，排它行级锁已经是限制程度最高的锁，因此无需再进行锁转换（lock conversion）。

2.5 锁的分类

Oracle 能够自动地选择不同类型的锁对数据并发访问进行控制，防止用户间破坏性的交互操作。Oracle 将自动地为事务进行锁管理，防止其它事务对需要排它访问的资源执行操作。当事务不再需要加锁的资源并触发某个事件后，锁能够被自动地释放。

在事务执行期间，Oracle 能够根据加锁的资源及需要执行的操作自动地决定锁的类型（types of lock）及对资源的限制级别（level of restrictiveness）。

V$LOCK_TYPE 该视图是对 DML 锁的类型的解释。

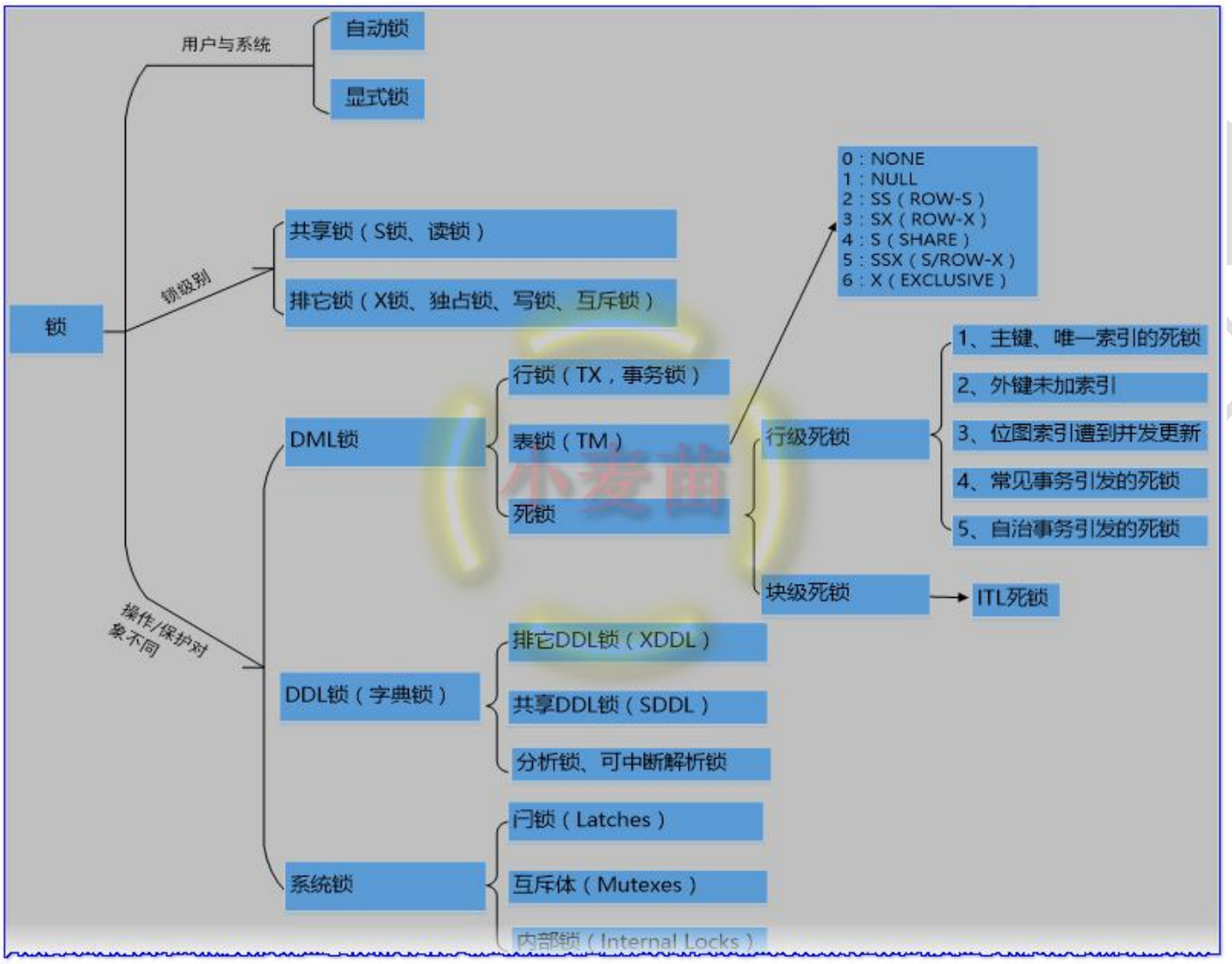
```
select * from V$LOCK_TYPE v where    v.IS_USER='YES';
```

TYPE	NAME	ID1_TAG	ID2_TAG	IS_USER	DESCRIPTION
TM	DML	object #	table/partition	YES	Synchronizes accesses to an object
TX	Transaction	usn<<16 | slot	sequence	YES	Lock held by a transaction to allow other transactions to wait for it
ZH	Compression Analyzer	obj#	ulevel	YES	Synchronizes analysis and insert into compression$, prevents multiple thr
UL	User-defined	id	0	YES	Lock used by user applications

当 Oracle 执行 DML 语句时，系统自动在所要操作的表上申请 TM 类型的锁。当 TM 锁获得后，系统再自动申请 TX 类型的锁，并将实际锁定的数据行的锁标志位进行置位。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 锁包括了 SS、SX、S、X 等多种模式，在数据库中用 0－6 来表示。不同的 SQL 操作产生不同类型的 TM 锁。

在数据行上只有 X 锁（排它锁）。在 Oracle 数据库中，当一个事务首次发起一个 DML 语句时就获得一个 TX 锁，该锁保持到事务被提交或回滚。当两个或多个会话在表的同一条记录上执行 DML 语句时，第一个会话在该条记录上加锁，其它的会话处于等待状态。当第一个会话提交后，TX 锁被释放，其它会话才可以加锁。

当 Oracle 数据库发生 TX 锁等待时，如果不及时处理常常会引起 Oracle 数据库挂起，或导致死锁的发生，产生 ORA-60 的错误。这些现象都会对实际应用产生极大的危害，如长时间未响应，大量事务失败等。



分类依据	锁类型			简介
用户与系统划分	自动锁			当进行一项数据库操作时，缺省情况下，系统自动为此数据库操作获得所有有必要的锁。
	显式锁			某些情况下，需要用户显式的锁定操作所要用到的数据，才能使数据库操作执行得更好，显式锁是用户为数据库对象设定的。
锁级别	共享锁（S锁、读锁）			共享锁使一个事务对特定数据库资源进行共享访问，另一事务也可对此资源进行访问或获得相同共享锁。共享锁为事务提供高并发性，但拙劣的事务设计+共享锁容易造成死锁或数据更新丢失。
	排它锁（X锁、独占锁、写锁、互斥锁）			事务设置排它锁后，该事务单独获得此资源，另一事务不能在此事务提交之前获得相同对象的共享锁或排它锁。
操作/保护的对象不同	DML锁（DML Locks）	（Data Lock，数据锁），保证并发情况下的数据完整性	行锁（Row Locks，TX，Transaction eXclusive，事务锁）	当事务执行INSERT、UPDATE、DELETE、MERGE、或SELECT ... FOR UPDATE 操作时，该事务自动获得操作表中操作行的排它锁。每个事务只能得到一个TX锁。
			表锁（TM，Table Locks，Table dML）	当事务获得行锁后，此事务也将自动获得该行的表锁（共享锁），以防止其它事务进行DDL操作影响记录行的更新。事务也可以在执行过程中获得共享锁或排它锁，只有当事务显示使用LOCK TABLE语句显示的定义一个排它锁时，事务才会获得表上的排它锁，也可使用LOCK TABLE显式的定义一个表级的共享锁。TM锁包括了SS、SX、S、X等7种模式，在数据库中用0-6来表示，不同的SQL操作产生不同类型的TM锁，具体内容如表 2-10 TM锁级别所示，其中数字越大锁级别越高，影响的操作越多。
			死锁	所谓死锁，是指两个或两个以上的进程在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力作用，它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。死锁又分为行级死锁和块级死锁，具体内容如3.2.9 Oracle中的死锁所示。
	DDL锁（DDL Locks）	（Data Dictionary Lock，数据字典锁），用于保护数据库对象的结构，如表、索引等的结构定义	排它DDL锁（eXclusive DDL Locks，XDDL）	也叫独占DDL锁，创建、修改、删除一个数据库对象的DDL语句获得操作对象的排它DDL锁。例如，当使用ALTER TABLE语句时，为了维护数据的完成性、一致性、合法性，该事务将获得排它DDL锁。
			共享DDL锁（Share DDL Locks，SDDL）	需在数据库对象之间建立相互依赖关系的DDL语句通常需共享获得DDL锁。例如，创建一个包，该包中的过程与函数引用了不同的数据库表，当编译此包时，该事务就获得了引用表的共享DDL锁。这些锁会保护所引用对象的结构，使之不会被其它会话修改，但是允许修改数据。
			分析锁（Breakable Parse Locks，可中断解析锁，BPL）	Oracle使用共享池存储分析与优化过的SQL语句及PL/SQL程序，使运行相同语句的应用速度更快。一个在共享池中缓存的对象获得它所引用数据库对象的分析锁。分析锁是一种独特的DDL锁类型，Oracle使用它追踪共享池对象及它所引用数据库对象之间的依赖关系。当一个事务修改或删除了共享池持有分析锁的数据库对象时，Oracle使共享池中的对象作废，当下次在引用这条SQL/PLSQL语句时，Oracle就会重新分析编译此语句。分析锁允许一个对象（如共享池中缓存的一个执行计划）向另外某个对象注册其依赖性。如果在被依赖的对象上执行DDL，那么Oracle会查看已经对该对象注册了依赖性的对象列表，并使这些对象无效。因此，这些锁是“可中断的”。
	系统锁（System Locks）	Oracle数据库使用各种类型的系统锁来保护数据库内部和内存结构。由于用户不能控制其何时发生或持续多久，这些机制对于用户几乎是不可访问的。闩锁、互斥体、和内部锁是完全自动的。	闩锁（Latches）	闩锁是简单、低级别的串行化机制，用于协调对共享数据结构、对象、和文件的多用户访问。闩锁防止共享内存资源被多个进程访问时遭到破坏。具体而言，闩锁在以下情况下保护数据结构： 1、被多个会话同时修改 2、正在被一个会话读取时，又被另一个会话修改 3、正在被访问时，其内存被释放（换出） V$LATCH视图包含每个闩锁的详细使用情况的统计信息，包括每个闩锁被请求和被等待的次数。
			互斥体（Mutexes，mutual exclusion object）	Mutexes是Oracle 11g新增的锁，也叫互斥对象，它是一种底层机制，用于防止在内存中的对象在被多个并发进程访问时，被换出内存或遭到破坏。Mutexes类似于闩锁，但闩锁通常保护一组对象，而互斥对象通常保护单个对象。 Mutexes提供以下几个优点： 1、Mutexes可以减少发生争用的可能性。由于闩锁保护多个对象，当多个进程试图同时访问这些对象的任何一个时，它可能成为一个瓶颈。而互斥体仅仅串行化对单个对象的访问，而不是一组对象，因此Mutexes提高了可用性。 2、Mutexes比闩锁消耗更少的内存。 3、在共享模式下，互斥体允许被多个会话并发引用。
			内部锁（Internal Locks）	内部锁是比闩锁和互斥体更高级、更复杂的机制，并用于各种目的。数据库使用以下类型的内部锁： 1、字典缓存锁（Dictionary cache locks）这些锁的持续时间很短，当字典缓存中的条目正在被修改或使用时被持有。它们保证正在被解析的语句不会看到不一致的对象定义。字典缓存锁可以是共享的或独占的。共享锁在解析完成后被释放，而独占锁在DDL操作完成时释放。 2、文件和日志管理锁（File and log management locks） 这些锁用于保护各种文件。 3、表空间和撤销段锁（Tablespace and undo segment locks）这些锁用于保护的表空间和撤销段。

2.5.1 DML 锁（DML Locks）

当 Oracle 执行 DELETE,UPDATE,INSERT,SELECT FOR UPDATE DML 语句时，oracle 首先自动在所要操作的表上申请 TM 类型的锁。当 TM 锁获得后，再自动申请 TX 类型的锁，并将实际锁定的数据行的锁标志位(lb 即 lock bytes)进行置位。在记录被某一会话锁定后，其它需要访问被锁定对象的会话会按先进先出的方式等待锁的释放，对于 select 操作而言，并不需要任何锁，所以即使记录被锁定，select 语句依然可以执行，实际上，在此情况下，oracle 是用到 undo 的内容进行一致性读来实现的。

当 Oracle 执行 DML 语句时，系统自动在所要操作的表上申请 TM 类型的锁。当 TM 锁获得后，系统再自动申请 TX 类型的锁，并将实际锁定的数据行的锁标志位进行置位。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。DML 语句能够自动地获得所需的表级锁（table-level lock）与行级锁（row-level lock）。

DML 锁，也称为数据锁，确保由多个用户并发访问的数据的完整性。例如，DML 锁可防止两个客户从一个在线书店购买某一本书所剩的最后一个拷贝。DML 锁也可以防止多个相互冲突的 DML 或 DDL 操作产生破坏性干扰。

DML 语句自动获取下列类型的锁：

- 行锁(TX)
- 表锁(TM)

2.5.1.1 行锁（Row Locks，TX）

Waits for Enq TX -Type Events - Transaction (TX) Lock Example Scenarios (文档 ID 62354.1).mhtml

Detecting and Resolving Locking Conflicts and Ora-00060 errors (文档 ID 15476.1)文档 15476.1.mhtml

行级锁（row-level lock）的作用是防止两个事务同时修改相同的数据行。当一个事务需要修改一行数据时，就需对此行数据加锁。Oracle 对语句或事务所能获得的行级锁的数量没有限制，Oracle 也不会讲行级锁的粒度升级（lock escalation）。行级锁是粒度最精细的锁，因此行级锁能够提供最好的数据并发访问能力及数据处理能力。

Oracle 同时支持多版本并发访问控制（multiversion concurrency control）及行级锁技术（row-level locking），因此用户只有在访问相同数据行时才会出现竞争，具体来说：

- 读取操作无需等待对相同数据行的写入操作。
- 写入操作无需等待对相同数据行的读取操作，除非读取操作使用了 SELECT ... FOR UPDATE 语句，此读取语句需要对数据加锁。
- 写入操作只需等待并发地且针对相同数据行的其它写入操作。

提示：读取操作可能会等待对相同数据块（data block）的写入操作，这种情况只会在出现挂起的分布式事务（pending distributed transaction）时偶尔出现。

在执行下列语句时，事务需要获得被修改的每一数据行的排它行级锁（exclusive row lock）：INSERT，UPDATE，DELETE，及使用了 FOR UPDATE 子句的 SELECT 语句。

在事务被提交或回滚前，此事务拥有在其内部被修改的所有数据行的排它锁，其它事务不能对这些数据行进行修改操作。但是，如果事务由于实例故障而终止，在整个事务被恢复前，数据块级的恢复将使数据块内数据行上的锁释放。执行前面提到的 4 种 SQL 语句时，Oracle 能自动地对行级锁进行管理。

当事务获得了某些数据行上的行级锁时，此事务同时获得了数据行所属表上的表级锁（table lock）。表级锁能够防止系统中并发地执行有冲突的 DDL 操作，避免当前事务中的数据操作被并发地 DDL 操作影响。

行级锁机制：

当一个事务开始时，必须申请一个 TX 锁，这种锁保护的资源是回滚段、回滚数据块。因此申请也就意味着：用户进程必须先申请到回滚段资源后才开始一个事务，才能执行 DML 操作。申请到回滚段后，用户事务就可以修改数据了。具体顺序如下：

1、首先获得 TM 锁，保护事务执行时，其他用户不能修改表结构

2、事务修改某个数据块中记录时，该数据块头部的 ITL 表中申请一个空闲表项，在其中记录事务项号，实际就是记录这个事务要使用的回滚段的地址（应该叫包含）

3、事务修改数据块中的某条记录时，会设置记录头部的 ITL 索引指向上一步申请到的表项。然后修改记录。修改前先在回滚段将记录之前的状态做一个拷贝，然后修改表中数据。

4、其他用户并发修改这条记录时，会根据记录头部 ITL 索引读取 ITL 表项内容，确认是否事务提交。

5、若没有提交，必须等待 TX 锁释放

从上面的机制来看，无论一个事务修改多少条记录，都只需要一个 TX 锁。所谓的“行级锁”其实也就是数据块头、数据记录头的一些字段，不会消耗额外的资源。 从另一方面也证明了，当用户被阻塞时，不是被某条记录阻塞，而是被 TX 锁堵塞。也正因为这点，很多人也倾向把 TX 锁称为事务锁。这里可通过实验来验证所说 结论。

会话 1：

```
SQL> select * from test;
       ID NAME
---------- --------
        1 A
        2 B
        3 C

SQL> savepoint a;
Savepoint created.

SQL> update test set name='ssss' where id=2;
1 row updated.
```

会话 2，更新同一行发生阻塞：

```
SQL> update test set name='ssdsdsds'where id=2;
```

会话 1：

```
SQL> rollback to a;
Rollback complete.
```

可以看到，虽然会话 1 已经撤销了对记录的修改，但是会话 2 仍然处于等待状态这是因为会话 2 是被会话 1 的 TX 锁阻塞的，而不是被会话 1 上的行级锁 阻塞（rollback to savepoint 不会结束事务）。

```
    会话 3：
SQL> select username,event,sid,blocking_session from v$session where SID IN (146,159);
USERNAME EVENT                               SID BLOCKING_SESSION
-------- ----------------------------------- ---------- ----------------
HR       enq: TX - row lock contention           146           159
HR       SQL*Net message from client             159
    会话 1：
SQL> rollback;
    会话 2：
SQL> update test set name='ssdsdsds'where id=2;
 1 row updated.
    会话 3：
SQL> select username,event,sid,blocking_session from v$session where username='HR';
USERNAME EVENT                               SID BLOCKING_SESSION
-------- ----------------------------------- ---------- ----------------
HR       SQL*Net message from client             159
```

事务结束，tx 锁释放，会话 2update 执行成功。

行锁，也称为 TX 锁，是一个表中单个行上的锁。一个事务在被 INSERT、UPDATE、DELETE、MERGE、或 SELECT ... FOR UPDATE 等语句所修改的每一行上获取一个行锁。行锁一直存在直到事务提交或回滚。行锁主要作为一种排队的机制，以防止两个事务修改相同的行。数据库始终以独占模式锁定修改的行，以便其它事务不能修改该行，直到持有锁的事务提交或回滚。行锁定提供了近乎最细粒度的锁定，并因此提供了近乎最佳的并发性和吞吐量。

如果一个事务因为数据库实例失效而终止，会先进行块级恢复以使行可用，之后进行整个事务恢复。

2.5.1.2 表锁（Table Locks，TM）

表级锁（table-level lock）的作用是对并发的 DDL 操作进行访问控制，例如防止在 DML 语句执行期间相关的表被移除。当用户对表执行 DDL 或 DML 操作时，将获取一个此表的表级锁。表级锁不会影响其他并发的 DML 操作。对于分区表来说，表级锁既可以针对整个表，也可以只针对某个分区。

当用户执行以下 DML 语句对表进行修改：INSERT，UPDATE，DELETE，及 SELECT ... FOR UPDATE，或执行 LOCK TABLE 语句时，事务将获取一个表级锁。这些 DML 语句获取表级锁的目的有两个：首先保证自身对表的访问不受其它事务 DML 语句的干扰，其次阻止其它事务中和自身有冲突的 DDL 操作执行。任何类型的表级锁都将阻止对此表的排它 DDL 锁（exclusive DDL lock），从而阻止了必须具备排它 DDL 锁才能执行的 DDL 操作。例如，当一个未提交的事务拥有某个表上的锁时，此表就无法被修改定义或被移除。

表级锁具有以下几种模式：行共享（row share，RS），行排它（row exclusive，RX），共享（share，S），共享行排它（share row exclusive，SRX），及排它（exclusive，X）。各种模式的表级锁具有的限制级别决定了其是否能与其他表级锁共处于同一数据表上。

下表显示了各种语句所获得的表级锁的模式，以及此模式下被允许或禁止的操作。

ORACLE 里锁有以下几种模式：

			是否允许锁操作？				
SQL 语句	行级锁模式	表级锁模式	RS（2）	RX（3）	S（4）	SRX（5）	X（6）
SELECT...FROM table...		NULL	Y	Y	Y	Y	Y
INSERT INTO table ...	X	RX	Y	Y	N	N	N
INSERT /*+APPEND*/ INTO table ...	X	X	N	N	N	N	N
UPDATE table ...	X	RX	Y*	Y*	N	N	N
DELETE FROM table ...	X	RX	Y*	Y*	N	N	N
SELECT ... FROM table FOR UPDATE (OF) ...	X	RX(Oracle 9i是RS)	Y*	Y*	Y*	Y*	N
LOCK TABLE table IN ROW SHARE MODE		RS	Y	Y	Y	Y	N
LOCK TABLE table IN SHARE UPDATE MODE		RS	Y	Y	Y	Y	N
LOCK TABLE table IN ROW EXCLUSIVE MODE		RX	Y	Y	N	N	N
LOCK TABLE table IN SHARE MODE		S	Y	N	Y	N	N
LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE		SRX	Y	N	N	N	N
LOCK TABLE table IN EXCLUSIVE MODE		X	N	N	N	N	N
注：Y*表示当不与其它事务的行级锁冲突时才允许，否则将产生等待。							

锁的兼容模式如下表所示：

	Held/Get	Null（1）	RS（2）	RX（3）	S（4）	SSX（5）	X（6）
0、1	none、Null	√	√	√	√	√	√
2	RS	√	√	√	√	√	
3	RX	√	√	√			
4	S	√	√		√		
5	SSX	√	√				
6	X	√					

表锁，也称为 TM 锁，当一个表被 INSERT、UPDATE、DELETE、MERGE、带 FOR UPDATE 子句的 SELECT 等修改时，由相关事务获取该锁。DML 操作需要表锁来为事务保护 DML 对表的访问，并防止可能与事务冲突的 DDL 操作。

表锁可能以下列模式之一持有：

锁模式	锁描述	锁别名	SQL语句举例	详解	允许的操作	禁止的操作
0	none	没有锁				
1	NULL	空	SELECT	NULL锁是一种分析锁，是系统自动生成的。有NULL锁的对象，一旦被删除，它会通知有该表NULL锁的会话，这个表被删除了。在某些情况下，如分布式数据库的查询会产生此锁。一些内部操作时会在某些阶段自动获得1号锁。		
2	SS（Sub-Share）、RS（Row Share，Row-S）	行共享表级锁（Row Share Table Lock，RS）或行级共享锁也被称为子共享表锁（SS，Subshare Table Lock）	LOCK TABLE ... IN SHARE UPDATE MODE; LOCK TABLE ... IN ROW SHARE MODE; CREATE/ALTER INDEX ... ONLINE;--Oracle 11g	SS锁在表级别只和X锁不兼容，和其它的锁都是兼容的。SS锁表明拥有此锁的事务已经锁定了表内的某些数据行，并有意对数据行进行更新操作。行共享锁是限制最少的表级锁模式，提供在表上最高程度的并发性。	某个事务拥有了某个表的RS锁后，其它事务依然可以并发地对相同数据表执行查询、插入、更新和删除操作，或对表内数据行执行加锁的操作。也就是说，其它事务同时也能获得相同表上的RS、RX、S和SSX模式的表级锁。	某个事务拥有了某个表的RS锁后，只会禁止其它事务对相同表获取X锁。
3	SX（Sub-Exclusive）、RX（Row Exclusive，Row-X）	行级排它锁（行独占表锁，Row Exclusive Table Lock）也被称为子独占表锁（SX，subexclusive table lock）	INSERT、UPDATE、DELETE、MERGE INTO SELECT ... FOR UPDATE SELECT ... FOR UPDATE OF column LOCK TABLE ... IN ROW EXCLUSIVE MODE; 注：Oracle 10g之前FOR UPDATE是SS锁	RX比RS的限制程度略高。RX锁表明拥有此锁的事务已经对表内的某些数据行进行了更新操作。当对话使用SELECT ... FOR UPDATE子串打开一个游标时，所有返回结果集中的数据行都将处于行级（Row-X）独占式锁定，其它对象只能查询这些数据行，不能进行UPDATE、DELETE或SELECT ... FOR UPDATE操作，但是可以执行INSERT的操作。在没有COMMIT之前其它会话更新（UPDATE、DELETE）相同记录会没有反应，因为后一个模式为3的锁会一直等待上一个模式为3的锁，此时必须释放掉上一个锁才能继续工作。	某个事务拥有了某个表的RX锁后，其它事务依然可以并发地对相同数据表执行查询、插入、更新和删除操作。RX锁允许其它多个事务同时获得相同表上的RS或RX锁。	某个事务拥有了某个表的RX锁后，将禁止其它事务对表加S、SSX和X锁。
4	S（Share）	共享表锁（Share Table Lock，S）	CREATE INDEX ALTER INDEX CREATE/ALTER INDEX ... ONLINE;--Oracle 10g LOCK TABLE ... IN SHARE MODE	不带ONLINE的新建或重建索引的SQL语句获取的是4级TM锁。在Oracle 10g中，带ONLINE的新建或重建索引的SQL语句在开始和结束的时候获取的是4级TM锁，而在读取表数据的过程中获取的是2级TM锁。在Oracle 11g中，带ONLINE的新建或重建索引的SQL语句在整个执行过程中获取的是2级TM锁。S锁和S、RS锁都兼容，和其它3种带X的锁模式（SX、SSX、X）都不兼容。	某个事务拥有了某个表的S锁后，其它事务可以查询表，也能够成功执行LOCK TABLE ... IN SHARE MODE语句，但其它事务不能对表进行执行INSERT、UPDATE和DELETE操作。多个事务可以并发地获得同一个表上的S锁。因此，拥有S锁的事务只有在此表上没有其它事务的S锁时，才能对表进行更新操作。	某个事务拥有了某个表的S锁后，将禁止其它事务修改此表，同时禁止其它事务获得3、5和6级锁。
5	SSX（Share Sub-Exclusive）、SRX（Share Row Exclusive，S/Row-X）	共享行级排它锁，也被称为共享行独占表锁（Share Row Exclusive Table Lock，SRX或共享子独占表锁	LOCK TABLE ... IN SHARE ROW EXCLUSIVE MODE	SSX比S锁的限制性更强，一次只能有一个事务可以获取给定的表上的SSX锁，SSX只和2级锁RX是兼容的。	同一时间只有一个事物能够获得表的SSX锁。若某个事务拥有了某个表的SSX锁后，则其它事务可以查询表，但不能对表进行更新操作。	拥有SSX锁的事务将阻止其它事务获取SX锁来修改数据。SSX锁还能阻止其它事务在相同表上获取S、SSX和X锁。
6	X（Exclusive）	排它锁或独占表锁（Exclusive Table Lock，X）	ALTER TABLE、DROP TABLE、DROP INDEX、TRUNCATE TABLE 、LOCK TABLE ... IN EXCLUSIVE、INSERT	这种锁是最严格的锁，禁止其它事务执行任何类型的DML语句，或在表上放置任何类型的锁。X锁是限制程度最高的表级锁，它能使获得此锁的事务排它地对表进行写操作。	同一时间只有一个事务能获得表上的X锁。在一个事务获得X锁后，其它事务只能对表进行查询操作。	一个事务获得排它表级锁后，将禁止其它事务对表执行任何 DML操作，其它事务也无法获取表上任何类型的锁。

一、 行共享(RS) Row Share (RS)

这种锁也被称为子共享表锁(SS，**subshare table lock**)，表示在表上持有锁的事务在表中有被锁定的行，并打算更新它们。行共享锁是限制最少的表级锁模式，提供在表上最高程度的并发性。

1、 实验

ROW SHARE 模式允许同时访问被锁定的表，但是禁止用户以排它方式锁定整个表。ROW SHARE 与 SHARE UPDATE 相同，只是为了兼容早期的 Oracle 版本。对应 lmode2，row-S （SS）。

版本：11.2.0.4

会话 1：

```
SYS@lhrdb> set sqlprompt "_user'@'_connect_identifier S1> "
SYS@lhrdb S1> select userenv('sid') from dual;

USERENV('SID')
--------------
           6

SYS@lhrdb S1> LOCK TABLE SCOTT.EMP IN ROW SHARE MODE;

Table(s) Locked.
```

会话 2：

```
SYS@lhrdb> set sqlprompt "_user'@'_connect_identifier S2> "
SYS@lhrdb S2> select userenv('sid') from dual;

USERENV('SID')
--------------
           114

SYS@lhrdb S2> LOCK TABLE SCOTT.EMP IN EXCLUSIVE MODE;



====>>>>> 产生了阻塞
```

查询 2 个会话的锁：

```
SYS@lhrdb S1> SELECT D.SID, D.TYPE, D.ID1, D.ID2, D.LMODE, D.REQUEST, D.CTIME, D.BLOCK
  2    FROM V$LOCK D
  3   WHERE D.SID IN (114, 6)
```

```
  4   ORDER BY D.SID, D.TYPE;
       SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
---------- -- ---------- ---------- ---------- ---------- ---------- ----------
         6 AE        100          0          4          0        231          0
         6 TM      86893          0          2          0        169          1
       114 AE        100          0          4          0        378          0
       114 TM      86893          0          0          6        144          0
       114 TO      79619          1          3          0        376          0
```

```
SELECT D.SID, D.TYPE, D.ID1, D.ID2, D.LMODE, D.REQUEST, D.CTIME, D.BLOCK
  FROM V$LOCK D
 WHERE D.SID IN (114, 6)
ORDER BY D.SID, D.TYPE;
```

	SID	TYPE	ID1	ID2	LMODE	REQUEST	CTIME	BLOCK
1	6	AE	100	0	4	0	100	0
2	6	TM	86893	0	2	0	38	1
3	114	AE	100	0	4	0	247	0
4	114	TM	86893	0	0	6	13	0
5	114	TO	79619	1	3	0	245	0

由 BLOCK 列可以看到 sid 为 6 的会话阻塞了一个会话，这里其实就是 114，而 114 正在请求模式为 6 的锁。将 2 个会话提交后继续测试：

```
SYS@lhrdb S1> LOCK TABLE SCOTT.EMP IN SHARE UPDATE MODE;

Table(s) Locked.

SYS@lhrdb S1> SELECT D.SID, D.TYPE, D.ID1, D.ID2, D.LMODE, D.REQUEST, D.CTIME, D.BLOCK
  2    FROM V$LOCK D
  3   WHERE D.SID IN (114, 6)
  4     AND D.TYPE = 'TM'
  5   ORDER BY D.SID, D.TYPE;

       SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
---------- -- ---------- ---------- ---------- ---------- ---------- ----------
         6 TM      86893          0          2          0        387          0
```

二、 行独占表锁 Row Exclusive Table Lock (RX)

这种锁也被称为子独占表锁（SX，**subexclusive table lock**），通常表示持有锁的事务已更新了表行或发出了 SELECT...FOR UPDATE。一个 SX 锁允许其它事务并发地查询、插入、更新、删除、或锁定在同一个表中的其它行。因此，SX 锁允许多个事务对同一个表同时获得 SX 和子共享表锁。

ROW EXCLUSIE 类似于 ROW SHARE 模式，但是不能应用在 SHARE 模式中。当 update,insert,delete 发生时，ROW EXCLUSIVE 会自动获得。对应 lmode3，row-X (SX) 。

1、 实验

实验内容：but it also prohibits locking in SHARE mode

会话 1：

```
SQL> set sqlprompt "_user'@'_connect_identifier S1> "
SYS@oratest S1>  select distinct sid from v$mystat;

      SID
----------
       21
SYS@oratest S1> lock table scott.emp in share mode;

Table(s) Locked.
```

会话 2：

```
SQL> set sqlprompt "_user'@'_connect_identifier S2> "
SYS@oratest S2>  select distinct sid from v$mystat;

      SID
----------
      142

SYS@oratest S2>  lock table scott.emp in  row exclusive mode;
```

====>>>>> 产生了阻塞

查看锁：

```
SYS@oratest S1> set line 9999
SYS@oratest S1> select * from v$lock where sid in (21,142);

ADDR             KADDR                 SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
---------------- ---------------- ---------- -- ---------- ---------- ---------- ---------- ---------- ----------
00000000774D8518 00000000774D8570        142 TO       68064          1          3          0       7021          0
00000000774D9870 00000000774D98C8        142 TO       76985          1          3          0       7365          0
00000000774D9DC8 00000000774D9E20         21 AE         100          0          4          0        162          0
00000000774DA068 00000000774DA0C0        142 AE         100          0          4          0       7379          0
00007F567ADC2700 00007F567ADC2760        142 TM       75335          0          0          3         36          0
00007F567ADC2700 00007F567ADC2760         21 TM       75335          0          4          0         58          1

6 rows selected.

SYS@oratest S1> select * from v$lock where sid in (21,142) AND TYPE IN ('TX','TM');

ADDR             KADDR                 SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
---------------- ---------------- ---------- -- ---------- ---------- ---------- ---------- ---------- ----------
00007F567ADC7818 00007F567ADC7878        142 TM       75335          0          0          3         76          0
00007F567ADC7818 00007F567ADC7878         21 TM       75335          0          4          0         98          1

SYS@oratest S1> SELECT * FROM DBA_DML_LOCKS;

SESSION_ID OWNER                          NAME                          MODE_HELD     MODE_REQUESTE LAST_CONVERT BLOCKING_OTHERS
---------- ------------------------------ ----------------------------- ------------- ------------- ------------ ----------------------------------------
       142 SCOTT                          EMP                           None          Row-X (SX)             101 Not Blocking
        21 SCOTT                          EMP                           Share         None                   123 Blocking

SYS@oratest S1>
```

这里可以看到会话 1 的 TM4 阻塞了会话 2 的 TM3。

提交 2 个会话后，接着实验：ROW EXCLUSIVE locks are automatically obtained when updating, inserting, or deleting.

```
SYS@oratest S1>  update scott.emp set sal=sal where empno=7369;

1 row updated.

SYS@oratest S1> select * from v$lock where sid in (21,142) AND TYPE IN ('TX','TM');

ADDR             KADDR                 SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
---------------- ---------------- ---------- -- ---------- ---------- ---------- ---------- ---------- ----------
00007F567ADE6AC8 00007F567ADE6B28         21 TM       75335          0          3          0          4          0
0000000076227AB0 0000000076227B28         21 TX      196620       1097          6          0          4          0
```

当会话 1 做了修改而没有 commit 或者 rollback 时，这里有两个锁，其中一个就是 TM3 的，一个是 TX6 的。

三、 共享表锁 Share Table Lock (S)

由某个事务拥有的共享表锁允许其它事务查询（而不使用 SELECT...FOR UPDATE），但是更新操作只能在仅有单个事务持有共享表锁时才允许。因为可能有多个事务同时持有共享表锁，所以持有此锁不足以确保一个事务可以修改该表。

SHARE 允许同时查询，但是禁止更新被锁定的表。对应 lmode4，share （S） 。

1、 实验

会话 1：

```
SQL> set sqlprompt "_user'@'_connect_identifier S1> "
SYS@oratest S1>  select distinct sid from v$mystat;

      SID
----------
       21
SYS@oratest S1> lock table scott.emp in share mode;

Table(s) Locked.
```

会话 2：

```
SQL> set sqlprompt "_user'@'_connect_identifier S2> "
SYS@oratest S2>  select distinct sid from v$mystat;

      SID
----------
      142

SYS@oratest S2>  update scott.emp set sal=sal where empno=7369;
```

====>>>>> 产生了阻塞

查看锁：

```
SYS@oratest S1> select * from v$lock where sid in (21,142) AND TYPE IN ('TX','TM');

ADDR             KADDR                 SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
---------------- ---------------- ---------- -- ---------- ---------- ---------- ---------- ---------- ----------
00007F567ADE6AC8 00007F567ADE6B28        142 TM      75335          0          0          3         43          0
00007F567ADE6AC8 00007F567ADE6B28         21 TM      75335          0          4          0         62          1


SYS@oratest S1> SELECT * FROM DBA_DML_LOCKS;

SESSION_ID OWNER    NAME   MODE_HELD     MODE_REQUESTE LAST_CONVERT BLOCKING_OTHERS
---------- -------- ------ ------------- ------------- ------------ ---------------
       142 SCOTT    EMP    None          Row-X (SX)             113 Not Blocking
        21 SCOTT    EMP    Share         None                   132 Blocking
SYS@oratest S1>
```

这里可以看到会话 1 的 TM4 阻塞了会话 2 的 TM3。

四、 共享行独占表锁 Share Row Exclusive Table Lock (SRX)

这种锁也称为共享子独占表锁（SSX，**share-subexclusive table lock**），比共享表锁的限制性更强。一次只能有一个事务可以获取给定的表上的 SSX 锁。由某个事务拥有的 SSX 锁允许其它事务查询该表（除 SELECT...FOR UPDATE）但不能更新该表。

共享行级排它锁有时也称共享子排它锁（Share Subexclusive Table Lock，SSX），它比共享锁有更多限制。定义共享行级排它锁的语法为：

```
Lock Table TableName In Share Row Exclusive Mode;
```

1、 实验

会话 1：

```
SQL> set sqlprompt "_user'@'_connect_identifier S1> "
SYS@oratest S1>  select distinct sid from v$mystat;

      SID
----------
       21
SYS@oratest S1> lock table scott.emp  in  share row exclusive  mode;

Table(s) Locked.
```

会话 2：

```
SQL> set sqlprompt "_user'@'_connect_identifier S2> "
SYS@oratest S2>  select distinct sid from v$mystat;

      SID
----------
      142

SYS@oratest S2>  lock table scott.emp  in  share mode;
```

```
====>>>>> 产生了阻塞
```

查看锁：

```
SYS@oratest S1> select * from v$lock where sid in (21,142) AND TYPE IN ('TX','TM');

ADDR             KADDR                 SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
---------------- ---------------- ---------- -- ---------- ---------- ---------- ---------- ---------- ----------
00007F567ADE7B00 00007F567ADE7B60        142 TM      75335          0          0          4         21          0
00007F567ADE7B00 00007F567ADE7B60         21 TM      75335          0          5          0         69          1
SYS@oratest S1> SELECT * FROM DBA_DML_LOCKS;

SESSION_ID OWNER   NAME   MODE_HELD     MODE_REQUESTE LAST_CONVERT BLOCKING_OTHERS
---------- ------- ------ ------------- ------------- ------------ ---------------
       142 SCOTT   EMP    None          Share                   44 Not Blocking
        21 SCOTT   EMP    S/Row-X (SSX) None                    92 Blocking
```

这里可以看到会话 1 的 TM5 阻塞了会话 2 的 TM4。

五、 独占表锁 Exclusive Table Lock (X)

这种锁是最严格的锁，禁止其它事务执行任何类型的 DML 语句，或在表上放置任何类型的锁。

EXCLUSIVE EXCLUSIVE permits queries on the locked table but prohibits any other activity on it.

EXCLUSIVE 模式允许查询被锁表上的数据，但是禁止任何其他任何活动（这里我理解是禁止添加其他任何模式的锁）。对应 lomde6，exclusive (X) 。

1、 实验

会话 1：

```
SQL> set sqlprompt "_user'@'_connect_identifier S1> "
SYS@oratest S1>  select distinct sid from v$mystat;

      SID
----------
       21
SYS@oratest S1> CREATE TABLE SCOTT.EMP_01 AS SELECT * FROM SCOTT.EMP;

Table created.

SYS@oratest S1>  update scott.emp_01 set sal=sal where empno=7369;

1 row updated.
```

会话 2：

```
SQL> set sqlprompt "_user'@'_connect_identifier S2> "
SYS@oratest S2>  select distinct sid from v$mystat;

      SID
----------
      142

SYS@oratest S2>  DELETE FROM scott.emp_01  where empno=7369;



====>>>>> 产生了阻塞
```

查看锁：

```
SYS@oratest S1> select * from v$lock where sid in (21,142) AND TYPE IN ('TX','TM');

ADDR             KADDR                 SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
---------------- ---------------- ---------- -- ---------- ---------- ---------- ---------- ---------- ----------
00000000774D9EA8 00000000774D9F00        142 TX     393247       1337          0          6         28          0
00007F567ABBC0A0 00007F567ABBC100        142 TM      77624          0          3          0         28          0
00007F567ABBC0A0 00007F567ABBC100         21 TM      77624          0          3          0         36          0
0000000076255548 00000000762555C0         21 TX     393247       1337          6          0         36          1

SYS@oratest S1> SELECT * FROM DBA_DML_LOCKS;

SESSION_ID OWNER    NAME     MODE_HELD     MODE_REQUESTE LAST_CONVERT BLOCKING_OTHERS
---------- -------- -------- ------------- ------------- ------------ ---------------
       142 SCOTT    EMP_01   Row-X (SX)    None                    35 Not Blocking
        21 SCOTT    EMP_01   Row-X (SX)    None                    43 Not Blocking
```

在这里，从 BLOCK 字段可以看到会话 1 的 TM3 并没堵塞会话 2 的 TM3，这里真正发生堵塞的是会话 1 的 TX6。

这里还有一个锁定对象的问题。上面两个 TM3 的锁针对的对象是 object_id 为 77624 的表，既然描述是类似行共享，自然是不会堵塞的。而两个 TX6 的锁针对的对象可以理解成表中的行，在这些行上添加 EXCLUSIVE 锁（lmode6，exclusive (X) ）自然是会堵塞其他的 EXCLUSIVE 锁的。

解决这种类型的锁堵塞当然就是在代码中尽早 commit 结束事务。很多地方都写到尽早 commit 可以提高运行效率，这里所指的是释放锁（特别是 lmode6 的

EXCLUSIVE 锁）减少堵塞，以提高并发性。（不是以减少数据的量来提高效率的，事实上不管多大的数据量，一个 commit 的过程都是很"平"的。

2、 INSERT /*+APPEND*/ INTO 加 6 级 TM 和 TX 独占锁

会话 1：

```
SYS@lhrdb> set sqlprompt "_user'@'_connect_identifier S1> "
SYS@lhrdb S1> SELECT DISTINCT SID FROM  V$MYSTAT;

      SID
----------
       27
SYS@lhrdb S1> CREATE TABLE T_APPEND_161107_LHR AS SELECT * FROM DUAL;

Table created.

SYS@lhrdb S1> INSERT /*+ APPEND */ INTO T_APPEND_161107_LHR SELECT * FROM DUAL;

3 rows created.
```

会话 2：

```
SYS@lhrdb> set sqlprompt "_user'@'_connect_identifier S2> "
SYS@lhrdb S2>  SELECT DISTINCT SID FROM  V$MYSTAT;

      SID
----------
      162

SYS@lhrdb S2> INSERT /*+ APPEND */ INTO T_APPEND_161107_LHR SELECT * FROM DUAL;


  <<<<<<<<<-------- 产生了阻塞
```

```
SYS@lhrdb S2> INSERT /*+ APPEND */ INTO T_APPEND_161107_LHR SELECT * FROM DUAL;
```

会话 3：

```
SYS@lhrdb> set sqlprompt "_user'@'_connect_identifier S3> "
SYS@lhrdb S3> set line 9999
SYS@lhrdb S3> SELECT * FROM V$LOCK T WHERE T.SID IN (27,162) AND T.TYPE IN ('TX','TM') ORDER BY T.SID ;

ADDR             KADDR                SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
---------------- ---------------- ---------- -- ---------- ---------- ---------- ---------- ---------- ----------
00000001109F5A40 00000001109F5AA0         27 TM     100957          0          6          0       2217          1
070001007C7EB2B0 070001007C7EB328         27 TX     589843      58249          6          0       2217          0
00000001109F5A40 00000001109F5AA0        162 TM     100957          0          0          6       2214          0

====>>>>> 过了很久
SYS@lhrdb S3> SELECT * FROM V$LOCK T WHERE T.SID IN (27,162) AND T.TYPE IN ('TX','TM') ORDER BY T.SID ;

ADDR             KADDR                SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
---------------- ---------------- ---------- -- ---------- ---------- ---------- ---------- ---------- ----------
00000001109F6A78 00000001109F6AD8         27 TM     100957          0          6          0       2882          1
070001007C7EB2B0 070001007C7EB328         27 TX     589843      58249          6          0       2882          0
00000001109F6A78 00000001109F6AD8        162 TM     100957          0          0          6       2879          0

SYS@lhrdb S3> /

ADDR             KADDR                SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
---------------- ---------------- ---------- -- ---------- ---------- ---------- ---------- ---------- ----------
00000001109F5A40 00000001109F5AA0         27 TM     100957          0          6          0       2885          1
070001007C7EB2B0 070001007C7EB328         27 TX     589843      58249          6          0       2885          0
00000001109F5A40 00000001109F5AA0        162 TM     100957          0          0          6       2882          0
```

```
SYS@lhrdb S3> set line 9999
SYS@lhrdb S3> SELECT * FROM V$LOCK T WHERE T.SID IN (27,162) AND T.TYPE IN ('TX','TM') ORDER BY T.SID ;

ADDR             KADDR                SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
---------------- ---------------- ---------- -- ---------- ---------- ---------- ---------- ---------- ----------
00000001109F5A40 00000001109F5AA0         27 TM     100957          0          6          0       2217          1
070001007C7EB2B0 070001007C7EB328         27 TX     589843      58249          6          0       2217          0
00000001109F5A40 00000001109F5AA0        162 TM     100957          0          0          6       2214          0
```

其中，会话 1 的 sid 为 27，分别在 TX 和 TM 级别，拥有 LMODE 为 6 的 X 锁。BLOCK 为 1 说明会话 1 阻塞了其它会话（0 表示没有阻塞，2 表示 RAC 环境需要用 GV$LOCK）。CTIME 表示拥有此锁的时间，单位为秒。会话 2 的 sid 为 162，REQUEST 为 6 表示正在请求模式为 6 的锁。

当 TYPE 列为 TM 的时候，即对于 TM 锁来说，ID1 列表示被锁定的对象的对象 ID，ID2 始终为 0，如下：

```
SYS@lhrdb S3> COL OWNER FORMAT A5
SYS@lhrdb S3> COL OBJECT_NAME FORMAT A20
SYS@lhrdb S3> SELECT D.OWNER,D.OBJECT_NAME,D.OBJECT_ID FROM DBA_OBJECTS D WHERE D.OBJECT_ID = 100957;
OWNER OBJECT_NAME          OBJECT_ID
```

```
----- -------------------- ----------
SYS   T_APPEND_161107_LHR      100957
```

当 TYPE 列为 TX 的时候，即对于 TX 锁来说，ID1 列表示事务使用的回滚段编号以及在事务表中对应的记录编号，ID2 表示该记录编号被重用的次数（wrap），ID1 列表示事务的信息，如下：

```
SYS@lhrdb S3> SELECT A.TADDR FROM V$SESSION A WHERE SID = 27;

TADDR
----------------
070001007C7EB2B0

SYS@lhrdb S3> SELECT A.XIDUSN, A.XIDSLOT, A.XIDSQN
  2    FROM V$TRANSACTION A
  3   WHERE A.ADDR = '070001007C7EB2B0';

    XIDUSN    XIDSLOT     XIDSQN
---------- ---------- ----------
         9         19      58249

SYS@lhrdb S3> SELECT TRUNC(589843 / POWER(2, 16)) AS UNDO_SEG#,
  2         BITAND(589843, TO_NUMBER('ffff', 'xxxx')) + 0 AS SLOT#,
  3                  58249 XIDSQN
  4    FROM DUAL;

 UNDO_SEG#      SLOT#     XIDSQN
---------- ---------- ----------
         9         19      58249

SYS@lhrdb S3> SELECT SID,
  2         STATUS,
  3         SQL_ID,
  4         LAST_CALL_ET,
  5         BLOCKING_INSTANCE,
  6         BLOCKING_SESSION,
  7         EVENT
  8    FROM GV$SESSION
  9   WHERE BLOCKING_SESSION IS NOT NULL;

       SID STATUS   SQL_ID        LAST_CALL_ET BLOCKING_INSTANCE BLOCKING_SESSION EVENT
---------- -------- ------------- ------------ ----------------- ---------------- --------------------
       162 ACTIVE   2kvrfkkjukryr         4875                 1               27 enq: TM - contention

SYS@lhrdb S3> select sql_text from v$sql where sql_id='2kvrfkkjukryr';

SQL_TEXT
--------------------------------------------------
INSERT /*+ APPEND */ INTO T_APPEND_161107_LHR SELECT * FROM DUAL

SYS@lhrdb S3> SELECT * FROM DBA_DML_LOCKS D WHERE D.SESSION_ID IN (27, 162);

SESSION_ID OWNER NAME                 MODE_HELD     MODE_REQUESTE LAST_CONVERT BLOCKING_OTHERS
---------- ----- -------------------- ------------- ------------- ------------ ---------------
        27 SYS   T_APPEND_161107_LHR   Exclusive     None                   647 Blocking
       162 SYS   T_APPEND_161107_LHR   None          Exclusive              468 Not Blocking
```

```
SYS@lhrdb S3> SELECT * FROM DBA_DML_LOCKS D WHERE D.SESSION_ID IN (27, 162);

SESSION_ID OWNER NAME                           MODE_HELD     MODE_REQUESTE LAST_CONVERT BLOCKING_OTHERS
---------- ----- ------------------------------ ------------- ------------- ------------ ---------------
        27 SYS   T_APPEND_161107_LHR            Exclusive     None                   647 Blocking
       162 SYS   T_APPEND_161107_LHR            None          Exclusive              468 Not Blocking
```

从视图 DBA_DML_LOCKS 可以非常直观的看出锁的情况，会话 1 即 SID 为 27，拥有 Exclusive 的排它锁，没有请求其它锁，而会话 2 即 SID 为 162 正在请求 Exclusive 的排它锁。

```
SELECT * FROM V$EVENT_NAME WHERE NAME = 'enq: TM - contention';
```

EVENT#	EVENT_ID	NAME	PARAMETER1	PARAMETER2	PARAMETER3	WAIT_CLASS_ID	WAIT_CLASS#	WAIT_CLASS
235	668627480	enq: TM - contention	name|mode	object #	table/partition	4217450380	1	Application

从会话查询锁的信息：

```
SELECT SID,
       STATUS,
       SQL_ID,
       LAST_CALL_ET,
       EVENT,
       A.P1,
       A.P2,
       A.P3,
       CHR(BITAND(P1, -16777216) / 16777215) ||
       CHR(BITAND(P1, 16711680) / 65535) "LOCK",
       BITAND(P1, 65535) "MODE",
       (SELECT OBJECT_NAME FROM DBA_OBJECTS D WHERE D.OBJECT_ID = A.P2) OBJECT_NAME
  FROM GV$SESSION A
 WHERE A.EVENT = 'enq: TM - contention';
```

	SID	STATUS	SQL_ID	LAST_CALL_ET	EVENT	P1	P2	P3	LOCK	MODE	OBJECT_NAME
1	162	ACTIVE	2kvrfkkjukryr	198	enq: TM - contention	1414332422	100957	0	TM	6	T_APPEND_161107_LHR

会话 1 提交，查看会话 2 的情况：

```
SYS@lhrdb S1> commit;

Commit complete.

SYS@lhrdb S1>
```

会话 2：

```
SYS@lhrdb S2> INSERT /*+ APPEND */ INTO T_APPEND_161107_LHR SELECT * FROM DUAL;



3 rows created.

SYS@lhrdb S2> SYS@lhrdb S2> SYS@lhrdb S2> commit;

Commit complete.

SYS@lhrdb S2> SELECT * FROM V$LOCK T WHERE T.SID IN (27,162) AND T.TYPE IN ('TX','TM') ORDER BY T.SID ;

no rows selected
```

2.5.1.3 总结

执行不同的 DML 语句时，Oracle 自动地对数据加锁。

一、 查询操作默认获取的锁

执行查询（query）的 SQL 语句不易与其他 SQL 语句冲突，因为查询只需读取数据。除了 SELECT 之外，INSERT，UPDATE，及 DELETE 语句中也可能包含隐式的查询。因此，以下语句都属于查询操作：

SELECT

INSERT ... SELECT ... ;

UPDATE ... ;

DELETE ... ;

但是以下语句不属于查询操作：

SELECT ... FOR UPDATE OF ... ;

查询操作具备以下特性：

- 查询无需获取数据锁。因此当某事务查询数据表时，其它事务可以并发地查询、更新同一个表，包括此表中正在被查询的数据行。没有使用 FOR UPDATE 子句的 SELECT 语句无需获取任何数据锁，因此也不会阻塞任何操作，此类查询在 Oracle 中被称为非阻塞查询（nonblocking query）。
- 执行查询也不受数据锁的限制。（在某些特殊情况下，查询需要等待挂起的分布式事务所拥有的数据锁）

二、 INSERT，UPDATE，DELETE，及 SELECT ... FOR UPDATE 语句默认获取的锁

INSERT，UPDATE，DELETE，及 SELECT ... FOR UPDATE 语句默认获取的锁有以下特点：

- 包含 DML 语句的事务需要获得被其修改的数据行上的排它行级锁（exclusive row lock）。在拥有锁的事务提交或回滚前，其它事务不能更新或删除被加锁的数据行。
- 事务无需获取 DML 语句内的子查询（subquery）或隐式查询（implicit query）（例如 WHERE 子句内的查询）所选择的行上的行级锁。DML 内的子查询或隐式查询获得的数据相对查询开始的时间点满足一致性，这些查询不会看到 DML 语句自身对数据的影响。
- 事务内的查询能够看到本事务内之前执行的 DML 语句对数据的修改，但无法看到本事务开始后执行的其它事务对数据的修改。
- 事务内的 DML 语句除了需要获得必要的排它行级锁（exclusive row lock）外，至少还需获得包含被修改数据行的表上的行排它表级锁（row exclusive table lock）。如果事务已经获得了相关表上的共享表级锁（share），共享行排它表级锁（share row exclusive），或排它表级锁（exclusive），那么就无需获取行排它表级锁了。如果事务已经获得了相关表上的行共享表级锁（row share table lock），Oracle 将自动地将此锁转换为行排它表级锁。

2.5.2 DDL 锁（DDL Locks）

当某个运行中的 DDL 操作正在操作或引用某模式对象时，数据字典(DDL)锁保护该模式对象的定义。在 DDL 操作的过程中，只有被修改或引用的单个模式的对象被锁定。数据库绝不会锁定整个数据字典。

Oracle 数据库将为任何要求锁的 DDL 事务自动获取 DDL 锁。用户不能显式请求 DDL 锁。例如，如果用户创建一个存储过程，则数据库自动为过程定义中引用的所有模式对象获取 DDL 锁。DDL 锁防止在过程编译完成之前，这些对象被更改或删除。

数据字典锁（data dictionary lock，DDL）的作用是在执行 DDL 操作时对被修改的方案对象或其引用对象的定义进行保护。管理员及开发者应该意识到 DDL 语句将会隐式地提交一个事务。例如，用户创建一个存储过程时，相当于执行一个只包含一条 SQL 语句的事务，Oracle 会自动获取过程定义中所引用的所有方案对象的 DDL 锁。DDL 锁能够防止编译期间过程所引用的对象被其它事务修改或移除。

当 DDL 事务需要时 Oracle 将自动地为其获取数据字典锁。用户不能显示地获取 DDL 锁。只有在 DDL 操作中被修改或引用的对象才会被加锁，整个数据字典不会被加锁。

当用户发布 DDL（Data Definition Language）语句时会对涉及的对象加 DDL 锁。由于 DDL 语句会更改数据字典，所以该锁也被称为字典锁。

DDL 锁能防止在用 DML 语句操作数据库表时，对表进行删除，或对表的结构进行更改。

对于 DDL 锁，要注意的是：

- DDL 锁只锁定 DDL 操作所涉及的对象，而不会锁定数据字典中的所有对象。
- DDL 锁由 Oracle 自动加锁和释放。不能显式地给对象加 DDL 锁，即没有加 DDL 锁的语句。
- 在过程中引用的对象，在过程编译结束之前不能被改变或删除，即不能被加排它 DDL 锁。

DDL 锁可以分为三类：排它 Ddl 锁（Exclusive DDL Lock），共享 Ddl 锁（Share DDL Lock），及可中断的解析锁（Breakable Parse Lock）。

2.5.2.1 排它 DDL 锁（eXclusive DDL Locks，XDDL）--独占 DDL 锁

大多数 DDL 都带有一个排它 DDL 锁。如果发出如下一条语句：

```
Alter table t add new_column date;
```

在执行这条语句时，表 T 不能被别人修改。在此期间，可以使用 SELECT 查询这个表，但是大多数其他操作都不允许执行，包括所有 DDL 语句。

独占 DDL 锁可防止其它会话获取 DDL 或 DML 锁。除了那些在"共享 DDL 锁"中所述操作之外，绝大多数 DDL 操作需要对资源获取独占锁，以防止和其它可能会修改或引用相同模式对象的 DDL 之间的破坏性干扰。例如，当 ALTER TABLE 正在将一列添加到表时，不允许 DROP TABLE 删除表，反之亦然。

独占 DDL 锁在整个 DDL 语句执行期间一直持续，并自动提交。在独占 DDL 锁获取过程中，如果另一个操作在该模式对象上持有另一个 DDL 锁，则这个锁获取将一直等待，直到前一个 DDL 锁被释放，才能继续。

2.5.2.2 共享 DDL 锁（Share DDL Locks，SDDL）

```
create index t_idx on t(x) ONLINE;
```

ONLINE 关键字会改变具体建立索引的方法。Oracle 并不是加一个排它 DDL 锁 防止数据修改，而只会试图得到表上的一个低级 （mode 2 ）TM 锁。这会有效地防止其他 DDL 发生，同时还允许 DML 正常进行。Oracle 执行这一壮举”的做法是，为 DDL 语句执行期 间对表所做的修改维护一个记录，执行 CREATE 时再把这些修改应用至新的索引。这样能大大增加数据的可用性。

另外一类 DDL 会获得共享 DDL 锁。在创建存储的编译对象（如过程和视图）时，会对依赖的对象加这种共享 DDL 锁。例如，如果 执行以下语句：

```
Create view MyView      as          select *          from emp, dept          where emp.deptno = dept.deptno;
```

表 EMP 和 DEPT 上都会加共享 DDL 锁，而 CREATE VIEW 命令仍在处理。可以修改这些表的内容，但是不能修改它们的结构。

A **share DDL lock** for a resource prevents destructive interference with conflicting DDL operations, but allows data concurrency for similar DDL operations.

在资源上的共享 DDL 锁可防止与冲突的 DDL 操作发生破坏性干扰，但允许类似的 DDL 操作的数据并发。

例如，当 CREATE PROCEDURE 语句运行时，所在事务将为所有被引用的表获取共享 DDL 锁。其它事务可以同时创建引用相同表的过程，并在相同的表上同时获得共享 DDL 锁，但没有任何事务能在任何被引用表上获取独占 DDL 锁。

共享 DDL 锁在整个 DDL 语句执行期间持续存在，并自动提交。因此，持有一个共享 DDL 锁的事务，可保证在事务过程中，被引用模式对象的定义保持不变。

某些 DDL 操作需要获取相关资源上的共享 DDL 锁（share DDL lock）以防止与之冲突的 DDL 操作造成破坏性的干扰，但与之类似的 DDL 操作可以并发地访问数据，不受共享 DDL 锁的限制。例如，执行 CREATE PROCEDURE 语句时，事务将获取所有引用对象上的共享 DDL 锁。此时，其它事务可以并发地获取相同表上的共享 DDL 锁并创建引用了相同表的过程。但任何事务都无法获取被引用表上的排它 DDL 锁（exclusive DDL lock），即任何事务都无法对表进行修改或移除操作。因此获得了共享 DDL 锁的事务能够保证在其执行期间，所有引用对象的定义不会被修改。

执行以下 DDL 语句时，需要获取引用对象上的共享 DDL 锁：AUDIT，NOAUDIT，COMMENT，CREATE [OR REPLACE] VIEW/ PROCEDURE/PACKAGE/PACKAGE BODY/FUNCTION/ TRIGGER，CREATE SYNONYM，及 CREATE TABLE（没有使用 CLUSTER 参数时）。

2.5.2.3 分析锁（Breakable Parse Locks，可中断解析锁，BPL）

SQL 语句或 PL/SQL 程序单元，为每个被其引用的模式对象持有一个解析锁。获取解析锁的目的是，如果被引用的对象被更改或删除，可以使相关联的共享 SQL 区无效。解析锁被称为可中断的解析锁，因为它并不禁止任何 DDL 操作，并可以被打破以允许冲突的 DDL 操作。

解析锁是在执行 SQL 语句的分析阶段，在共享池中获取的。只要该语句的共享 SQL 区仍保留在共享池中，该锁就一直被持有。

位于共享池（shared pool）内的 SQL 语句（或 PL/SQL 程序结构）拥有其引用的所有方案对象上的解析锁（parse lock）。解析锁的作用是，当共享 SQL 区（shared SQL area）所引用的对象被修改或移除后，此共享 SQL 区能够被置为无效。解析锁不会禁止任何 DDL 操作，当出现与解析锁冲突的 DDL 操作时，解析锁将被解除，因此也称之为可解除的解析锁。

解析锁是在 SQL 语句执行的解析阶段（parse phase）获得的，在共享 SQL 区被清除出共享池（shared pool）前一直保持。

你的会话解析一条语句时，对于该语句引用的每一个对象都会加一个解析锁。加这些锁的目的是：如果以某种方式删除或修改了一个被引用的对象，可以将共享池中已解析的缓存语句置为无效（刷新输出）。

一、 查看分析锁

```
CREATE OR REPLACE PROCEDURE P_BPL_LHR AS
BEGIN
  NULL;
END;
```

要看到一个实际的可中断解析锁，下面先创建并运行存储过程 P_BPL_LHR：

```
SYS@lhrdb> CREATE OR REPLACE PROCEDURE P_BPL_LHR AS
  2  BEGIN
  3    NULL;
  4  END;
  5  /
```

```
Procedure created.

SYS@lhrdb> exec P_BPL_LHR;

PL/SQL procedure successfully completed.

SYS@lhrdb> SELECT DISTINCT SID FROM  V$MYSTAT;

      SID
----------
      194
```

过程 P_BPL_LHR 现在会出现在 DBA_DDL_LOCKS 视图中。我们有这个过程的一个解析锁：

```
SELECT * FROM DBA_DDL_LOCKS D WHERE D.SESSION_ID = 194;
```

	SESSION_ID	OWNER	NAME	TYPE	MODE_HELD	MODE_REQUESTED
1	194	SYS	DBMS_STANDARD	Table/Procedure/Type	Null	None
2	194	SYS	P_BPL_LHR	Table/Procedure/Type	Null	None
3	194	SYS	DATABASE	18	Null	None

然后重新编译这个过程，并再次查询视图：

```
SYS@lhrdb> ALTER PROCEDURE P_BPL_LHR COMPILE;

Procedure altered.
```

	SESSION_ID	OWNER	NAME	TYPE	MODE_HELD	MODE_REQUESTED
1	194	SYS	DBMS_STANDARD	Table/Procedure/Type	Null	None
2	194	SYS	DATABASE	18	Null	None

可以看到，现在这个视图中没有 P_BPL_LHR 了。我们的解析锁被中断了。这个视图对 发人员很有用，发现测试或开发系统中某段代码无法编译时，将会挂起并最终超时。这说明，有人正在使用这段代码 （实际上在运行这段代码），你可以使用这个视图 查看这个人是谁。对于 GRANTS 和对象的其他类型的 DDL 也是一样。例如，无法对正在运行的过程授予 EXECUTE 权限。可以使用同样的方法 发现潜在的阻塞者和等待者。

2.5.2.4 DDL 锁的持续时间

DDL 锁的持续时间取决于其类型。共享 DDL 锁（share DDL lock）及排它 DDL 锁（exclusive DDL lock）在 DDL 语句执行期间一直存在，在 DDL 语句自动提交后释放。而解析锁一直存在，直至相关的共享 SQL 区从共享池中被清除。

2.5.2.5 DDL 锁与簇

对簇（cluster）执行的 DDL 操作需要获取簇及簇内所有表及物化视图上的排它 DDL 锁（exclusive DDL lock）。对簇内表及物化视图的 DDL 操作需要获取簇上的共享 DDL 锁（share DDL lock），以及表或物化视图上的共享 DDL 锁或排它 DDL 锁。簇上的共享 DDL 锁能够防止操作期间其他 DDL 操作将簇移除。

2.5.3 系统锁（System Locks）

Oracle 数据库使用各种类型的系统锁，来保护数据库内部和内存结构。由于用户不能控制其何时发生或持续多久，这些机制对于用户几乎是不可访问的。闩锁、互斥体、和内部锁是完全自动的。

2.5.3.1 闩锁（Latches）

闩锁（latche）是一种简单的底层串行化机制，用于保护 SGA 内的共享数据结构。例如，用于记录当前正在访问数据库的用户的列表，或用于记录位于数据库缓存（buffer cache）内的数据块的数据结构，都可通过闩锁进行保护。当服务进程（background process）或后台进程（server process）需要操作或查询此类数据结构时，就需要获取一个闩锁，但其加锁时间极短。闩锁的实现与操作系统有关，例如进程是否需要等待栓锁以及等待多长时间等。

闩锁是简单、低级别的串行化机制，用于协调对共享数据结构、对象、和文件的多用户访问。闩锁防止共享内存资源被多个进程访问时遭到破坏。具体而言，闩锁在以下情况下保护数据结构：

- 被多个会话同时修改
- 正在被一个会话读取时，又被另一个会话修改
- 正在被访问时，其内存被释放（换出）

通常，一个单一的闩锁保护 SGA 中的多个对象。例如，后台进程（如 DBWn 和 LGWR）从共享池分配内存来创建数据结构。为分配此内存，这些进程使用共享池闩锁来串行化对内存的访问，以防止两个进程同时尝试检查或修改共享池。内存分配后，其它进程可能需要访问共享池区域，如用于解析所需的库高速缓存。在这种情况下，进程只在库缓存获取闩锁，而不是在整个共享池。

与行锁之类的入队闩锁不同，闩锁不允许会话排队。当闩锁可用时，请求闩锁的第一个会话将获得它的独占访问权限。闩锁旋转（Latch spinning）发生在当一个进程不断地循环来请求一个闩锁时，而闩锁睡眠（latch sleeping）发生在重新发起闩锁请求之前，释放 CPU 时。

通常，一个 Oracle 进程在操作或查看一种数据结构时，只需在一个极短的时间内获得闩锁。例如，仅仅为某一名员工处理工资更新，数据库就可能需要获取并释放成千上万个闩锁。闩锁的实现依赖于操作系统，特别是在一个进程是否会在闩锁上等待以及会在闩锁等待多长时间方面。

闩锁的增加意味着并发的降低。例如，过度硬解析操作会产生库缓存闩锁争用。**V$LATCH** 视图包含每个闩锁的详细使用情况的统计信息，包括每个闩锁被请求和被等待的次数。

2.5.3.2 互斥对象（Mutexes）

互斥对象（mutual exclusion object，mutex），也叫互斥体，它是一种底层机制，用于防止在内存中的对象在被多个并发进程访问时，被换出内存或遭到破坏。互斥对象类似于闩锁，但闩锁通常保护一组对象，而互斥对象通常保护单个对象。

互斥对象提供以下几个优点：

1、 互斥体可以减少发生争用的可能性。

由于闩锁保护多个对象，当多个进程试图同时访问这些对象的任何一个时，它可能成为一个瓶颈。而互斥体仅仅串行化对单个对象的访问，而不是一组对象，因此互斥体提高了可用性。

2、 互斥体比闩锁消耗更少的内存。

3、 **在共享模式下，互斥体允许被多个会话并发引用。**

2.5.3.3 内部锁（Internal Locks）

内部锁是比闩锁和互斥体更高级、更复杂的机制，并用于各种目的。数据库使用以下类型的内部锁：

1、 字典缓存锁（Dictionary cache locks）

这些锁的持续时间很短，当字典缓存中的条目正在被修改或使用时被持有。它们保证正在被解析的语句不会看到不一致的对象定义。字典缓存锁可以是共享的或独占的。共享锁在解析完成后被释放，而独占锁在 DDL 操作完成时释放。

当用户更新或使用时数据字典缓存内的条目（entry）时，需要获取条目上的数据字典缓存锁（dictionary cache lock），此类锁的持续时间极短。此类锁的作用是确保正在被解析的语句不会看到不一致的对象定义。数据字典缓存锁可以为共享或排它的。当语句解析结束时共享锁将被释放，而当 DDL 操作结束时排它锁将被释放。

2、 文件和日志管理锁（File and log management locks）

这些锁保护各种文件。例如，一种内部锁保护控制文件，以便一次只有一个进程可以对其进行更改。而另一种锁用于协调联机重做日志文件的使用和归档。数据文件被锁定，确保数据库被多个实例以共享模式装载，或以独占模式被单个实例装载。因为文件和日志锁表示文件的状态，这些锁必要时会被持有较长一段时间。

此类内部锁（internal lock）用于保护各种文件。例如，保护控制文件（control file）的锁，确保同一时间只有一个进程能够对其进行修改。还有协调重做日志文件（redo log file）使用与归档的锁。以及数据文件（datafile）锁，实现多实例在共享模式下挂载数据库，或一个实例在排它模式下挂载数据库。由于文件及重做日志锁反映的是 物理文件的状态，因此此类锁的持续时间较长。

3、 表空间和撤销段锁（Tablespace and undo segment locks）

这些锁保护的表空间和撤销段。例如，访问数据库的所有实例对一个表空间是否处于联机或脱机必须保持一致。撤销段被锁定，以便只能有一个数据库实例可以写入该段。

此类锁用于保护表空间及回滚段（rollback segment）。例如，一个表空间处于联机（online）还是脱机（offline）状态对访问同一数据库的所有实例应该是一致的。回滚段上的锁保证 同一时间只有一个实例能够对其执行写操作。

2.6 死锁(Deadlock)

有关死锁的内容之前发布过一次，具体内容参考：http://blog.itpub.net/26736162/viewspace-2127247/，本篇文章不再讲解。

2.7 数据字典

常用的数据字典视图有 DBA_DML_LOCKS、DBA_DDL_LOCKS、V$LOCK、DBA_LOCK、V$LOCKED_OBJECT。
---查询的都是当前实例的锁

```
select * from dba_dml_locks;
select * from dba_ddl_locks d where d.owner not in('SYS','WMSYS','MDSYS');

select * from DBA_LOCK V where V.session_id=23;
select * from V$LOCK V where V.SID=23;
select * from V$LOCK_TYPE;
select * from V$LOCKED_OBJECT;
```

2.7.1 V$LOCK 和 dba_lock、dba_locks

本视图列出 Oracle 服务器当前拥有的锁以及未完成的锁或栓锁请求。

列	数据类型	说明	英文解释
ADDR	RAW(4)	锁状态对象的地址	Address of lock state object
KADDR	RAW(4)	锁地址，可以和V$SESSION.LOCKWAIT进行关联	Address of lock
SID	NUMBER	拥有或获得此锁的会话的标识符，对应于V$SESSION.SID列	Identifier for session holding or acquiring the lock
TYPE	VARCHAR2(2)	锁的类型,分为system lock和user lock，部分内容参考，B-4和B-5表，详细内容参考视图V$LOCK_TYPE	Type of user or system lock The locks on the user types are obtained by user applications. Any process that is blocking others is likely to be holding one of these locks. The user type locks are: TM - DML enqueue TX - Transaction enqueue UL - User supplied The system type locks are listed in Table 8-1. Be aware that not all types of locks are documented. To find a complete list of locks for the current release, query the V$LOCK_TYPE data dictionary view, described on "V$LOCK_TYPE".
ID1	NUMBER	锁标识符#1（依赖于类型）,当为TM锁时，该值为DBA_OBJECTS.OBJECT_ID；当为TX锁时，ID1对应视图V$TRANSACTION中的XIDUSN字段（Undo segment number：事务对应的撤销段序列号）和XIDSLOT字段（Slot number：事务对应的槽位号）。其中ID1的高16位为XIDUSN，低16位为XIDSLOT。计算公式为：SELECT TRUNC(ID1/POWER(2,16)) AS XIDUSN,BITAND(ID1,TO_NUMBER('FFFF','XXXX')) + 0 AS XIDSLOT , ID2 XIDSQN FROM DUAL;	Lock identifier #1 (depends on type)
ID2	NUMBER	锁标识符#2（依赖于类型），当为TM锁时，该值为0；当为TX锁时，ID2对应视图V$TRANSACTION中的XIDSQN字段（Sequence number：事务对应的序列号）。	Lock identifier #2 (depends on type)
LMODE	NUMBER	会话拥有此锁的锁模式： 0 ， 没有 1 ， 空（NULL） 2 ， 行子共享模式（SS） 3 ， 行共享互斥模式（SX） 4 ， 共享模式（S ） 5 ， 行子共享互斥模式 6 ， 互斥模式（X ）	Lock mode in which the session holds the lock: •0 - none •1 - null (NULL) •2 - row-S (SS) •3 - row-X (SX) •4 - share (S) •5 - S/Row-X (SSX) •6 - exclusive (X)
REQUEST	NUMBER	会话拥有此锁的锁模式： 0 ， 没有 1 ， 空（NULL） 2 ， 行子共享模式（SS） 3 ， 行共享互斥模式（SX） 4 ， 共享模式（S ） 5 ， 行子共享互斥模式 6 ， 互斥模式（X ）	Lock mode in which the session holds the lock: •0 - none •1 - null (NULL) •2 - row-S (SS) •3 - row-X (SX) •4 - share (S) •5 - S/Row-X (SSX) •6 - exclusive (X)
CTME	NUMBER	授予当前模式以来的时间	Time since current mode was granted
BLOCK	NUMBER	0表示没有阻塞其它锁，1表示阻塞了其它会话，2表示该锁是全局锁，应该查找GV$LOCK	A value of either 0、2 or 1, depending on whether or not the lock in question is the blocker. 0, 'Not Blocking', /* Not blocking any other processes */ 1, 'Blocking', /* This lock blocks other processes */ 2, 'Global', /* This lock is global, so we can't tell */

2.7.1.1 三者关系

v$lock和dba_locks和 dba_lock 内容一样,dba_locks是dba_lock的同义词。可以用动态性能视图的定义来查看它们的关系**V$FIXED_VIEW_DEFINITION。**

```
SELECT * FROM Dba_Objects d WHERE d.object_name LIKE '%DBA_LOCK%' ;
SELECT * FROM Dba_Synonyms d WHERE d.synonym_name LIKE '%DBA_LOCK%' ;
SELECT * FROM V$FIXED_VIEW_DEFINITION d WHERE d.VIEW_NAME LIKE '%V$LOCK%' ;
```

2.7.2 V$LOCKED_OBJECT

注意：V$LOCKED_OBJECT 记录的是 DML 锁信息，DDL 锁的信息不在里面。

这个视图列出系统上的每个事务处理所获得的所有锁。　　记录了当前已经被锁定的对象的信息

XIDUSN 表示当前事务使用的回滚段的编号

XIDSLOT 说明该事务在回滚段头部的事务表中对应的记录编号

XIDSQN 说明序列号

OBJECT_ID 说明当前被锁定的对象的 ID 号，可以根据该 ID 号到 dba_objects 里查找被锁定的对象名称

LOCKED_MODE 说明锁定模式的数字编码

列	数据类型	说明	英文注释
XIDUSN	NUMBER	撤消的段号，回滚段号，可以和v$transaction关联	Undo segment number
XIDSLOT	NUMBER	位置号，槽号	Slot number
XIDSQN	NUMBER	序列号	Sequence number
OBJECT_ID	NUMBER	被锁定的对象ID，可以和dba_objects关联	Object ID being locked
SESSION_ID	NUMBER	持有锁的sessionID，可以和v$session关联	Session ID
ORACLE_USERNAME	VARCHAR2(30)	持有锁的Oracle用户名	Oracle user name
OS_USER_NAME	VARCHAR2(15)	持有锁的操作系统用户名	OS user name
PROCESS	VARCHAR2(9)	OS进程号，可以和v$process关联	OS process ID
LOCKED_MODE	NUMBER	锁模式，含义同v$lock.lmode	Lock mode

V$LOCKED_OBJECT 中的列说明：

示例：1.以DBA角色查看当前数据库里锁的情况可以用如下SQL语句：

```
SELECT v.object_id,
       d.OBJECT_NAME,
       d.OBJECT_TYPE,
       locked_mode,
       v2.username,
       v2.sid,
       v2.serial#,
       v2.logon_time
  FROM   v$locked_object v,
         dba_objects     d,
         v$session       v2
  WHERE  v.OBJECT_ID = d.OBJECT_ID
  AND    v.SESSION_ID = v2.SID
  ORDER  BY v2.logon_time;
```

v$locked_object 视图列出当前系统中哪些对象正被锁定.
v$lock 视图列出当前系统持有的或正在申请的所有锁的情况.

2.7.3 DBA_DDL_LOCKS

DBA_DDL_LOCKS lists all DDL locks held in the database and all outstanding requests for a DDL lock.

DBA_DDL_LOCKS			
DBA_DDL_LOCKS lists all DDL locks held in the database and all outstanding			
Column	Datatype	NULL	Description
SESSION_ID	NUMBER		Session identifier
OWNER	VARCHAR2(30)		Owner of the lock
NAME	VARCHAR2(30)		Name of the lock
TYPE	VARCHAR2(40)		Lock type: Cursor Table/Procedure/Type Body Trigger Index Cluster Java Source Java Resource Java Data
MODE_HELD	VARCHAR2(9)		Lock mode: None Null Share Exclusive
MODE_REQUESTED	VARCHAR2(9)		Lock request type: None Null Share Exclusive

查询所有 DDL 锁的信息：

```
SELECT * FROM DBA_DDL_LOCKS D WHERE D.SESSION_ID = 115;
```

	SESSION_ID	OWNER	NAME	TYPE	MODE_HELD	MODE_REQUESTED
1	115	SYS	STANDARD	Table/Procedure/Type	Null	None
2	115		SYS	73	Share	None
3	115	SYS	DBMS_OUTPUT	Body	Null	None
4	115	SYS	DBMS_OUTPUT	Table/Procedure/Type	Null	None
5	115	SYS	SYSEVENT	Table/Procedure/Type	Null	None
6	115	SYS	PRO_TRI_DDL_INSET_LHR	Table/Procedure/Type	Null	None
7	115	SYS	DBMS_SESSION	Table/Procedure/Type	Null	None
8	115	SYS	DBMS_APPLICATION_INFO	Body	Null	None
9	115	SYS	STANDARD	Body	Null	None
10	115	SYS	DBMS_STANDARD	Table/Procedure/Type	Null	None
11	115	SYS	DBMS_SESSION	Body	Null	None
12	115	SYS	DBMS_APPLICATION_INFO	Table/Procedure/Type	Null	None
13	115	SYS	DBMS_TRANSACTION	Body	Null	None
14	115	SYS	DBMS_TRANSACTION	Table/Procedure/Type	Null	None
15	115	SYS	DATABASE	18	Null	None

如果提示没有这个视图，可以在 sys 用户下执行$ORACLE_HOME/rdbms/admin/catblock.sql 脚本进行创建（这个脚本还包含其他一些非常有意义的锁相关视图）

```
sys@ora10g> conn / as sysdba
Connected.
sys@ora10g> @?/rdbms/admin/catblock.sql
```

这里省略创建过程

打印一下 catblock.sql 脚本的内容，这个创建脚本其实可以当做一个参考文档来用，尤其是其中关于锁类型的描述。

catblock.sql

2.7.4 DBA_DML_LOCKS

DBA_DML_LOCKS lists all DML locks held in the database and all outstanding requests for a DML lock.

Column	Datatype	NULL	Description
SESSION_ID	NUMBER		Session holding or acquiring the lock
OWNER	VARCHAR2(30)	NOT NULL	Owner of the lock
NAME	VARCHAR2(30)	NOT NULL	Name of the lock
MODE_HELD	VARCHAR2(13)		The type of lock held. The values are: ROWS_S (SS): row share lock ROW-X (SX): row exclusive lock SHARE (S): share lock S/ROW-X (SSX): exclusive lock NONE: lock requested but not yet obtained
MODE_REQUESTED	VARCHAR2(13)		Lock request type. The values are: ROWS_S (SS): row share lock ROW-X (SX): row exclusive lock SHARE (S): share lock S/ROW-X (SSX): exclusive lock NONE: Lock identifier obtained; lock not held or requested
LAST_CONVERT	NUMBER		The last convert
BLOCKING_OTHERS	VARCHAR2(40)		Blocking others

```
SQL> CREATE TABLE TB_DML_LOCK_LHR (ID NUMBER);

Table created.

SQL> INSERT INTO TB_DML_LOCK_LHR VALUES(1);

1 row created.

SQL> set line 9999
SQL> SELECT * FROM DBA_DML_LOCKS;

SESSION_ID OWNER   NAME              MODE_HELD     MODE_REQUESTE LAST_CONVERT BLOCKING_OTHERS
---------- ------- ----------------- ------------- ------------- ------------ -----------------
       151 SYS     TB_DML_LOCK_LHR   Row-X (SX)    None                    10 Not Blocking

SQL>
```

2.7.5 一些字段的说明

会话 1：

```
SYS@oratest S1> select distinct sid from v$mystat;

      SID
----------
       22
```

```
SYS@oratest S1> CREATE TABLE SCOTT.EMP_LHR AS SELECT * FROM SCOTT.EMP;

Table created.

SYS@oratest S1> delete from scott.EMP_LHR where empno=7369;

1 row deleted.

SYS@oratest S1>
```

会话 2：

```
SYS@oratest S2>  select distinct sid from v$mystat;

      SID
----------
      143

SYS@oratest S2> delete from scott.EMP_LHR where empno=7369;
```

====>>>>> 产生了阻塞

会话 3 查询锁：

```
SQL> set line 9999
SQL> SELECT A.TADDR,
  2        A.LOCKWAIT,
  3        A.ROW_WAIT_OBJ#,
  4        A.ROW_WAIT_FILE#,
  5        A.ROW_WAIT_BLOCK#,
  6        A.ROW_WAIT_ROW#,
  7        A.EVENT,
  8        A.P1,
  9        A.P2,
 10        A.SID,
 11        A.BLOCKING_SESSION
 12    FROM V$SESSION A
 13   WHERE A.SID IN (22, 143);
TADDR            LOCKWAIT         ROW_WAIT_OBJ# ROW_WAIT_FILE# ROW_WAIT_BLOCK# ROW_WAIT_ROW# EVENT                          P1         P2        SID BLOCKING_SESSION
---------------- ---------------- ------------- -------------- ---------------- ------------- ------------------------------ ---------- ---------- ---------- ----------------
000000007622B710                            -1              0                0             0 SQL*Net message from client    1650815232          1         22
000000007622AD00 00000000774DA0C0         77669              8             2799             0 enq: TX - row lock contention  1415053318     524299        143               22
```

	TADDR	LOCKWAIT	ROW_WAIT_OBJ#	ROW_WAIT_FILE#	ROW_WAIT_BLOCK#	ROW_WAIT_ROW#	EVENT	P1	P2	SID	BLOCKING_SESSION
1	000000007622B710		-1	0	0	0	SQL*Net message from client	1650815232	1	22	
2	000000007622AD00	00000000774DA0C0	77766	4	131	0	enq: TX - row lock contention	1415053318	262174	143	22

V$SESSION 视图的 TADDR 列表示事务处理状态对象的地址，对应于 V$TRANSACTION.ADDR 列；V$SESSION 视图的 LOCKWAIT 列表示等待锁的地址，对应于 V$LOCK 的 KADDR 列；若当前会话没有被阻塞则为空。V$SESSION 视图的 SADDR 列对应于 V$TRANSACTION 的 SES_ADDR 列。可以通过 ROW_WAIT_OBJ#、ROW_WAIT_FILE#、ROW_WAIT_BLOCK#、ROW_WAIT_ROW#这几个字段查询现在正在被锁的表的相关信息（ROWID），例如，表名、文件名及行号。P1 和 P2 根据等待事件的不同所代表的含义不同，可以从 V$EVENT_NAME 视图获知每个参数的含义。

```
SQL> SELECT D.PARAMETER1,D.PARAMETER2,D.PARAMETER3 FROM V$EVENT_NAME  D WHERE D.NAME='enq: TX - row lock contention';

PARAMETER1   PARAMETER2      PARAMETER3
------------ --------------- ----------
name|mode    usn<<16 | slot  sequence

SQL> SELECT CHR(BITAND(P1, -16777216) / 16777215) ||
  2        CHR(BITAND(P1, 16711680) / 65535) "LOCK",
  3        BITAND(P1, 65535) "MODE",
  4        TRUNC(P2 / POWER(2, 16)) AS XIDUSN,
  5        BITAND(P2, TO_NUMBER('FFFF', 'XXXX')) + 0 AS XIDSLOT,
  6        P3 XIDSQN
  7    FROM V$SESSION A
  8   WHERE A.SID IN (143);

LOCK       MODE     XIDUSN    XIDSLOT     XIDSQN
---- ---------- ---------- ---------- ----------
TX           6          4         30        894

<<<<<---从 P1 参数获知请求的锁的类型和模式；从 P2 参数可以获知槽位号

SQL> SELECT ADDR,XIDUSN,XIDSLOT,XIDSQN FROM v$transaction a WHERE a.ADDR IN ('000000007622B710');

ADDR               XIDUSN    XIDSLOT     XIDSQN
---------------- ---------- ---------- ----------
000000007622B710          4         30        894

SQL> SELECT  ADDR,XIDUSN,XIDSLOT,XIDSQN FROM v$transaction a WHERE a.SES_ADDR ='0000000077E6F600';

ADDR               XIDUSN    XIDSLOT     XIDSQN
---------------- ---------- ---------- ----------
000000007622B710          4         30        894

SQL> SELECT * FROM V$LOCK A WHERE A.SID IN (22, 143) AND A.TYPE IN ('TX','TM') AND A.KADDR='00000000774DA0C0' ORDER BY a.SID,a.TYPE;

ADDR             KADDR                SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
---------------- ---------------- ---------- -- ---------- ---------- ---------- ---------- ---------- ----------
```

```
00000000774DA068 00000000774DA0C0       143 TX     262174        894          0          6        658          0

SQL> SELECT DBMS_ROWID.ROWID_CREATE(1, 77766, 4, 131, 0) FROM DUAL;

DBMS_ROWID.ROWID_C
------------------
AAAS/GAAEAAAACDAAA

SQL> SELECT * FROM SCOTT.EMP A WHERE A.ROWID='AAAS/GAAEAAAACDAAA';

     EMPNO ENAME      JOB              MGR HIREDATE                 SAL       COMM     DEPTNO
---------- ---------- --------- ---------- ------------------- ---------- ---------- ----------
      7369 SMITH      CLERK           7902 1980-12-17 00:00:00        800                    20

SQL>
```

可以看到被锁的行的地址。

```
SQL> SELECT * FROM V$LOCK A WHERE A.SID IN (22, 143) AND A.TYPE IN ('TX','TM') ORDER BY a.SID,a.TYPE;

ADDR             KADDR                 SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
---------------- ---------------- ---------- -- ---------- ---------- ---------- ---------- ---------- ----------
00007FF44BF72D18 00007FF44BF72D78         22 TM      77766          0          3          0        793          0
000000007622B710 000000007622B788         22 TX     262174        894          6          0        793          1
00007FF44BF72D18 00007FF44BF72D78        143 TM      77766          0          3          0        787          0
00000000774DA068 00000000774DA0C0        143 TX     262174        894          0          6        787          0
6 rows selected.

SQL>
SQL>
SQL> SELECT * FROM DBA_DML_LOCKS D WHERE D.SESSION_ID IN (22, 143) ORDER BY d.SESSION_ID;

SESSION_ID OWNER    NAME     MODE_HELD     MODE_REQUESTE LAST_CONVERT BLOCKING_OTHERS
---------- -------- -------- ------------- ------------- ------------ ---------------
        22 SCOTT    EMP_LHR  Row-X (SX)    None                 1146 Not Blocking
       143 SCOTT    EMP_LHR  Row-X (SX)    None                 1140 Not Blocking

SQL> SELECT D.OWNER, D.OBJECT_NAME, D.OBJECT_ID, D.OBJECT_TYPE
  2   FROM DBA_OBJECTS D
  3   WHERE D.OBJECT_ID IN (77766);

OWNER    OBJECT_NAME   OBJECT_ID OBJECT_TYPE
-------- ------------ ---------- -------------------
SCOTT    EMP_LHR           77766 TABLE

SQL> SQL>  SELECT a.XIDUSN,
  2          a.XIDSLOT,
  3          a.XIDSQN FROM v$transaction a WHERE a.XIDSQN =894;

    XIDUSN    XIDSLOT     XIDSQN
---------- ---------- ----------
         4         30        894
SQL> SELECT 4*POWER(2,16)+30 FROM DUAL;

4*POWER(2,16)+30
----------------
          262174

SQL>
SQL> SELECT TRUNC(ID1 / POWER(2, 16)) AS XIDUSN,
  2         BITAND(ID1, TO_NUMBER('FFFF', 'XXXX')) + 0 AS XIDSLOT,
  3         894  XIDSQN
  4    FROM V$LOCK A
  5   WHERE A.SID IN (22, 143)
  6     AND A.TYPE IN ('TX', 'TM')
  7     AND A.ADDR = '000000007622B710'
  8   ORDER BY A.SID, A.TYPE;

    XIDUSN    XIDSLOT     XIDSQN
---------- ---------- ----------
         4         30        894
```

在 V$LOCK 中，当 TYPE 列的值为 TM 时，ID1 的值为 DBA_OBJECTS.OBJECT_ID；当为 TX 锁时，ID1 对应视图 V$TRANSACTION 中的 XIDUSN 字段（Undo segment number：事务对应的撤销段序列号）和 XIDSLOT 字段（Slot number：事务对应的槽位号）。其中 ID1 的高 16 位为 XIDUSN，低 16 位为 XIDSLOT。计算公式为：`SELECT TRUNC(ID1/POWER(2,16)) AS XIDUSN,BITAND(ID1,TO_NUMBER('FFFF','XXXX')) + 0 AS XIDSLOT , ID2 XIDSQN FROM DUAL;`

在 V$LOCK 中，当 TYPE 列的值为 TM 锁时，ID2 的值为 0；当为 TX 锁时，ID2 对应视图 V$TRANSACTION 中的 XIDSQN 字段（Sequence number：事务对应的序列号）。

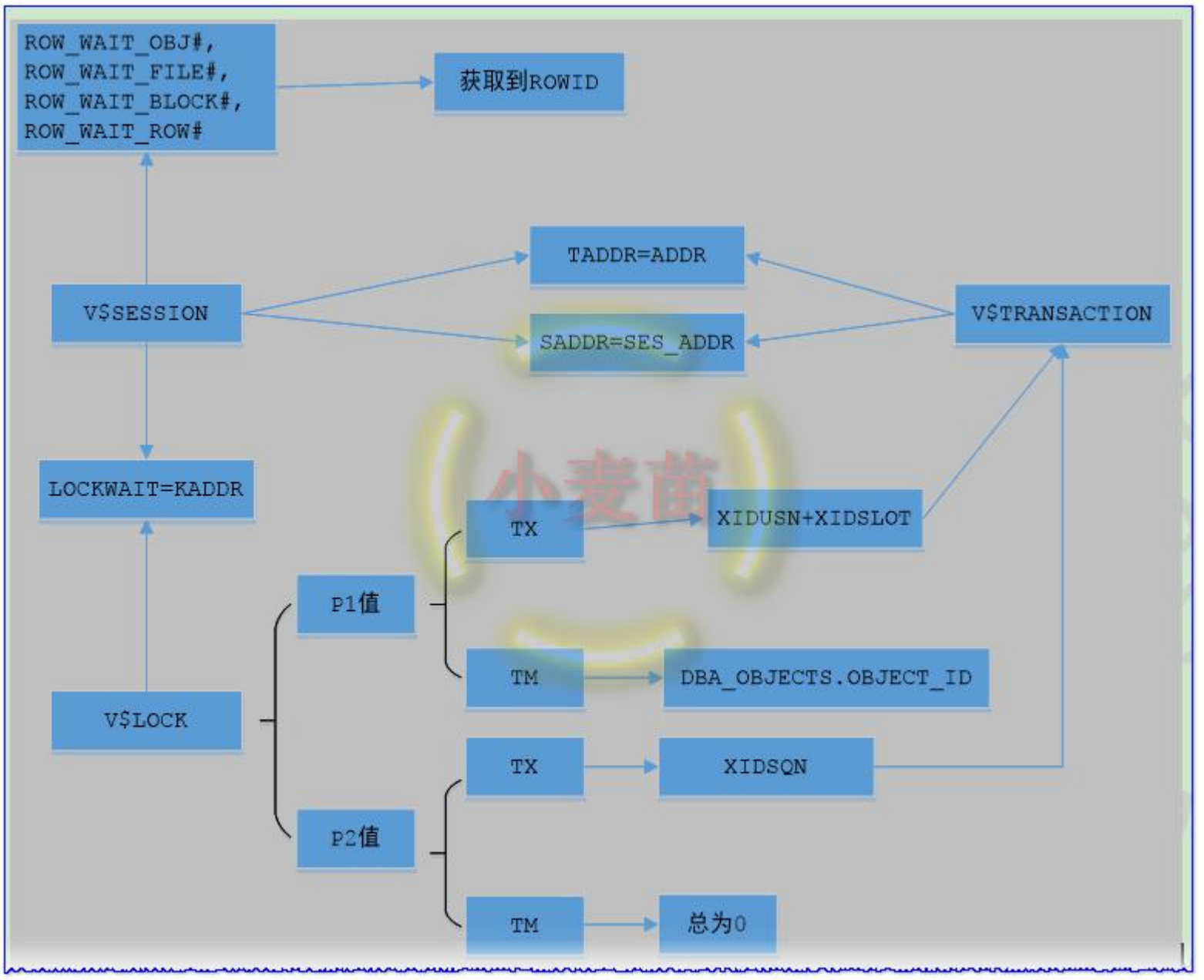
从 V$SESSION 视图可以得到所有内容：

```
SELECT A.TADDR,
       A.LOCKWAIT,
       A.ROW_WAIT_OBJ#,
       A.ROW_WAIT_FILE#,
       A.ROW_WAIT_BLOCK#,
       A.ROW_WAIT_ROW#,
       (SELECT D.OWNER || '|' || D.OBJECT_NAME || '|' || D.OBJECT_TYPE
```

```
          FROM DBA_OBJECTS D
         WHERE D.OBJECT_ID = A.ROW_WAIT_OBJ#) OBJECT_NAME,
       A.EVENT,
       A.P1,
       A.P2,
         A.P3,
       CHR(BITAND(P1, -16777216) / 16777215) ||
       CHR(BITAND(P1, 16711680) / 65535) "LOCK",
       BITAND(P1, 65535) "MODE",
       TRUNC(P2 / POWER(2, 16)) AS XIDUSN,
       BITAND(P2, TO_NUMBER('FFFF', 'XXXX')) + 0 AS XIDSLOT,
       P3 XIDSQN,
       A.SID,
       A.BLOCKING_SESSION,
       A.SADDR,
       DBMS_ROWID.ROWID_CREATE(1, 77669, 8, 2799, 0) REQUEST_ROWID,
       (SELECT B.SQL_TEXT
          FROM V$SQL B
         WHERE B.SQL_ID = NVL(A.SQL_ID, A.PREV_SQL_ID)) SQL_TEXT
  FROM V$SESSION A
 WHERE A.SID IN (143);
```

Row 1	Fields
TADDR	000000007622AD00
LOCKWAIT	00000000774DA0C0
ROW_WAIT_OBJ#	77766
ROW_WAIT_FILE#	4
ROW_WAIT_BLOCK#	131
ROW_WAIT_ROW#	0
OBJECT_NAME	SCOTT|EMP_LHR|TABLE
EVENT	enq: TX - row lock contention
P1	1415053318
P2	262174
P3	894
LOCK	TX
MODE	6
XIDUSN	4
XIDSLOT	30
XIDSQN	894
SID	143
BLOCKING_SESSION	22
SADDR	0000000077FE9A58
REQUEST_ROWID	AAAS9lAAIAAAArvAAA
SQL_TEXT	delete from scott.EMP_LHR where empno=7369

2.7.5.1 关联关系图



2.8 参数

2.8.1 DML_LOCKS 参数

可以获得的 TX 锁定的总个数由初始化参数 transactions 决定，而可以获得的 TM 锁定的个数则由初始化参数 dml_locks 决定

```
select name,value from v$parameter where name in('transactions','dml_locks');
```

```
SYS@racdb1> col name format a15
SYS@racdb1> col value format a5
SYS@racdb1> select name,value from v$parameter where name in('transactions','dml_locks');

NAME          VALUE
--------------- -----
dml_locks      1088
transactions    272

SYS@racdb1> select 272*4 from dual;

    272*4
----------
     1088
```

DML_LOCKS 参数属于推导参数，**DML_LOCKS=4 * TRANSACTIONS**。

```
select resource_name as "R_N",current_utilization as "C_U",max_utilization as "M_U",initial_allocation as "I_U"
from v$resource_limit
where resource_name in('transactions','dml_locks');
```

```
SYS@racdb1> select resource_name as "R_N",current_utilization as "C_U",max_utilization as
"M_U",initial_allocation as "I_U"
  2  from v$resource_limit
  3  where resource_name in('transactions','dml_locks');

R_N                            C_U       M_U I_U
---------------------------- ---------- ---------- --------------------
dml_locks                             0         28       1088
transactions                          0          6        272
```

系统中允许的 TM 锁总数可以由你配置（有关细节请见 Oracle Database Reference 手册中的 DML_LOCKS 参数定义）。实际上，这个数可能设置为 0。但这并不是说你的数据库变成了一个只读数据库（没有锁），而是说不允许 DDL。在非常专业的应用（如 RAC 实现）中，这一点就很有用，可以减少实例内可能发生的协调次数。通过使用 ALTER TABLE TABLENAME DISABLE TABLE LOCK 命令，还可以逐对象地禁用 TM 锁。这是一种快捷方法，可以使意外删除表的难度更大”，因为在删除表之前，你必须重新启用表锁。还能用它检测由于外键未加索引而导致的全表锁（前面已经讨论过）。

Property	Description
Parameter type	Integer
Default value	Derived: 4 * TRANSACTIONS
Modifiable	No
Range of values	20 to unlimited; a setting of 0 disables enqueues
Basic	No
Oracle RAC	You must set this parameter for every instance, and all instances must have positive values or all must be 0.

A DML lock is a lock obtained on a table that is undergoing a DML operation (insert, update, delete). DML_LOCKS specifies the maximum number of DML locks—one for each table modified in a transaction. The value should equal the grand total of locks on tables currently referenced by all users. For example, if three users are modifying data in one table, then three entries would be required. If three users are modifying data in two tables, then six entries would be required.

The default value assumes an average of four tables referenced for each transaction. For some systems, this value may not be enough.

Enqueues are shared memory structures that serialize access to database resources. If you set the value of DML_LOCKS to 0, enqueues are disabled and performance is slightly increased. However, you should be aware of the following restrictions when you set you DML_LOCKS to 0:

- You cannot use DROP TABLE, CREATE INDEX statements
- You cannot use explicit lock statements such as LOCK TABLE IN EXCLUSIVE MODE
- Enterprise Manager cannot run on any instances for which DML_LOCKS is set to 0

Oracle holds more locks during parallel DML than during serial execution. Therefore, if your database supports a lot of parallel DML, you may need to increase the value of this parameter.

2.8.2 DDL_LOCK_TIMEOUT

11g 以前，DDL 语句是不会等待 DML 语句的，当 DDL 语句访问的对象正在执行的 DML 语句，会立即报错 ORA-00054：资源正忙，但指定以 NOWAIT 方式获取资源，或者超时失效。而在 11g 以后，DDL_LOCK_TIMEOUT 参数可以修改这一状态，当 DDL_LOCK_TIMEOUT=0 时，DDL 不等待 DML，当 DDL_LOCK_TIMEOUT 为 N（秒）时，DDL 等待 DML N 秒，该值默认为 0。

Property	Description
Parameter type	Integer
Default value	0
Modifiable	ALTER SESSION
Range of values	0 to 1,000,000 (in seconds)
Basic	No

DDL_LOCK_TIMEOUT specifies a time limit for how long DDL statements will wait in a DML lock queue. The default value of zero indicates a status of NOWAIT. The maximum value of 1,000,000 seconds will result in the DDL statement waiting forever to acquire a DML lock.

If a lock is not acquired before the timeout period expires, then an error is returned.

会话 1：

```
Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.3.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> set sqlprompt "_user'@'_connect_identifier S1> "
SYS@oratest S1> set timing on
SYS@oratest S1> update scott.emp set ename='' where empno=7499;

1 row updated.

Elapsed: 00:00:00.00
SYS@oratest S1>
```

会话 2：

```
SQL> set sqlprompt "_user'@'_connect_identifier S2> "
SYS@oratest S2> set timing on
SYS@oratest S2> drop table scott.emp;
drop table scott.emp
             *
ERROR at line 1:
ORA-00054: resource busy and acquire with NOWAIT specified or timeout expired


Elapsed: 00:00:00.74
SYS@oratest S2> show parameter ddl_lock_timeout

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
ddl_lock_timeout                     integer     0
SYS@oratest S2> alter session set ddl_lock_timeout=5;

Session altered.

Elapsed: 00:00:00.00
SYS@oratest S2> drop table scott.emp;
drop table scott.emp
             *
ERROR at line 1:
ORA-00054: resource busy and acquire with NOWAIT specified or timeout expired


Elapsed: 00:00:05.01
SYS@oratest S2> alter session set ddl_lock_timeout=10;

Session altered.

Elapsed: 00:00:00.00
SYS@oratest S2> drop table scott.emp;
drop table scott.emp
             *
ERROR at line 1:
ORA-00054: resource busy and acquire with NOWAIT specified or timeout expired


Elapsed: 00:00:10.03
SYS@oratest S2>
```

综上，设置 ddl_lock_timeout 为 N（秒）后，DDL 执行后将等待 N 秒钟后才抛出报错信息。在 ddl_lock_timeout 为默认值 0 时，DDL 语句提交之后马上报错。

2.9 for update、for update of、for update nowait

SELECT...FOR UPDATE 语句的语法如下：

SELECT ... FOR UPDATE [OF column_list][WAIT n|NOWAIT][SKIP LOCKED];

其中：

- OF 这个 OF 子句在牵连到多个表时，具有较大作用，如不使用 OF 指定锁定的表的列，则所有表的相关行均被锁定，若在 OF 中指定了需修改的列，则只有与这些列相关的表的行才会被锁定。
- WAIT 子句指定等待其他用户释放锁的秒数，防止无限期的等待。

“使用 FOR UPDATE WAIT”子句的优点如下：

1 防止无限期地等待被锁定的行；

2 允许应用程序中对锁的等待时间进行更多的控制。

3 对于交互式应用程序非常有用，因为这些用户不能等待不确定

4 若使用了 skip locked，则可以越过锁定的行，不会报告由 wait n 引发的‘资源忙’异常报告

2.9.1 FOR UPDATE 和 FOR UPDATE NOWAIT 的区别

for update nowait 和 for update 都会对所查询到得结果集进行加锁，所不同的是，如果另外一个进程正在修改结果集中的数据，for update nowait 不会进行资源等待，只要发现结果集中有些数据被加锁，立刻返回“ORA-00054 错误，内容是资源正忙，但指定以 NOWAIT 方式获取资源”。

for update 和 for update nowait 加上的是一个行级锁，也就是只有符合 where 条件的数据被加锁。如果仅仅用 update 语句来更改数据时，可能会因为加不上锁而没有响应地、莫名其妙地等待，但如果在此之前，for update NOWAIT 语句将要更改的数据试探性地加锁，就可以通过立即返回的错误提示而明白其中的道理，或许这就是 For Update 和 NOWAIT 的意义之所在。

会话 1：

```
SYS@oratest S1> SELECT EMPNO, ENAME FROM SCOTT.EMP WHERE EMPNO = '7369' FOR UPDATE NOWAIT;

    EMPNO ENAME
---------- ----------
     7369 SMITH
```

会话 2：

```
SYS@oratest S2> SELECT EMPNO, ENAME FROM SCOTT.EMP WHERE EMPNO = '7369' FOR UPDATE NOWAIT;
SELECT EMPNO, ENAME FROM SCOTT.EMP WHERE EMPNO = '7369' FOR UPDATE NOWAIT
```

```
                         *
ERROR at line 1:
ORA-00054: resource busy and acquire with NOWAIT specified or timeout expired
```

上面会话都提交 commit，开启会话 1，不使用 NOWAIT：

```
SYS@oratest S1>  SELECT EMPNO, ENAME FROM SCOTT.EMP WHERE EMPNO = '7369' FOR UPDATE ;

    EMPNO ENAME
---------- ----------
     7369 SMITH

SYS@oratest S1>
```

开启另一会话

```
SYS@oratest S2>  SELECT EMPNO, ENAME FROM SCOTT.EMP WHERE EMPNO = '7369' FOR UPDATE ;


====>>>>> 产生了阻塞
```

阻塞，不返回错误。提交第一个会话，第二个回话自动执行，然后提交第二个会话

2.9.2 SELECT...FOR UPDATE OF COLUMNS

select for update of，这个 of 子句在牵连到多个表时，具有较大作用，如不使用 of 指定锁定的表的列，则所有表的相关行均被锁定，若在 of 中指定了需修改的列，则只有与这些列相关的表的行才会被锁定。

会话 1：

```
SYS@oratest S1> SELECT * FROM SCOTT.EMP E, SCOTT.DEPT D WHERE E.DEPTNO = D.DEPTNO FOR UPDATE;

    EMPNO ENAME      JOB              MGR HIREDATE                 SAL       COMM     DEPTNO     DEPTNO DNAME          LOC
--------- ---------- --------- --------- ------------------- --------- ---------- ---------- ---------- -------------- -------------
     7369 SMITH      CLERK          7902 1980-12-17 00:00:00       800                    20         20 RESEARCH       DALLAS
     7499 ALLEN      SALESMAN       7698 1981-02-20 00:00:00      1600        300         30         30 SALES          CHICAGO
     7521 WARD       SALESMAN       7698 1981-02-22 00:00:00      1250        500         30         30 SALES          CHICAGO
     7566 JONES      MANAGER        7839 1981-04-02 00:00:00      2975                    20         20 RESEARCH       DALLAS
     7654 MARTIN     SALESMAN       7698 1981-09-28 00:00:00      1250       1400         30         30 SALES          CHICAGO
     7698 BLAKE      MANAGER        7839 1981-05-01 00:00:00      2850                    30         30 SALES          CHICAGO
     7782 CLARK      MANAGER        7839 1981-06-09 00:00:00      2450                    10         10 ACCOUNTING     NEW YORK
     7788 SCOTT      ANALYST        7566 1987-04-19 00:00:00      3000                    20         20 RESEARCH       DALLAS
     7839 KING       PRESIDENT           1981-11-17 00:00:00      5000                    10         10 ACCOUNTING     NEW YORK
     7844 TURNER     SALESMAN       7698 1981-09-08 00:00:00      1500          0         30         30 SALES          CHICAGO
     7876 ADAMS      CLERK          7788 1987-05-23 00:00:00      1100                    20         20 RESEARCH       DALLAS
     7900 JAMES      CLERK          7698 1981-12-03 00:00:00       950                    30         30 SALES          CHICAGO
     7902 FORD       ANALYST        7566 1981-12-03 00:00:00      3000                    20         20 RESEARCH       DALLAS
     7934 MILLER     CLERK          7782 1982-01-23 00:00:00      1300                    10         10 ACCOUNTING     NEW YORK

14 rows selected.
```

会话 2：

```
SYS@oratest S2> SELECT * FROM SCOTT.DEPT FOR UPDATE NOWAIT;
SELECT * FROM SCOTT.DEPT FOR UPDATE NOWAIT
                *
ERROR at line 1:
ORA-00054: resource busy and acquire with NOWAIT specified or timeout expired


SYS@oratest S2>
SYS@oratest S2> SELECT * FROM V$LOCK A WHERE A.SID IN (16,27) AND A.TYPE IN ('TX','TM') ORDER BY a.SID,a.TYPE;

ADDR             KADDR                 SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
---------------- ---------------- ---------- -- ---------- ---------- ---------- ---------- ---------- ----------
00007F8FABF13398 00007F8FABF133F8         16 TM      77667          0          3          0        201          0
00007F8FABF13398 00007F8FABF133F8         16 TM      77669          0          3          0        201          0
000000007620A7C0 000000007620A838         16 TX     327687       1138          6          0        201          0

SYS@oratest S2> SELECT * FROM DBA_DML_LOCKS D WHERE D.SESSION_ID IN (16,27) ORDER BY d.SESSION_ID;

SESSION_ID OWNER     NAME   MODE_HELD     MODE_REQUESTE LAST_CONVERT BLOCKING_OTHERS
---------- --------- ------ ------------- ------------- ------------ ---------------
        16 SCOTT     EMP    Row-X (SX)    None                  225 Not Blocking
        16 SCOTT     DEPT   Row-X (SX)    None                  225 Not Blocking
SYS@oratest S2>
```

可以看到，会话 1 在 SCOTT.EMP 和 SCOTT.DEPT 表上都加上了 3 级的行级排它锁。

提交以上的会话，然后继续试验 OF 特性：

会话 1：

```
SYS@oratest S1> SELECT * FROM SCOTT.EMP E, SCOTT.DEPT D WHERE E.DEPTNO = D.DEPTNO FOR UPDATE OF SAL ;

    EMPNO ENAME      JOB              MGR HIREDATE                 SAL       COMM     DEPTNO     DEPTNO DNAME          LOC
--------- ---------- --------- --------- ------------------- --------- ---------- ---------- ---------- -------------- -------------
     7369 SMITH      CLERK          7902 1980-12-17 00:00:00       800                    20         20 RESEARCH       DALLAS
     7499 ALLEN      SALESMAN       7698 1981-02-20 00:00:00      1600        300         30         30 SALES          CHICAGO
     7521 WARD       SALESMAN       7698 1981-02-22 00:00:00      1250        500         30         30 SALES          CHICAGO
```

```
     7566 JONES      MANAGER         7839 1981-04-02 00:00:00       2975                    20         20 RESEARCH       DALLAS
     7654 MARTIN     SALESMAN        7698 1981-09-28 00:00:00       1250       1400         30         30 SALES          CHICAGO
     7698 BLAKE      MANAGER         7839 1981-05-01 00:00:00       2850                    30         30 SALES          CHICAGO
     7782 CLARK      MANAGER         7839 1981-06-09 00:00:00       2450                    10         10 ACCOUNTING     NEW YORK
     7788 SCOTT      ANALYST         7566 1987-04-19 00:00:00       3000                    20         20 RESEARCH       DALLAS
     7839 KING       PRESIDENT            1981-11-17 00:00:00       5000                    10         10 ACCOUNTING     NEW YORK
     7844 TURNER     SALESMAN        7698 1981-09-08 00:00:00       1500          0         30         30 SALES          CHICAGO
     7876 ADAMS      CLERK           7788 1987-05-23 00:00:00       1100                    20         20 RESEARCH       DALLAS
     7900 JAMES      CLERK           7698 1981-12-03 00:00:00        950                    30         30 SALES          CHICAGO
     7902 FORD       ANALYST         7566 1981-12-03 00:00:00       3000                    20         20 RESEARCH       DALLAS
     7934 MILLER     CLERK           7782 1982-01-23 00:00:00       1300                    10         10 ACCOUNTING     NEW YORK

14 rows selected.
```

会话 2：

```
SYS@oratest S2> SELECT * FROM SCOTT.DEPT FOR UPDATE NOWAIT;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON

SYS@oratest S2>

SYS@oratest S1> SELECT * FROM V$LOCK A WHERE A.SID IN (16,27) AND A.TYPE IN ('TX','TM') ORDER BY a.SID,a.TYPE;

ADDR             KADDR                 SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
---------------- ---------------- ---------- -- ---------- ---------- ---------- ---------- ---------- ----------
00007F73CBCE38D8 00007F73CBCE3938         16 TM      77669          0          3          0        114          0
000000007620A7C0 000000007620A838         16 TX     327698       1138          6          0        114          0
00007F73CBCE38D8 00007F73CBCE3938         27 TM      77667          0          3          0         81          0
000000007620B1D0 000000007620B248         27 TX     131076       1128          6          0         81          0

SYS@oratest S1> SELECT * FROM DBA_DML_LOCKS D WHERE D.SESSION_ID IN (16,27) ORDER BY d.SESSION_ID;

SESSION_ID OWNER     NAME   MODE_HELD     MODE_REQUESTE LAST_CONVERT BLOCKING_OTHERS
---------- --------- ------ ------------- ------------- ------------ ---------------
        16 SCOTT     EMP    Row-X (SX)    None                   123 Not Blocking
        27 SCOTT     DEPT   Row-X (SX)    None                    90 Not Blocking

SYS@oratest S1>
```

可以看到，会话 1 在 SCOTT.EMP 表上加上了 3 级的行级排它锁，而会话 2 在和 SCOTT.DEPT 表上加上了 3 级的行级排它锁。

2.9.3 9i 中的 SELECT FOR UPDATE 锁

```
SQL*Plus: Release 9.2.0.1.0 - Production on 星期一 11月 14 17:29:40 2016

Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.

请输入用户名:  sys as sysdba
请输入口令:

连接到:
Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production

SQL> set line 9999
SQL> set pagesize 9999
SQL> select * from scott.emp for update;

     EMPNO ENAME      JOB              MGR HIREDATE                 SAL       COMM     DEPTNO
---------- ---------- --------- ---------- ------------------ ---------- ---------- ----------
      7369 SMITH      CLERK           7902 1980-12-17 00:00:00        800                    20
      7499 ALLEN      SALESMAN        7698 1981-02-20 00:00:00       1600        300         30
      7521 WARD       SALESMAN        7698 1981-02-22 00:00:00       1250        500         30
      7566 JONES      MANAGER         7839 1981-04-02 00:00:00       2975                    20
      7654 MARTIN     SALESMAN        7698 1981-09-28 00:00:00       1250       1400         30
      7698 BLAKE      MANAGER         7839 1981-05-01 00:00:00       2850                    30
      7782 CLARK      MANAGER         7839 1981-06-09 00:00:00       2450                    10
      7788 SCOTT      ANALYST         7566 1987-04-19 00:00:00       3000                    20
      7839 KING       PRESIDENT            1981-11-17 00:00:00       5000                    10
      7844 TURNER     SALESMAN        7698 1981-09-08 00:00:00       1500          0         30
      7876 ADAMS      CLERK           7788 1987-05-23 00:00:00       1100                    20
      7900 JAMES      CLERK           7698 1981-12-03 00:00:00        950                    30
      7902 FORD       ANALYST         7566 1981-12-03 00:00:00       3000                    20
      7934 MILLER     CLERK           7782 1982-01-23 00:00:00       1300                    10

已选择 14 行。

SQL> select distinct sid from v$mystat;
```

```
       SID
----------
        10

SQL> SELECT * FROM V$LOCK A WHERE A.SID=10  ORDER BY a.SID,a.TYPE;

ADDR     KADDR           SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
-------- -------- ---------- -- ---------- ---------- ---------- ---------- ---------- ----------
67B4E0F8 67B4E10C         10 TM      30139          0          2          0         35          0
67BAB0CC 67BAB1D8         10 TX     131082       2874          6          0         25          0

SQL> SELECT * FROM DBA_DML_LOCKS D WHERE D.SESSION_ID =10 ORDER BY d.SESSION_ID;

SESSION_ID OWNER     NAME   MODE_HELD     MODE_REQUESTE LAST_CONVERT BLOCKING_OTHERS
---------- --------- ------ ------------- ------------- ------------ ------------------
        10 SCOTT     EMP    Row-S (SS)    None                   99 Not Blocking
```

可以看到在 Oracle 10g 之前，SELECT FOR UPDATE 获取的是 2 级 TM 锁，在 Oracle 10g 及其之后的版本中，SELECT FOR UPDATE 获取的是 3 级 TM 锁。

2.9.4 总结

1. SELECT * FROM TABLE1 FOR UPDATE 锁定表的所有行，其它会话只能读不能写
2. SELECT * FROM TABLE1 WHERE PKID = 1 FOR UPDATE 只锁定 PKID=1 的行
3. SELECT * FROM TABLE1 A JOIN TABLE2 B ON A.PKID=B.PKID FOR UPDATE 锁定两个表的所有记录
4. SELECT * FROM TABLE1 A JOIN TABLE2 B ON A.PKID=B.PKID WHERE A.PKID = 10 FOR UPDATE 锁定两个表的中满足条件的行
5. SELECT * FROM TABLE1 A JOIN TABLE2 B ON A.PKID=B.PKID WHERE A.PKID = 10 FOR UPDATE OF A.PKID 只锁定 TABLE1 中满足条件的行

FOR UPDATE 是把所有的表都锁定。FOR UPDATE OF 根据 OF 后表的条件锁定相对应的表。

2.10 Oracle 包被锁定的原因分析及解决方案

摘抄自网络，小麦苗感觉自己对这个部分也没啥可写的，主要是包不能编译的时候需要查询 DBA_DDL_LOCKS 视图，最后杀会话的时候需要稳重一点。

在数据库的开发过程中，经常碰到包、存储过程、函数无法编译或编译时会导致 PL/SQL 无法响应的问题。碰到这种问题，基本上都要重启数据库解决，严重浪费开发时间。本文将就产生这种现象的原因和解决方案做基本的介绍。

问题分析

从事数据库开发的都知道锁的概念，如：执行 Update Table xxx Where xxx 的时候就会产生锁。这种常见的锁在 Oracle 里面被称为 DML 锁。在 Oracle 中还有一种 DDL 锁，主要用来保证存储过程、表结构、视图、包等数据库对象的完整性，这种锁的信息可以在 DBA_DDL_LOCKS 中查到。**注意：V$LOCKED_OBJECT 记录的是 DML 锁信息，DDL 锁的信息不在里面。**

对应 DDL 锁的是 DDL 语句，DDL 语句全称数据定义语句（Data Define Language）。用于定义数据的结构或 Schema，如：CREATE、ALTER、DROP、TRUNCATE、COMMENT、RENAME。**当我们在执行某个存储过程、或者编译它的时候 Oracle 会自动给这个对象加上 DDL 锁，同时也会对这个存储过程所引用的对象加锁。**

举例：

1、 打开一个 PL/SQL，开始调试某个函数（假设为：FUN_CORE_SERVICECALL），并保持在调试状态

2、 打开一个 SQL Window，输入 `Select * From dba_ddl_locks a Where a.name = 'FUN_CORE_SERVICECALL'` 会发现一行记录：

	SESSION_ID	OWNER	NAME	TYPE	MODE_HELD	MODE_REQUESTED
1	1612	HIS3	FUN_CORE_SERVICECALL	Table/Procedure/Type	Null	None

3、 打开一个新的 PL/SQL，重新编译这个函数。我们会发现此时已经无法响应了

4、 回到第一个 PL/SQL,重新执行 `Select * From dba_ddl_locks a Where a.name = 'FUN_CORE_SERVICECALL'` 我们将会看到如下记录：

SESSION_ID	OWNER	NAME	TYPE	MODE_HELD	MODE_REQUESTED
1612	HIS3	FUN_CORE_SERVICECALL	Table/Procedure/Type	Null	None
1510	HIS3	FUN_CORE_SERVICECALL	Table/Procedure/Type	Exclusive	None

5、 上述的情况表明发生了锁等待的情况。

当我们试图编译、修改存储过程、函数、包等对数据对象的时候，如果别人也正在编译或修改他们时就会产生锁等待；或者我们在编译某个存储过程的时候，如果它所引用的数据库对象正在被修改应该也会产生锁等待。这种假设有兴趣的兄弟可以测试下，不过比较困难。

解决方案

碰到这种问题，如果知道是被谁锁定了（可以查出来的），可以让对方尽快把锁释放掉；实在查不出来只能手工将这个锁杀掉了。步骤如下：

1、 首先查出哪些进程锁住了这个对象，语句如下：

```
Select b.SID,b.SERIAL#
```

```
 From dba_ddl_locks a, v$session b
Where a.session_id = b.SID
 And a.name = 'FUN_CORE_SERVICECALL';
```

2、 执行如下语句杀进程：`alter system kill session 'sid,serial#' IMMEDIATE;`

3、 执行了以上的语句后，有的时候不一定能够将进程杀掉。这个时候就需要连到数据库服务器上杀掉服务器端的进程了，查询语句：

```
Select spid, osuser, s.program
  From v$session s, v$process p
 Where s.paddr = p.addr
   And s.sid =（上面查出来的 SID）
```

在服务器上执行如下语句：

#kill -9 spid（UNIX 平台）

orakill sid thread（Windows 平台 SID 是 Oracle 的实例名，thread 是上面查出来的 SID）

执行完 4 步以后基本上就可以杀掉这些锁死的进程了，不放心的话可以再执行第一步确认下。

2.10.1 实验

```
SQL> select distinct sid from v$mystat;

      SID
----------
       24

SQL> CREATE OR REPLACE PROCEDURE PRO_TESTDDL_LHR AS
  2
  3   V_COUNT NUMBER;
  4
  5  BEGIN
  6
  7   SELECT COUNT(1) INTO V_COUNT FROM SCOTT.EMP_LHR;
  8
  9   DBMS_LOCK.SLEEP(600);
 10
 11  END;
 12  /

Procedure created.

SQL> exec PRO_TESTDDL_LHR;



====>>>>> 脚本在执行
```

查看 DDL 锁：

```
SELECT * FROM DBA_DDL_LOCKS D WHERE D.SESSION_ID = 24;
```

	SESSION_ID	OWNER	NAME	TYPE	MODE_HELD	MODE_REQUESTED
1	24	SYS	DBMS_LOCK	Body	Null	None
2	24	SYS	DBMS_STANDARD	Table/Procedure/Type	Null	None
3	24	SYS	DBMS_LOCK	Table/Procedure/Type	Null	None
4	24	SYS	PRO_TESTDDL_LHR	Table/Procedure/Type	Null	None
5	24	SYS	DATABASE	18	Null	None

```
SELECT *
  FROM V$ACCESS A
 WHERE A.SID = 24
   AND A.OBJECT IN ('PRO_TESTDDL_LHR', 'EMP_LHR', 'DBMS_LOCK');
```

	SID	OWNER	OBJECT	TYPE
1	24	SYS	PRO_TESTDDL_LHR	PROCEDURE
2	24	SYS	DBMS_LOCK	PACKAGE
3	24	SCOTT	EMP_LHR	TABLE

2.11 创建索引的锁

2.11.1 创建或重建索引会阻塞 DML 操作

版本：11.2.0.3

首先建表 T_INDEX_161113 并插入很多数据

```
SYS@oratest S1> CREATE TABLE T_INDEX_161113 AS SELECT * FROM DBA_OBJECTS;

Table created.

SYS@oratest S1> INSERT INTO T_INDEX_161113 SELECT * FROM T_INDEX_161113;

75349 rows created.

SYS@oratest S1> INSERT INTO T_INDEX_161113 SELECT * FROM T_INDEX_161113;

150698 rows created.

SYS@oratest S1> INSERT INTO T_INDEX_161113 SELECT * FROM T_INDEX_161113;

301396 rows created.

SYS@oratest S1> INSERT INTO T_INDEX_161113 SELECT * FROM T_INDEX_161113;

602792 rows created.

SYS@oratest S1> COMMIT;

Commit complete.
```

接着再在该表上创建一个索引

```
SYS@oratest S1> CREATE INDEX IDX_TEST_LHR ON T_INDEX_161113(OBJECT_NAME);
```

在创建索引的同时，在会话 2 上插入一条记录：

```
SYS@oratest S2> INSERT INTO T_INDEX_161113 SELECT * FROM T_INDEX_161113 WHERE ROWNUM<=1;
```

====>>>>> 产生了阻塞

在创建索引的同时，查询相关锁的信息：

```
SQL> SELECT SID,
  2        A.BLOCKING_SESSION,
  3        EVENT,
  4        A.P1,
  5        A.P2,
  6        A.P3,
  7        CHR(BITAND(P1, -16777216) / 16777215) ||
  8        CHR(BITAND(P1, 16711680) / 65535) "LOCK",
  9        BITAND(P1, 65535) "MODE",
 10        (SELECT OBJECT_NAME FROM DBA_OBJECTS D WHERE D.OBJECT_ID = A.P2) OBJECT_NAME
 11    FROM GV$SESSION A
 12   WHERE A.SID=141;

       SID BLOCKING_SESSION EVENT                          P1         P2         P3 LOCK       MODE OBJECT_NAME
---------- ---------------- ------------------------- ---------- ---------- ---------- ---- ---------- ----------------
       142               21 enq: TM - contention    1414332419      77629          0 TM            3 T_INDEX_161113

SQL> SELECT * FROM V$LOCK A WHERE A.SID IN (21,142) AND A.TYPE IN ('TX','TM');
ADDR             KADDR                 SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
---------------- ---------------- ---------- -- ---------- ---------- ---------- ---------- ---------- ----------
00007F44001842E0 00007F4400184340        142 TM      77629          0          0          3          2          0
00007F44001842E0 00007F4400184340         21 TM      77629          0          4          0          3          1
00007F44001842E0 00007F4400184340         21 TM         18          0          3          0          3          0
0000000076273C58 0000000076273CD0         21 TX      65567        846          6          0          3          0

SQL> SELECT * FROM DBA_DML_LOCKS D WHERE D.SESSION_ID IN (21, 142);
SESSION_ID OWNER     NAME                 MODE_HELD     MODE_REQUESTE LAST_CONVERT BLOCKING_OTHERS
---------- --------- -------------------- ------------- ------------- ------------ ------------------
       142 SYS       T_INDEX_161113       None          Row-X (SX)               2 Not Blocking
        21 SYS       T_INDEX_161113       Share         None                     3 Blocking
        21 SYS       OBJ$                 Row-X (SX)    None                     3 Not Blocking

SQL> SELECT D.OWNER, D.OBJECT_NAME, D.OBJECT_ID, D.OBJECT_TYPE
  2    FROM DBA_OBJECTS D
  3   WHERE D.OBJECT_ID IN (18, 77629);

OWNER      OBJECT_NAME            OBJECT_ID OBJECT_TYPE
---------- -------------------- ---------- -------------------
SYS        T_INDEX_161113            77629 TABLE
SYS        OBJ$                         18 TABLE
SQL> SELECT * FROM DBA_DDL_LOCKS D WHERE D.SESSION_ID IN (21, 142) AND D.name NOT IN
('STANDARD','DICTIONARY_OBJ_OWNER','DICTIONARY_OBJ_NAME','UTL_RAW','DBMS_APPLICATION_INFO','SDO_GEOR_DEF','SQL_TXT','DBMS_ASSERT','SDO_GEOR_DEF'
,'TRACE_PUT_LINE','PLITBLM','DICTIONARY_OBJ_TYPE','DDLREPLICATION','DBMS_STANDARD','DBMS_APPLICATION_INFO','UTL_FILE','DDLAUX','DBMS_ASSERT','ST
ANDARD','UTL_RAW','DDLREPLICATION','UTL_FILE','DDLAUX','GGS_MARKER_SEQ','DATABASE','LOGIN_USER','FILTERDDL','DBMS_UTILITY','GGS_DDL_SEQ','SYSEVE
NT','DBMS_UTILITY','LOGIN_USER','UTL_FILE','DATABASE','SDO_GEOR_DEF','UTL_RAW','GGS_DDL_SEQ','SDO_GEOR_DEF','DICTIONARY_OBJ_TYPE','UTL_RAW','DDL
REPLICATION','DBMS_UTILITY','SYSEVENT','IS_VPD_ENABLED','DBMS_APPLICATION_INFO','FILTERDDL','DDLREPLICATION','STANDARD','DDLAUX','GGS_MARKER_SEQ
','DDLAUX','SQL_TXT','PLITBLM','AW_DROP_PROC','DBMS_APPLICATION_INFO','DBMS_UTILITY','DICTIONARY_OBJ_OWNER','DICTIONARY_OBJ_NAME','STANDARD','DB
MS_STANDARD','TRACE_PUT_LINE','UTL_FILE');

SESSION_ID OWNER     NAME             TYPE         MODE_HELD MODE_REQU
---------- --------- ---------------- ------------ --------- ---------
        21           SYS              73           Share     None
        21 SYS       IDX_TEST_LHR     Index        Exclusive None
```

可以发现在会话 1 中，在创建索引的过程中会生成 2 个 TM 锁，锁类别分别为 4 和 3，根据查询结果发现 lmode=4 的 object_id 为 77629 的对象对应的是 T_INDEX_161113 这个表，对应的是 TM 的 S 锁。另一个 lmode=3 的锁对象是系统基表 OBJ$表，允许其它会话对该表执行 DML 操作。可以得出这样一个结论:当对表进行创建索引操作时，会伴随出现 LMODE=4 的 S 锁。根据锁的兼容模式可以发现 **S 锁和任何 DML 操作都是冲突的！**所以，尤其是在生产上，当在一个很大的表上进行索引创建的时候，任何对该表的 DML 操作都会被夯住!!!

从 DBA_DDL_LOCKS 视图可以看到，建索引的同时有 6 级排它 DDL 锁。

2.11.2 Oracle 11g 下 ONLINE 选项不会堵塞 DML 操作

版本：11.2.0.3

接着上面的实验，重建索引的时候加上 ONLINE，由于会话断开了，重新开 2 个会话，会话 1 为 22，会话 2 为 142：

```
SYS@oratest S1> ALTER INDEX IDX_TEST_LHR REBUILD ONLINE;
```

在创建索引的同时，在会话 2 上插入一条记录：

```
SYS@oratest S2> INSERT INTO T_INDEX_161113 SELECT * FROM T_INDEX_161113 WHERE ROWNUM<=1;

1 row created.
```

====>>>>> 加上 ONLINE 后无阻塞产生

在创建索引的同时，查询相关锁的信息：

```
SQL> SELECT * FROM V$LOCK A WHERE A.SID IN (22,141) AND A.TYPE IN ('TX','TM');

ADDR             KADDR                SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
---------------- ---------------- ---------- -- ---------- ---------- ---------- ---------- ---------- ----------
00000000774D9C08 00000000774D9C60         22 TX     327688       1122          0          4        761          0
00007FD883B38350 00007FD883B383B0         22 TM      77629          0          2          0        768          0
00007FD883B38350 00007FD883B383B0         22 TM      77643          0          4          0        767          0
0000000076274668 00000000762746E0         22 TX     196612       1119          6          0        768          0
0000000076236E38 0000000076236EB0        141 TX     327688       1122          6          0        763          1
00007FD883B38350 00007FD883B383B0        141 TM      77629          0          3          0        763          0

6 rows selected.

SQL> SELECT * FROM DBA_DML_LOCKS D WHERE D.SESSION_ID IN (22,141);

SESSION_ID OWNER      NAME                MODE_HELD     MODE_REQUESTE LAST_CONVERT BLOCKING_OTHERS
---------- ---------- ------------------- ------------- ------------- ------------ ----------------------------------------
       141 SYS        T_INDEX_161113      Row-X (SX)    None                   625 Not Blocking
        22 SYS        T_INDEX_161113      Row-S (SS)    None                   630 Not Blocking
        22 SYS        SYS_JOURNAL_77631   Share         None                   629 Not Blocking
SQL>  SELECT D.OWNER, D.OBJECT_NAME, D.OBJECT_ID, D.OBJECT_TYPE
  2        FROM DBA_OBJECTS D
  3       WHERE D.OBJECT_ID IN (77629, 77643);

OWNER      OBJECT_NAME               OBJECT_ID OBJECT_TYPE
---------- ------------------------ ---------- -------------------
SYS        SYS_JOURNAL_77631             77643 TABLE
SYS        T_INDEX_161113                77629 TABLE

SQL> SELECT * FROM DBA_DDL_LOCKS D WHERE D.SESSION_ID IN (22,141) AND D.name NOT IN
('STANDARD','DICTIONARY_OBJ_OWNER','DICTIONARY_OBJ_NAME','UTL_RAW','DBMS_APPLICATION_INFO','SDO_GEOR_DEF','SQL_TXT','DBMS_ASSERT','SDO_GEOR_DE
F','TRACE_PUT_LINE','PLITBLM','DICTIONARY_OBJ_TYPE','DDLREPLICATION','DBMS_STANDARD','DBMS_APPLICATION_INFO','UTL_FILE','DDLAUX','DBMS_ASSERT'
,'STANDARD','UTL_RAW','DDLREPLICATION','UTL_FILE','DDLAUX','GGS_MARKER_SEQ','DATABASE','LOGIN_USER','FILTERDDL','DBMS_UTILITY','GGS_DDL_SEQ','
SYSEVENT','DBMS_UTILITY','LOGIN_USER','UTL_FILE','DATABASE','SDO_GEOR_DEF','UTL_RAW','GGS_DDL_SEQ','SDO_GEOR_DEF','DICTIONARY_OBJ_TYPE','UTL_R
AW','DDLREPLICATION','DBMS_UTILITY','SYSEVENT','IS_VPD_ENABLED','DBMS_APPLICATION_INFO','FILTERDDL','DDLREPLICATION','STANDARD','DDLAUX','GGS_
MARKER_SEQ','DDLAUX','SQL_TXT','PLITBLM','AW_DROP_PROC','DBMS_APPLICATION_INFO','DBMS_UTILITY','DICTIONARY_OBJ_OWNER','DICTIONARY_OBJ_NAME','S
TANDARD','DBMS_STANDARD','TRACE_PUT_LINE','UTL_FILE');
no rows selected

SQL> SELECT SID,
  2        A.BLOCKING_SESSION,
  3        EVENT,
  4        A.P1,
  5        A.P2,
  6        A.P3,
  7        CHR(BITAND(P1, -16777216) / 16777215) ||
  8        CHR(BITAND(P1, 16711680) / 65535) "LOCK",
  9        BITAND(P1, 65535) "MODE",
 10   (SELECT b.SQL_TEXT FROM v$sql b WHERE b.SQL_ID=NVL(a.sql_id,a.PREV_SQL_ID)) SQL_TEXT
 11    FROM GV$SESSION  A
 12   WHERE A.SID IN (141,22);

     SID BLOCKING_SESSION EVENT                            P1         P2         P3 LOCK       MODE SQL_TEXT
---------- ---------------- -------------------------------- ---------- ---------- ---------- ---- ---------- ----------------------------------------------------------------------------------------
      22              141 enq: TX - row lock contention    1415053316     327688       1122 TX            4 ALTER INDEX IDX_TEST_LHR REBUILD ONLINE
     141                  SQL*Net message from client      1650815232          1          0 be        28928 INSERT INTO T_INDEX_161113 SELECT * FROM T_INDEX_161113 WHERE ROWNUM<=1
```

	SID	BLOCKING_SESSION	EVENT	P1	P2	P3	LOCK	MODE	SQL_TEXT
1	22	141	enq: TX - row lock contention	1415053316	327688	1122	TX	4	ALTER INDEX IDX_TEST_LHR REBUILD ONLINE
2	141		SQL*Net message from client	1650815232	1	0	be	28928	INSERT INTO T_INDEX_161113 SELECT * FROM T_INDEX_16

可以发现在会话 1 中，在加上 ONLINE 重建索引的过程中会生成 2 个 TM 锁，锁类别分别为 2 和 4，根据查询结果发现 lmode=2 的 object_id 为 77629 的对象对应的是 T_INDEX_161113 这个表，对应的是 TM 的 Row-S (SS) 锁即行级共享锁，该锁允许其它会话对该表执行 DML 操作。另一个 lmode=4 的锁对象是 SYS_JOURNAL_77631，应该为系统临时创建的对象，对应的是 TM 的 S 锁。

在会话 2 中，TX 为 6 的锁，阻塞了其它会话，在这里其实是阻塞了会话 1 的重建索引的操作。

可以得出这样一个结论：**当对表进行创建或重建索引操作时，可以加上 ONLINE 选项，不阻塞其它会话的 DML 操作，但是在创建或重建索引的过程中，其它的会话产生的事务会阻塞索引的创建或重建操作，所以必须结束其它会话的事务才能让创建或重建索引的操作完成。**

注意：在加上 ONLINE 选项创建索引的过程中，若手动 CTRL+C 取消后，可能导致索引被锁，出现 ORA-08104: this index object 77645 is being online built or rebuilt 的错误，这个时候可以利用如下的脚本清理对象，77645 为对象的 OBJECT_ID：

```
DECLARE
  DONE BOOLEAN;
BEGIN
  DONE := DBMS_REPAIR.ONLINE_INDEX_CLEAN(77645);
END;
```

2.11.3 Oracle 10g 下 ONLINE 选项会堵塞 DML 操作

版本为：10.2.0.1.0

重新开 3 个会话，会话 1 为 143，会话 2 为 152，会话 3 为 158：

```
SYS@lhrdb S1> alter index IDX_TEST1_LHR rebuild online;



    在创建索引的同时，在会话 2 上插入一条记录：
SYS@oratest S2> INSERT INTO T_INDEX_161113 SELECT * FROM T_INDEX_161113 WHERE ROWNUM<=1;

1 row created.

====>>>>> 加上 ONLINE 后仍然会阻塞 DML 语句，若无阻塞可以重新连接会话 2 再执行插入操作
    在创建索引的同时，在会话 3 上插入一条记录：
SYS@oratest S2> INSERT INTO T_INDEX_161113 SELECT * FROM T_INDEX_161113 WHERE ROWNUM<=1;

1 row created.

====>>>>> 加上 ONLINE 后仍然会阻塞 DML 语句，若无阻塞可以重新连接会话 3 再执行插入操作

    在创建索引的同时，查询相关锁的信息：
SQL> SELECT * FROM V$LOCK A WHERE A.SID IN (143,152,158)  ORDER BY a.SID,a.TYPE;

ADDR             KADDR                 SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
---------------- ---------------- ---------- -- ---------- ---------- ---------- ---------- ---------- ----------
00000000704A7850 00000000704A7870        143 DL      53121          0          3          0        144          0
00000000704A7980 00000000704A79A0        143 DL      53121          0          3          0        144          0
00000000703B8630 00000000703B8658        143 TM      53121          0          2          4        161          0
00000000703B8730 00000000703B8758        143 TM      53156          0          4          0        161          0
000000006F49F268 000000006F49F3F0        143 TX     196651        452          6          0        159          0
00000000703B8930 00000000703B8958        152 TM      53121          0          0          3        141          0
00000000703B8830 00000000703B8858        158 TM      53121          0          3          0        153          1
000000006F45DC78 000000006F45DE00        158 TX     262170        423          6          0        153          0

8 rows selected.
SQL> SELECT * FROM V$lock_Type d WHERE d.TYPE='DL';

TYPE     NAME                                ID1_TAG      DESCRIPTION
-------- ----------------------------------- -----------  ------------
DL       Direct Loader Index Creation        object #     Lock to prevent index DDL during direct load


SQL> SELECT * FROM V$LOCK A WHERE A.SID IN (143,152,158) AND A.TYPE IN ('TX','TM') ORDER BY a.SID,a.TYPE;

ADDR             KADDR                 SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
---------------- ---------------- ---------- -- ---------- ---------- ---------- ---------- ---------- ----------
00000000703B8630 00000000703B8658        143 TM      53121          0          2          4        161          0
00000000703B8730 00000000703B8758        143 TM      53156          0          4          0        161          0
000000006F49F268 000000006F49F3F0        143 TX     196651        452          6          0        159          0
00000000703B8930 00000000703B8958        152 TM      53121          0          0          3        141          0
00000000703B8830 00000000703B8858        158 TM      53121          0          3          0        153          1
000000006F45DC78 000000006F45DE00        158 TX     262170        423          6          0        153          0

6 rows selected.
SQL> SELECT * FROM DBA_DML_LOCKS D WHERE D.SESSION_ID IN (143,152,158) ORDER BY d.SESSION_ID;

SESSION_ID OWNER                          NAME                           MODE_HELD     MODE_REQUESTE LAST_CONVERT BLOCKING_OTHERS
---------- ------------------------------ ------------------------------ ------------- ------------- ------------ ----------------------------------------
       143 SYS                            T_INDEX_161113                 Row-S (SS)    Share                  335 Not Blocking
       143 SYS                            SYS_JOURNAL_53122              Share         None                   335 Not Blocking
       152 SYS                            T_INDEX_161113                 None          Row-X (SX)             315 Blocking
       158 SYS                            T_INDEX_161113                 Row-X (SX)    None                   327 Blocking
```

```
SQL>  SELECT D.OWNER, D.OBJECT_NAME, D.OBJECT_ID, D.OBJECT_TYPE
  2        FROM DBA_OBJECTS D
  3       WHERE D.OBJECT_ID IN (53121, 53156);

OWNER     OBJECT_NAME            OBJECT_ID OBJECT_TYPE
--------- -------------------- ---------- -------------------
SYS       T_INDEX_161113              53121 TABLE
SYS       SYS_JOURNAL_53122           53156 TABLE


SQL> SELECT d.owner,d.table_name,d.iot_type  FROM dba_tables d WHERE d.table_name='SYS_JOURNAL_53122';

OWNER                          TABLE_NAME                     IOT_TYPE
------------------------------ ------------------------------ ------------
SYS                            SYS_JOURNAL_53122              IOT

SQL>
SQL> SELECT * FROM DBA_DDL_LOCKS D WHERE D.SESSION_ID IN (143,152,158) AND D.name NOT IN
('STANDARD','DICTIONARY_OBJ_OWNER','DICTIONARY_OBJ_NAME','UTL_RAW','DBMS_APPLICATION_INFO','SDO_GEOR_DEF','SQL_TXT','DBMS_ASSERT','SDO_GEOR_DEF'
,'TRACE_PUT_LINE','PLITBLM','DICTIONARY_OBJ_TYPE','DDLREPLICATION','DBMS_STANDARD','DBMS_APPLICATION_INFO','UTL_FILE','DDLAUX','DBMS_ASSERT','ST
ANDARD','UTL_RAW','DDLREPLICATION','UTL_FILE','DDLAUX','GGS_MARKER_SEQ','DATABASE','LOGIN_USER','FILTERDDL','DBMS_UTILITY','GGS_DDL_SEQ','SYSEVE
NT','DBMS_UTILITY','LOGIN_USER','UTL_FILE','DATABASE','SDO_GEOR_DEF','UTL_RAW','GGS_DDL_SEQ','SDO_GEOR_DEF','DICTIONARY_OBJ_TYPE','UTL_RAW','DDL
REPLICATION','DBMS_UTILITY','SYSEVENT','IS_VPD_ENABLED','DBMS_APPLICATION_INFO','FILTERDDL','DDLREPLICATION','STANDARD','DDLAUX','GGS_MARKER_SEQ
','DDLAUX','SQL_TXT','PLITBLM','AW_DROP_PROC','DBMS_APPLICATION_INFO','DBMS_UTILITY','DICTIONARY_OBJ_OWNER','DICTIONARY_OBJ_NAME','STANDARD','DB
MS_STANDARD','TRACE_PUT_LINE','UTL_FILE','DBMS_SYS_SQL','DBMS_XDBZ0','DBMS_SYS_SQL','DBMS_SQL','DBMS_SQL','DBMS_XDBZ0');

no rows selected
SQL> SELECT SID,
  2        A.BLOCKING_SESSION,
  3        EVENT,
  4        A.P1,
  5        A.P2,
  6        A.P3,
  7        CHR(BITAND(P1, -16777216) / 16777215) ||
  8        CHR(BITAND(P1, 16711680) / 65535) "LOCK",
  9        BITAND(P1, 65535) "MODE",
 10                  (SELECT b.SQL_TEXT FROM v$sql b WHERE b.SQL_ID=NVL(a.sql_id,a.PREV_SQL_ID)) SQL_TEXT
 11    FROM GV$SESSION  A
 12   WHERE A.SID IN (143,152,158);

       SID BLOCKING_SESSION EVENT                          P1         P2         P3 LOCK       MODE SQL_TEXT
---------- ---------------- ---------------------------- ---------- ---------- ---------- ---- ---------- ----------------------------------------------
       143              158 enq: TM - contention         1414332420      53121          0 TM            4 alter index IDX_TEST1_LHR rebuild online
       152              143 enq: TM - contention         1414332419      53121          0 TM            3  INSERT INTO T_INDEX_161113 SELECT * FROM T_INDEX_161113 WHERE ROWNUM<=1
       158                  SQL*Net message from client  1650815232          1          0 be        28928  INSERT INTO T_INDEX_161113 SELECT * FROM T_INDEX_161113 WHERE ROWNUM<=1
```

	SID	BLOCKING_SESSION	EVENT	P1	P2	P3	LOCK	MODE	SQL_TEXT
1	143	158	enq: TM - contention	1414332420	53121	0	TM	4	alter index IDX_TEST1_LHR rebuild online
2	152	143	enq: TM - contention	1414332419	53121	0	TM	3	INSERT INTO T_INDEX_161113 SELECT * FROM T_INDEX_161113 WHERE
3	158		SQL*Net message from client	1650815232	1	0	be	28928	INSERT INTO T_INDEX_161113 SELECT * FROM T_INDEX_161113 WHERE

可以发现在会话 1 中，在加上 ONLINE 重建索引的过程中会生成 2 个 TM 锁，锁类别分别为 2 和 4，根据查询结果发现 lmode=2 的 object_id 为 53121 的对象对应的是 T_INDEX_161113 这个表,对应的是 TM 的 Row-S (SS)锁即行级共享锁,该锁允许其它会话对该表执行 DML 操作,但是该会话在请求模式为 4 的 S 锁。另一个 lmode=4 的锁对象是 SYS_JOURNAL_53122，为系统临时创建的索引组织表（IOT），对应的是 TM 的 S 锁。

在会话 2 中，请求 3 级 TM 锁。会阻塞关系可以看出，会话 3 阻塞了会话 1，而会话 1 阻塞了会话 2，所以提交会话 3 即可让索引创建完成。

2.11.3.1 实验 10.2.0.1.0

版本为：10.2.0.1.0

重新开 3 个会话，会话 1 为 143，会话 2 为 152，会话 3 为 158，会话 1 插入一条记录：

```
SYS@lhrdb S1> INSERT INTO T_INDEX_161113 SELECT * FROM T_INDEX_161113 WHERE ROWNUM<=1;

1 row created.
```

在会话 2 上采用 ONLINE 建立索引：

```
SYS@lhrdb S2> alter index IDX_TEST1_LHR rebuild online;
```

====>>>>> 加上 ONLINE 后仍然会被阻塞

在创建索引的同时，查询相关锁的信息：

```
SQL> SELECT * FROM V$LOCK A WHERE A.SID IN (143,152)  ORDER BY a.SID,a.TYPE;

ADDR             KADDR                 SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
---------------- ---------------- ---------- -- ---------- ---------- ---------- ---------- ---------- ----------
00000000703B8630 00000000703B8658        143 TM      53121          0          3          0       1119          1
000000006F495E38 000000006F495FC0        143 TX     524318        484          6          0       1119          0
00000000704A7980 00000000704A79A0        152 DL      53121          0          3          0       1113          0
00000000704A7850 00000000704A7870        152 DL      53121          0          3          0       1113          0
00000000703B8730 00000000703B8758        152 TM      53121          0          2          4       1113          0
00000000703B8830 00000000703B8858        152 TM      53162          0          4          0       1112          0

6 rows selected.
SQL> SELECT * FROM V$lock_Type d WHERE d.TYPE='DL';
```

```
TYPE     NAME                              ID1_TAG      DESCRIPTION
-------- --------------------------------- -----------  ------------
DL       Direct Loader Index Creation      object #     Lock to prevent index DDL during direct load

SQL> SELECT * FROM DBA_DML_LOCKS D WHERE D.SESSION_ID IN (143,152) ORDER BY d.SESSION_ID;

SESSION_ID OWNER                         NAME                           MODE_HELD     MODE_REQUESTE LAST_CONVERT BLOCKING_OTHERS
---------- ----------------------------- ------------------------------ ------------- ------------- ------------ ----------------------------------------
       143 SYS                           T_INDEX_161113                 Row-X (SX)    None                  1176 Blocking
       152 SYS                           SYS_JOURNAL_53122              Share         None                  1169 Not Blocking
       152 SYS                           T_INDEX_161113                 Row-S (SS)    Share                 1170 Not Blocking
SQL>  SELECT D.OWNER, D.OBJECT_NAME, D.OBJECT_ID, D.OBJECT_TYPE
  2        FROM DBA_OBJECTS D
  3       WHERE D.OBJECT_ID IN (53121, 53162);

OWNER     OBJECT_NAME            OBJECT_ID OBJECT_TYPE
--------- -------------------- ---------- -------------------
SYS       T_INDEX_161113              53121 TABLE
SYS       SYS_JOURNAL_53122           53162 TABLE


SQL> SELECT d.owner,d.table_name,d.iot_type  FROM dba_tables d WHERE d.table_name='SYS_JOURNAL_53122';

OWNER                          TABLE_NAME                     IOT_TYPE
------------------------------ ------------------------------ ------------
SYS                            SYS_JOURNAL_53122              IOT

SQL>
SQL> SELECT * FROM DBA_DDL_LOCKS D WHERE D.SESSION_ID IN (143,152,158) AND D.name NOT IN
('ALERT_QUE_R','AQ$_ALERT_QT_E','AW_DROP_PROC','DATABASE','DBMS_APPLICATION_INFO','DBMS_BACKUP_RESTORE','DBMS_HA_ALERTS_PRVT','DBMS_OUTPUT','DB
MS_PRVT_TRACE','DBMS_RCVMAN','DBMS_SQL','DBMS_STANDARD','DBMS_SYS_SQL','DBMS_TRANSACTION','DBMS_UTILITY','DBMS_XDBZ0','DICTIONARY_OBJ_NAME','DI
CTIONARY_OBJ_OWNER','PLITBLM','SCHEDULER$_INSTANCE_S','STANDARD');

no rows selected
SQL> SELECT SID,
  2        A.BLOCKING_SESSION,
  3        EVENT,
  4        A.P1,
  5        A.P2,
  6        A.P3,
  7        CHR(BITAND(P1, -16777216) / 16777215) ||
  8        CHR(BITAND(P1, 16711680) / 65535) "LOCK",
  9        BITAND(P1, 65535) "MODE",
 10                   (SELECT b.SQL_TEXT FROM v$sql b WHERE b.SQL_ID=NVL(a.sql_id,a.PREV_SQL_ID)) SQL_TEXT
 11    FROM GV$SESSION  A
 12   WHERE A.SID IN (143,152);

       SID BLOCKING_SESSION EVENT                          P1         P2         P3 LOCK       MODE SQL_TEXT
---------- ---------------- ---------------------------- ---------- ---------- ---------- ---- ---------- ----------------------------------------------
       143                  SQL*Net message from client  1650815232          1          0 be         28928
       152              143 enq: TM - contention         1414332420      53121          0 TM             4 alter index IDX_TEST1_LHR rebuild online
```

	SID	BLOCKING_SESSION	EVENT	P1	P2	P3	LOCK	MODE	SQL_TEXT
1	143		SQL*Net message from client	1650815232	1	0	be	28928	
2	152	143	enq: TM - contention	1414332420	53121	0	TM	4	alter index IDX_TEST1_LHR rebuild online

从上面的结果可以知道，会话 2 即创建索引的会话一共出现了 4 个锁，两个 DL 锁，一个针对表 T_INDEX_161113 的 TM 锁，一个是 online rebuild index 时需要的一个中间表的 TM 锁，中间表用于记录 rebuild 期间的增量数据，原理类似于物化视图日志，其 object_id 为 53162，这是一个索引组织表(IOT)，从这里我们也可以发现 IOT 的优点和适合的场合，这张中间表只有插入，不会有删除和修改操作，而且只有主键条件查询，正是 IOT 最合适的场景。

会话 2 在请求一个模式为 4 的 TM 锁，模式 4 会阻塞这个表上的所有 DML 操作，所以这时再往这个表上执行 DML 也会挂起。

会话 3 删除一条语句：

```
SYS@lhrdb S3> delete from  T_INDEX_161113 where rownum<=1;



====>>>>> 有阻塞
```

查询锁的资源：

```
SQL> SELECT * FROM V$LOCK A WHERE A.SID IN (143,152,158)  ORDER BY a.SID,a.TYPE;

ADDR             KADDR                 SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
---------------- ---------------- ---------- -- ---------- ---------- ---------- ---------- ---------- ----------
00000000703B8630 00000000703B8658        143 TM      53121          0          3          0       7573          1
000000006F495E38 000000006F495FC0        143 TX     524318        484          6          0       7573          0
00000000704A7850 00000000704A7870        152 DL      53121          0          3          0       7567          0
00000000704A7980 00000000704A79A0        152 DL      53121          0          3          0       7567          0
00000000703B8830 00000000703B8858        152 TM      53162          0          4          0       7566          0
00000000703B8730 00000000703B8758        152 TM      53121          0          2          4       7567          0
00000000703B8930 00000000703B8958        158 TM      53121          0          0          3        165          0

7 rows selected.

SQL> SELECT * FROM DBA_DML_LOCKS D WHERE D.SESSION_ID IN (143,152,158) ORDER BY d.SESSION_ID;

SESSION_ID OWNER      NAME                           MODE_HELD     MODE_REQUESTE LAST_CONVERT BLOCKING_OTHERS
---------- ---------- ------------------------------ ------------- ------------- ------------ ----------------
       143 SYS        T_INDEX_161113                 Row-X (SX)    None                  7582 Blocking
       152 SYS        T_INDEX_161113                 Row-S (SS)    Share                 7576 Not Blocking
       152 SYS        SYS_JOURNAL_53122              Share         None                  7575 Not Blocking
```

```
      158 SYS       T_INDEX_161113               None         Row-X (SX)           174 Blocking

SQL> SELECT * FROM DBA_DDL_LOCKS D WHERE D.SESSION_ID IN (143,152,158) AND D.name NOT IN
('ALERT_QUE_R','AQ$_ALERT_QT_E','AW_DROP_PROC','DATABASE','DBMS_APPLICATION_INFO','DBMS_BACKUP_RESTORE','DBMS_HA_ALERTS_PRVT','DBMS_OUTPUT','D
BMS_PRVT_TRACE','DBMS_RCVMAN','DBMS_SQL','DBMS_STANDARD','DBMS_SYS_SQL','DBMS_TRANSACTION','DBMS_UTILITY','DBMS_XDBZ0','DICTIONARY_OBJ_NAME','
DICTIONARY_OBJ_OWNER','PLITBLM','SCHEDULER$_INSTANCE_S','STANDARD');

no rows selected

SQL> SELECT SID,
  2        A.BLOCKING_SESSION,
  3        EVENT,
  4        A.P1,
  5        A.P2,
  6        A.P3,
  7        CHR(BITAND(P1, -16777216) / 16777215) ||
  8        CHR(BITAND(P1, 16711680) / 65535) "LOCK",
  9        BITAND(P1, 65535) "MODE",
 10        (SELECT b.SQL_TEXT FROM v$sql b WHERE b.SQL_ID=NVL(a.sql_id,a.PREV_SQL_ID)) SQL_TEXT
 11    FROM GV$SESSION  A
 12   WHERE A.SID IN (143,152,158);

       SID BLOCKING_SESSION EVENT                             P1         P2         P3 LOCK       MODE SQL_TEXT
---------- ---------------- ------------------------------ ---------- ---------- ---------- ---- ---------- ----------------------------------------
       143                  SQL*Net message from client    1650815232          1          0 be        28928
       152              143 enq: TM - contention           1414332420      53121          0 TM            4 alter index IDX_TEST1_LHR rebuild online
       158              152 enq: TM - contention           1414332419      53121          0 TM            3 delete from  T_INDEX_161113 where rownum<=1

SQL>
```

会话 3 请求模式为 3 的 TM 锁无法获得，会话被阻塞。这是因为锁请求是需要排队的，即使会话 3 和会话 1 是可以并发的，但由于会话 2 先请求锁并进入等待队列，后来的会话 3 也只好进入队列等待。所以，如果在执行 rebuild index online 前有长事务，并且并发量比较大，则一旦执行 alter index rebuild online，可能因为长事务阻塞，可能导致系统瞬间出现大量的锁，对于压力比较大的系统，这是一个不小的风险。这是需要迅速找出导致阻塞的会话 kill 掉，rebuild index online 一旦执行，不可轻易中断，否则可能遇到 ORA-08104。

从会话级别可以看出，会话 1 阻塞了会话 2，会话 2 阻塞了会话 3，在会话 1 执行 rollback，可以发现很短时间内会话 3 也正常执行完毕，说明会话 2 持有模式 4 的 TM 锁的时间很短，然后在 rebuild online 的进行过程中，对表加的是模式为 2 的 TM 锁，所以这段时间不会阻塞 DML 操作：

回滚会话 1，然后观察锁的情况：

```
SQL> SELECT * FROM V$LOCK A WHERE A.SID IN (143,152,158)  ORDER BY a.SID,a.TYPE;

ADDR             KADDR                SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
---------------- ---------------- ---------- -- ---------- ---------- ---------- ---------- ---------- ----------
00000000704A7850 00000000704A7870        152 DL      53121          0          3          0       8219          0
00000000704A7980 00000000704A79A0        152 DL      53121          0          3          0       8219          0
00000000703B8730 00000000703B8758        152 TM      53121          0          2          4        238          0
00000000703B8830 00000000703B8858        152 TM      53162          0          4          0       8218          0
000000006FFFDEB8 000000006FFFDF18        152 TS          0    4257321          6          0        237          0
000000006F4A7558 000000006F4A76E0        152 TX     262184        426          6          0        237          0
00000000703B8930 00000000703B8958        158 TM      53121          0          3          0        238          1
000000006F45DC78 000000006F45DE00        158 TX     589824        470          6          0        238          0

8 rows selected.
```

会话 2 又开始在请求模式 4 的 TM 锁，被会话 3 阻塞!这时在会话 1 再执行 DML 操作，同样会被会话 2 阻塞，进入锁等待队列。

在会话 3 执行 rollback 或者 commit 以后，会话 2 和会话 3 都很快执行完毕。

会话 3：

```
SYS@lhrdb S3> rollback;

Rollback complete.
```

会话 2：

```
SYS@lhrdb S2> alter index IDX_TEST1_LHR rebuild online;



Index altered.

SYS@lhrdb S2> SYS@lhrdb S2> SYS@lhrdb S2>
```

从上面的试验可以发现，虽然 rebuild index online 在执行期间只持有模式 2 的 TM 锁，不会阻塞 DML 操作，但在操作的开始和结束阶段，是需要短暂的持有模式为 4 的 TM 锁的，这段会阻塞表上的所有 DML 操作。我们在做 rebuild index online 的时候，一定要在开始和结束阶段观察系统中是否有长事务的存储，对于并发量较大的系统，最严重的后果，可能在这两个关键点导致数据库产生大量锁等待，系统负载飙升，甚至宕机。

2.11.3.2 实验 11.2.0.3.0

版本为：11.2.0.3.0

开 3 个会话，会话 1 为 16，会话 2 为 27，会话 3 为 150，会话 1 删除一条记录：

```
SYS@oratest S1> delete from  T_INDEX_161113 where rownum<=1;
```

```
1 row deleted.
```

在会话 2 上采用 ONLINE 建立索引：

```
SYS@lhrdb S2> alter index IDX_TEST_LHR rebuild online;
```

====>>>>> 会话 2 挂起

在创建索引的同时，查询相关锁的信息：

```
SYS@oratest S3> SELECT * FROM V$LOCK A WHERE A.SID IN (16,27)  ORDER BY a.SID,a.TYPE;

ADDR             KADDR                 SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
---------------- ---------------- ---------- -- ---------- ---------- ---------- ---------- ---------- ----------
00000000774DA148 00000000774DA1A0         16 AE        100          0          4          0      17039          0
00007F95B6CC6C88 00007F95B6CC6CE8         16 TM      77629          0          3          0       4034          0
000000007620A7C0 000000007620A838         16 TX     131076       1126          6          0       4034          1
00000000774D9410 00000000774D9468         27 AE        100          0          4          0      18569          0
00000000774D9250 00000000774D92A8         27 DL      77629          0          3          0        115          0
00000000774DA4C8 00000000774DA520         27 DL      77629          0          3          0        115          0
00000000774DA5A8 00000000774DA600         27 OD      77631          0          6          0        115          0
00000000774D9A30 00000000774D9A88         27 OD      77629          0          4          0        115          0
00007F95B6CC6C88 00007F95B6CC6CE8         27 TM      77629          0          2          0        115          0
00007F95B6CC6C88 00007F95B6CC6CE8         27 TM      77665          0          4          0        115          0
00000000774D9090 00000000774D90E8         27 TO      68064          1          3          0      16833          0
0000000076218728 00000000762187A0         27 TX     196627       1131          6          0        115          0
00000000774D9B10 00000000774D9B68         27 TX     131076       1126          0          4        115          0

13 rows selected.

SYS@oratest S3> SELECT * FROM V$LOCK A WHERE A.SID IN (16,27) AND A.TYPE IN ('TX','TM') ORDER BY a.SID,a.TYPE;

ADDR             KADDR                 SID TY        ID1        ID2      LMODE    REQUEST      CTIME      BLOCK
---------------- ---------------- ---------- -- ---------- ---------- ---------- ---------- ---------- ----------
00007F95B6CC5588 00007F95B6CC55E8         16 TM      77629          0          3          0       4071          0
000000007620A7C0 000000007620A838         16 TX     131076       1126          6          0       4071          1
00007F95B6CC5588 00007F95B6CC55E8         27 TM      77629          0          2          0        152          0
00007F95B6CC5588 00007F95B6CC55E8         27 TM      77665          0          4          0        152          0
00000000774D9B10 00000000774D9B68         27 TX     131076       1126          0          4        152          0
0000000076218728 00000000762187A0         27 TX     196627       1131          6          0        152          0

6 rows selected.

SYS@oratest S3> SELECT * FROM V$LOCK_TYPE D WHERE D.TYPE IN ('AE','DL','OD','TO');

TYPE   NAME                           ID1_TAG         DESCRIPTION                                    ---  ----------
------ ------------------------------ --------------- ---------------------------------------------- --- ----------
DL     Direct Loader Index Creation   object #         Lock to prevent index DDL during direct load
AE     Edition Lock                   edition obj#     Prevent Dropping an edition in use
OD     Online DDLs                    object #         Lock to prevent concurrent online DDLs
TO     Temp Object                    object #         Synchronizes DDL and DML operations on a temp object

SYS@oratest S3>   SELECT D.OWNER, D.OBJECT_NAME, D.OBJECT_ID, D.OBJECT_TYPE
  2        FROM DBA_OBJECTS D
  3      WHERE D.OBJECT_ID IN (77665, 77629);

OWNER   OBJECT_NAME          OBJECT_ID OBJECT_TYPE
------- ------------------ ---------- -------------------
SYS     SYS_JOURNAL_77631       77665 TABLE
SYS     T_INDEX_161113          77629 TABLE

SYS@oratest S3>

SYS@oratest S3> SELECT * FROM DBA_DML_LOCKS D WHERE D.SESSION_ID IN (16,27) ORDER BY d.SESSION_ID;

SESSION_ID OWNER     NAME              MODE_HELD     MODE_REQUESTE LAST_CONVERT BLOCKING_OTHERS
---------- --------- ----------------- ------------- ------------- ------------ ---------------
        16 SYS       T_INDEX_161113    Row-X (SX)    None                 4093 Not Blocking
        27 SYS       SYS_JOURNAL_77631 Share         None                  174 Not Blocking
        27 SYS       T_INDEX_161113    Row-S (SS)    None                  174 Not Blocking


SYS@oratest S3> SELECT * FROM DBA_DDL_LOCKS D WHERE D.SESSION_ID IN (16,27) AND D.name NOT IN
('ALERT_QUE_R','AQ$_ALERT_QT_E','AW_DROP_PROC','DATABASE','DBMS_APPLICATION_INFO','DBMS_BACKUP_RESTORE','DBMS_HA_ALERTS_PRVT','DBMS_OUTPUT','DBMS_PRVT_TRACE','DBMS
_RCVMAN','DBMS_SQL','DBMS_STANDARD','DBMS_SYS_SQL','DBMS_TRANSACTION','DBMS_UTILITY','DBMS_XDBZ0','DICTIONARY_OBJ_NAME','DICTIONARY_OBJ_OWNER','PLITBLM','SCHEDULER
$_INSTANCE_S','STANDARD','SDO_GEOR_DEF','SQL_TXT');

no rows selected

SYS@oratest S3> SELECT SID,
  2        A.BLOCKING_SESSION,
  3        EVENT,
  4        A.P1,
  5        A.P2,
  6        A.P3,
  7        CHR(BITAND(P1, -16777216) / 16777215) ||
  8        CHR(BITAND(P1, 16711680) / 65535) "LOCK",
  9        BITAND(P1, 65535) "MODE",
 10         (SELECT b.SQL_TEXT FROM v$sql b WHERE b.SQL_ID=NVL(a.sql_id,a.PREV_SQL_ID)) SQL_TEXT
 11    FROM GV$SESSION  A
```

```
12   WHERE A.SID IN (16,27);

       SID BLOCKING_SESSION EVENT                                P1         P2         P3 LOCK       MODE SQL_TEXT
---------- ---------------- ------------------------------ ---------- ---------- ---------- ---- ---------- ----------------------------------------
        16                  SQL*Net message from client    1650815232          1          0 be        28928 delete from  T_INDEX_161113 where rownum<=1
        27               16 enq: TX - row lock contention  1415053316     131076       1126 TX            4 alter index IDX_TEST_LHR rebuild online
```

	SID	BLOCKING_SESSION	EVENT	P1	P2	P3	LOCK	MODE	SQL_TEXT
1	16		SQL*Net message from client	1650815232	1	0	be	28928	delete from T_INDEX_161113 where rownum<=1
2	27	16	enq: TX - row lock contention	1415053316	131076	1126	TX	4	alter index IDX_TEST_LHR rebuild online

可以看到会话 2 正在请求一个模式为 4 的 TX 锁，注意和 Oracle 10g 请求的 TM 锁是不一样的，而且在我们以前的概念中，TX 锁的模式都是 6，这里出现了模式 4 的 TX 锁请求，应该是 Oracle 11g 中新引入的。那么模式 4 的 TX 锁和 TM 锁有什么不同呢？我们继续前面的实验步骤：

```
SYS@oratest S3> delete from  T_INDEX_161113 where object_id=2;

16 rows deleted.
```

会话 3 的 DML 操作顺利完成，没有被阻塞。而在 10g 当中，会话 3 是会被会话 2 请求的 TM 锁所阻塞的，这一点改进是非常有意思的，这样即使 rebuid online 操作被会话 1 的长事务阻塞，其他会话的 DML 操作，只要不和会话 1 冲突，都可以继续操作，在 Oracle 10g 及以前版本中的执行 rebuild index online 而造成锁等待的风险被大大的降低了。

依次提交会话 1 和会话 3，则会话 2 成功完成。

Oracle 11g 在很多细节方面确实做了不少的优化，而且像这样的优化，对于提高系统的高可用性的好处是不言而喻的，在 Oracle 11g 中，执行 rebuild index online 的风险将比 10g 以及更老版本中小得多，因为从头至尾都不再阻塞 DML 操作了，终于可以算得上名副其实的 online 操作了。

2.11.4 利用 10704 和 10046 跟踪锁

使用 10704 事件跟踪以下四类操作并对比跟踪结果：

***create index**

***alter index rebuild**

***create index online**

***alter index rebuild online**

1、create index 与 alter index rebuild 所获取的 TM 锁完全一致

2、create index online 与 alter index rebuild online 所获取的 TM 锁、临时表完全一致

2.11.4.1 10g

版本：10.2.0.1

一、 create index

```
SQL> drop index IDX_TEST1_LHR;

Index dropped.

SQL> ALTER SESSION SET EVENTS '10704 trace name context forever,level 10';

Session altered.

SQL> ALTER SESSION SET EVENTS '10046 trace name context forever,level 12';

Session altered.

SQL> CREATE INDEX IDX_TEST1_LHR ON T_INDEX_161113(OBJECT_NAME);

Index created.

SQL>  ALTER SESSION SET EVENTS '10704 trace name context off';

Session altered.

SQL> ALTER SESSION SET EVENTS '10046 trace name context off';

Session altered.

SQL> CREATE OR REPLACE VIEW VW_SQL_TRACE_NAME_LHR AS
  2  SELECT D.VALUE || '/' || LOWER(RTRIM(I.INSTANCE, CHR(0))) || '_ora_' ||
  3         P.SPID || '.trc' TRACE_FILE_NAME
  4    FROM (SELECT P.SPID
  5            FROM V$MYSTAT M, V$SESSION S, V$PROCESS P
  6           WHERE M.STATISTIC# = '1'
  7             AND S.SID = M.SID
  8             AND P.ADDR = S.PADDR) P,
  9         (SELECT T.INSTANCE
 10            FROM V$THREAD T, V$PARAMETER V
 11           WHERE V.NAME = 'thread'
 12             AND (V.VALUE = '0' OR TO_CHAR(T.THREAD#) = V.VALUE)) I,
 13         (SELECT VALUE FROM V$PARAMETER WHERE NAME = 'user_dump_dest') D;
```

```
View created.

SQL>
SQL>
SQL>
SQL> CREATE OR REPLACE PUBLIC SYNONYM SYN_TRACENAME_LHR FOR VW_SQL_TRACE_NAME_LHR;

Synonym created.

SQL>
SQL> select * from VW_SQL_TRACE_NAME_LHR;

TRACE_FILE_NAME
------------------------------
/u01/app/oracle/admin/jiagulun/udump/jiagulun_ora_516.trc
SQL> SELECT OBJECT_NAME,
  2         OBJECT_ID,
  3         DATA_OBJECT_ID,
  4         TO_CHAR(OBJECT_ID, 'xxxxxxxxxxxx') HEX_OBJECTID,
  5         TO_CHAR(DATA_OBJECT_ID, 'xxxxxxxxxxxx') HEX_DOBJECTID
  6    FROM DBA_OBJECTS
  7   WHERE OBJECT_NAME IN ('T_INDEX_161113', 'IDX_TEST1_LHR');

OBJECT_NAME      OBJECT_ID DATA_OBJECT_ID HEX_OBJECTID  HEX_DOBJECTID
-------------- ---------- -------------- ------------ ------------
IDX_TEST1_LHR        53239          53239         cff7         cff7
T_INDEX_161113       53121          53121         cf81         cf81
```

trace 文件如下，搜字符串“cf81”：

jiagulun_ora_516.trc

1、获取 T_INDEX_161113 表 mode=3 DL 锁

```
*** 2016-11-21 16:23:57.846
ksqgtl *** DL-0000cf81-00000000 mode=3 flags=0x11 timeout=0 ***
ksqgtl: xcb=0x0x6f45dc78, ktcdix=2147483647, topxcb=0x0x6f45dc78
    ktcipt(topxcb)=0x0
```

2、获取 T_INDEX_161113 表 mode=4 TM 锁

```
*** 2016-11-21 16:23:57.847
ksqgtl *** TM-0000cf81-00000000 mode=4 flags=0x401 timeout=0 ***
ksqgtl: xcb=0x0x6f45dc78, ktcdix=2147483647, topxcb=0x0x6f45dc78
    ktcipt(topxcb)=0x0
```

3、释放 T_INDEX_161113 表 DL 锁

```
*** 2016-11-21 16:24:06.899
ksqrcl: DL,cf81,0
ksqrcl: returns 0
```

4、释放 T_INDEX_161113 表 TM 锁

```
*** 2016-11-21 16:24:06.902
ksqrcl: TM,cf81,0
ksqrcl: returns 0
```

二、 alter index ... rebuild

```
SQL> CONN / AS SYSDBA
Connected.
SQL> ALTER SESSION SET EVENTS '10704 trace name context forever,level 10';

Session altered.

SQL> ALTER SESSION SET EVENTS '10046 trace name context forever,level 12';

Session altered.

SQL> ALTER INDEX IDX_TEST1_LHR REBUILD;

Index altered.

SQL> ALTER SESSION SET EVENTS '10704 trace name context off';

Session altered.

SQL> ALTER SESSION SET EVENTS '10046 trace name context off';

Session altered.

SQL> select * from VW_SQL_TRACE_NAME_LHR;
```

```
TRACE_FILE_NAME
------------------------------
/u01/app/oracle/admin/jiagulun/udump/jiagulun_ora_1383.trc

SQL> SELECT OBJECT_NAME,
  2         OBJECT_ID,
  3         DATA_OBJECT_ID,
  4         TO_CHAR(OBJECT_ID, 'xxxxxxxxxxxx') HEX_OBJECTID,
  5         TO_CHAR(DATA_OBJECT_ID, 'xxxxxxxxxxxx') HEX_DOBJECTID
  6    FROM DBA_OBJECTS
  7   WHERE OBJECT_NAME IN ('T_INDEX_161113', 'IDX_TEST1_LHR');
OBJECT_NAME      OBJECT_ID DATA_OBJECT_ID HEX_OBJECTID  HEX_DOBJECTID
--------------- ---------- -------------- ------------ ------------
IDX_TEST1_LHR         53239          53242         cff7         cffa
T_INDEX_161113        53121          53121         cf81         cf81
```

trace 文件如下，搜字符串“cf81”：

jiagulun_ora_1383.trc

1、获取 T_INDEX_161113 表 mode=3 DL 锁

```
*** 2016-11-21 16:37:04.615
ksqgtl *** DL-0000cf81-00000000 mode=3 flags=0x11 timeout=0 ***
ksqgtl: xcb=0x0x6f45dc78, ktcdix=2147483647, topxcb=0x0x6f45dc78
    ktcipt(topxcb)=0x0
```

2、获取 T_INDEX_161113 表 mode=4 TM 锁

```
*** 2016-11-21 16:37:04.616
ksqgtl *** TM-0000cf81-00000000 mode=4 flags=0x401 timeout=0 ***
ksqgtl: xcb=0x0x6f45dc78, ktcdix=2147483647, topxcb=0x0x6f45dc78
    ktcipt(topxcb)=0x0
```

3、释放 T_INDEX_161113 表 DL 锁

```
*** 2016-11-21 16:37:09.948
ksqrcl: DL,cf81,0
ksqrcl: returns 0
```

4、释放 T_INDEX_161113 表 TM 锁

```
*** 2016-11-21 16:37:10.003
ksqrcl: TM,cf81,0
ksqrcl: returns 0
```

三、 create index ... online

```
SQL> conn / as sysdba
Connected.

SQL> drop index IDX_TEST1_LHR;

Index dropped.

SQL> ALTER SESSION SET EVENTS '10704 trace name context forever,level 10';

Session altered.

SQL> ALTER SESSION SET EVENTS '10046 trace name context forever,level 12';

Session altered.

SQL> CREATE INDEX IDX_TEST1_LHR ON T_INDEX_161113(OBJECT_NAME) ONLINE;

Index created.

SQL>  ALTER SESSION SET EVENTS '10704 trace name context off';

Session altered.

SQL> ALTER SESSION SET EVENTS '10046 trace name context off';

Session altered.

SQL> select * from VW_SQL_TRACE_NAME_LHR;

TRACE_FILE_NAME
------------------------------
/u01/app/oracle/admin/jiagulun/udump/jiagulun_ora_1915.trc

SQL> col object_name format a15
```

```
SQL> SELECT OBJECT_NAME,
  2         OBJECT_ID,
  3         DATA_OBJECT_ID,
  4         TO_CHAR(OBJECT_ID, 'xxxxxxxxxxxx') HEX_OBJECTID,
  5         TO_CHAR(DATA_OBJECT_ID, 'xxxxxxxxxxxx') HEX_DOBJECTID
  6    FROM DBA_OBJECTS
  7   WHERE OBJECT_NAME IN ('T_INDEX_161113', 'IDX_TEST_LHR');
OBJECT_NAME     OBJECT_ID DATA_OBJECT_ID HEX_OBJECTID  HEX_DOBJECTID
--------------- ---------- -------------- ------------- -------------
IDX_TEST1_LHR       53243          53243          cffb          cffb
T_INDEX_161113      53121          53121          cf81          cf81
```

trace 文件如下，搜字符串“cf81”：

jiagulun_ora_1915.trc

1、获取 T_INDEX_161113 表 mode=3 DL 锁

```
*** 2016-11-21 16:45:14.381
ksqgtl *** DL-0000cf81-00000000 mode=3 flags=0x11 timeout=0 ***
ksqgtl: xcb=0x0x6f45dc78, ktcdix=2147483647, topxcb=0x0x6f45dc78
    ktcipt(topxcb)=0x0
```

2、获取 T_INDEX_161113 表 mode=2 TM 锁

```
*** 2016-11-21 16:45:14.383
ksqgtl *** TM-0000cf81-00000000 mode=2 flags=0x401 timeout=21474836 ***
ksqgtl: xcb=0x0x6f45dc78, ktcdix=2147483647, topxcb=0x0x6f45dc78
    ktcipt(topxcb)=0x0
```

3、2 级 TM 锁转换为 4 级 TM 锁，4 级 TM 锁转换为 2 级 TM 锁

```
*** 2016-11-21 16:45:14.659
ksqcnv: TM-0000cf81,00000000 mode=4 timeout=21474836
*** 2016-11-21 16:45:14.659
ksqcmi: TM,cf81,0 mode=4 timeout=21474836
ksqcmi: returns 0
ksqcnv: RETURNS 0
*** 2016-11-21 16:45:14.659
ksqcnv: TM-0000cf81,00000000 mode=2 timeout=21474836
*** 2016-11-21 16:45:14.659
ksqcmi: TM,cf81,0 mode=2 timeout=21474836
ksqcmi: returns 0
ksqcnv: RETURNS 0
WAIT #1: nam='db file sequential read' ela= 14264 file#=1 block#=62781 blocks=1 obj#=53121 tim=1445037026096411
WAIT #1: nam='db file scattered read' ela= 19094 file#=1 block#=62913 blocks=3 obj#=53121 tim=1445037026118946
WAIT #1: nam='db file scattered read' ela= 4712 file#=1 block#=62980 blocks=5 obj#=53121 tim=1445037026125569
。。。。。。。。。。
```

4、2 级 TM 锁转换为 4 级 TM 锁

```
*** 2016-11-21 16:45:26.192
ksqcnv: TM-0000cf81,00000000 mode=4 timeout=21474836
*** 2016-11-21 16:45:26.192
ksqcmi: TM,cf81,0 mode=4 timeout=21474836
ksqcmi: returns 0
ksqcnv: RETURNS 0
```

5、释放 T_INDEX_161113 表 DL 锁

```
*** 2016-11-21 16:45:27.274
ksqrcl: DL,cf81,0
ksqrcl: returns 0
```

6、释放 T_INDEX_161113 表 TM 锁

```
*** 2016-11-21 16:45:27.393
ksqrcl: TM,cf81,0
ksqrcl: returns 0
```

四、 alter index ... rebuild online

```
SQL> conn / as sysdba
Connected.

SQL> ALTER SESSION SET EVENTS '10704 trace name context forever,level 10';

Session altered.

SQL> ALTER SESSION SET EVENTS '10046 trace name context forever,level 12';

Session altered.

SQL> ALTER INDEX IDX_TEST1_LHR REBUILD ONLINE;
```

```
Index created.

SQL>  ALTER SESSION SET EVENTS '10704 trace name context off';

Session altered.

SQL> ALTER SESSION SET EVENTS '10046 trace name context off';

Session altered.

SQL> select * from VW_SQL_TRACE_NAME_LHR;

TRACE_FILE_NAME
------------------------------
/u01/app/oracle/admin/jiagulun/udump/jiagulun_ora_3347.trc

SQL> col object_name format a15

SQL> SELECT OBJECT_NAME,
  2         OBJECT_ID,
  3         DATA_OBJECT_ID,
  4         TO_CHAR(OBJECT_ID, 'xxxxxxxxxxxx') HEX_OBJECTID,
  5         TO_CHAR(DATA_OBJECT_ID, 'xxxxxxxxxxxx') HEX_DOBJECTID
  6    FROM DBA_OBJECTS
  7   WHERE OBJECT_NAME IN ('T_INDEX_161113', 'IDX_TEST_LHR');
OBJECT_NAME      OBJECT_ID DATA_OBJECT_ID HEX_OBJECTID  HEX_DOBJECTID
--------------- ---------- -------------- ------------ ------------
IDX_TEST1_LHR        53243          53247         cffb         cfff
T_INDEX_161113       53121          53121         cf81         cf81
```

trace 文件如下，搜字符串“cf81”：

jiagulun_ora_3347.trc

1、获取 T_INDEX_161113 表 mode=3 DL 锁

```
*** 2016-11-21 17:06:23.837
ksqgtl *** DL-0000cf81-00000000 mode=3 flags=0x11 timeout=0 ***
ksqgtl: xcb=0x0x6f45dc78, ktcdix=2147483647, topxcb=0x0x6f45dc78
    ktcipt(topxcb)=0x0
```

2、获取 T_INDEX_161113 表 mode=2 TM 锁

```
PARSING IN CURSOR #1 len=40 dep=0 uid=0 oct=9 lid=0 tim=1445038265466869 hv=1374438854 ad='6c6dc948'
ALTER INDEX IDX_TEST1_LHR REBUILD ONLINE
END OF STMT
PARSE #1:c=6999,e=7057,p=0,cr=12,cu=0,mis=1,r=0,dep=0,og=1,tim=1445038265466867
*** 2016-11-21 17:06:23.838
ksqgtl *** TM-0000cf81-00000000 mode=2 flags=0x401 timeout=21474836 ***
ksqgtl: xcb=0x0x6f45dc78, ktcdix=2147483647, topxcb=0x0x6f45dc78
    ktcipt(topxcb)=0x0
```

3、2 级 TM 锁转换为 4 级 TM 锁，4 级 TM 锁转换为 2 级 TM 锁

```
*** 2016-11-21 17:06:23.937
ksqcnv: TM-0000cf81,00000000 mode=4 timeout=21474836
*** 2016-11-21 17:06:23.937
ksqcmi: TM,cf81,0 mode=4 timeout=21474836
ksqcmi: returns 0
ksqcnv: RETURNS 0
*** 2016-11-21 17:06:23.937
ksqcnv: TM-0000cf81,00000000 mode=2 timeout=21474836
*** 2016-11-21 17:06:23.937
ksqcmi: TM,cf81,0 mode=2 timeout=21474836
ksqcmi: returns 0
ksqcnv: RETURNS 0
WAIT #1: nam='db file sequential read' ela= 17434 file#=1 block#=62781 blocks=1 obj#=53121 tim=1445038265592696
WAIT #1: nam='db file scattered read' ela= 25149 file#=1 block#=62913 blocks=3 obj#=53121 tim=1445038265625891
WAIT #1: nam='db file scattered read' ela= 22659 file#=1 block#=62980 blocks=5 obj#=53121 tim=1445038265654375
WAIT #1: nam='db file sequential read' ela= 19 file#=1 block#=62984 blocks=1 obj#=53121 tim=1445038265654750
WAIT #1: nam='db file scattered read' ela= 23256 file#=1 block#=63142 blocks=2 obj#=53121 tim=1445038265680595
。。。。。。。。。。。。。
```

4、2 级 TM 锁转换为 4 级 TM 锁

```
*** 2016-11-21 17:06:31.754
ksqcnv: TM-0000cf81,00000000 mode=4 timeout=21474836
*** 2016-11-21 17:06:31.754
ksqcmi: TM,cf81,0 mode=4 timeout=21474836
ksqcmi: returns 0
ksqcnv: RETURNS 0
```

5、释放 T_INDEX_161113 表 DL 锁

```
*** 2016-11-21 17:06:32.806
ksqrcl: DL,cf81,0
```

```
ksqrcl: returns 0


    6、释放 T_INDEX_161113 表 TM 锁
*** 2016-11-21 17:06:32.976
ksqrcl: TM,cf81,0
ksqrcl: returns 0
```

2.11.4.2 11g

版本：11.2.0.3

一、 create index

```
SQL> drop index IDX_TEST_LHR;

Index dropped.

SQL> ALTER SESSION SET EVENTS '10704 trace name context forever,level 10';

Session altered.

SQL> ALTER SESSION SET EVENTS '10046 trace name context forever,level 12';

Session altered.

SQL> CREATE INDEX IDX_TEST_LHR ON T_INDEX_161113(OBJECT_NAME);

Index created.

SQL>  ALTER SESSION SET EVENTS '10704 trace name context off';

Session altered.

SQL> ALTER SESSION SET EVENTS '10046 trace name context off';

Session altered.

SQL> select value from v$diag_info where name like '%File%';

VALUE
--------------------------------------------------------------------------------
/u02/app/oracle/diag/rdbms/oratest/oratest/trace/oratest_ora_23527.trc

SQL> col object_name format a15
SQL> SELECT OBJECT_NAME,
  2         OBJECT_ID,
  3         DATA_OBJECT_ID,
  4         TO_CHAR(OBJECT_ID, 'xxxxxxxxxxxx') HEX_OBJECTID
  5    FROM DBA_OBJECTS
  6   WHERE OBJECT_NAME IN ('T_INDEX_161113', 'IDX_TEST_LHR');
OBJECT_NAME      OBJECT_ID DATA_OBJECT_ID HEX_OBJECTID
--------------- ---------- -------------- ------------
T_INDEX_161113       77629          77629        12f3d
IDX_TEST_LHR         77884          77884        1303c
```

trace 文件如下，搜字符串“12f3d”：

oratest_ora_23527.trc

```
    1、获取 T_INDEX_161113 表 mode=4  TM 锁
PARSING IN CURSOR #140411478315224 len=50 dep=1 uid=0 oct=26 lid=0 tim=1479709305055527 hv=3478035675 ad='716d5f28' sqlid='b3p9ubr7nx76v'
LOCK TABLE "T_INDEX_161113" IN SHARE MODE  NOWAIT
END OF STMT
PARSE #140411478315224:c=2000,e=3081,p=0,cr=19,cu=0,mis=1,r=0,dep=1,og=4,plh=0,tim=1479709305055527

*** 2016-11-21 14:21:45.055
ksqgtl *** TM-00012f3d-00000000 mode=4 flags=0x401 timeout=0 ***
ksqgtl: xcb=0x76273c58, ktcdix=2147483647, topxcb=0x76273c58
    ktcipt(topxcb)=0x0


    2、获取 T_INDEX_161113 表 mode=3  DL 锁
*** 2016-11-21 14:21:45.056
ksqgtl *** DL-00012f3d-00000000 mode=3 flags=0x10001 timeout=0 ***
ksqgtl: xcb=0x76273c58, ktcdix=2147483647, topxcb=0x76273c58
    ktcipt(topxcb)=0x0


    3、释放 T_INDEX_161113 表 DL 锁
*** 2016-11-21 14:21:50.392
```

```
ksqrcl: DL,12f3d,0
ksqrcl: returns 0
```

4、释放 T_INDEX_161113 表 TM 锁

```
*** 2016-11-21 14:21:50.395
ksqrcl: TM,12f3d,0
ksqrcl: returns 0
```

二、 alter index ... rebuild

```
SQL> CONN / AS SYSDBA
Connected.
SQL> ALTER SESSION SET EVENTS '10704 trace name context forever,level 10';

Session altered.

SQL> ALTER SESSION SET EVENTS '10046 trace name context forever,level 12';

Session altered.

SQL> ALTER INDEX IDX_TEST_LHR REBUILD;

Index altered.

SQL> ALTER SESSION SET EVENTS '10704 trace name context off';

Session altered.

SQL> ALTER SESSION SET EVENTS '10046 trace name context off';

Session altered.

SQL> select value from v$diag_info where name like '%File%';

VALUE
--------------------------------------------------------------------------------
/u02/app/oracle/diag/rdbms/oratest/oratest/trace/oratest_ora_23540.trc

SQL> col object_name format a15
SQL> SELECT OBJECT_NAME,
  2         OBJECT_ID,
  3         DATA_OBJECT_ID,
  4         TO_CHAR(OBJECT_ID, 'xxxxxxxxxxxx') HEX_OBJECTID
  5    FROM DBA_OBJECTS
  6   WHERE OBJECT_NAME IN ('T_INDEX_161113', 'IDX_TEST_LHR');

SQL> SELECT OBJECT_NAME,
  2         OBJECT_ID,
  3         DATA_OBJECT_ID,
  4         TO_CHAR(OBJECT_ID, 'xxxxxxxxxxxx') HEX_OBJECTID,
  5         TO_CHAR(DATA_OBJECT_ID, 'xxxxxxxxxxxx') HEX_DOBJECTID
  6    FROM DBA_OBJECTS
  7   WHERE OBJECT_NAME IN ('T_INDEX_161113', 'IDX_TEST_LHR');

OBJECT_NAME      OBJECT_ID DATA_OBJECT_ID HEX_OBJECTID  HEX_DOBJECTID
--------------- ---------- -------------- ------------- -------------
T_INDEX_161113       77629          77629         12f3d         12f3d
IDX_TEST_LHR         77885          77886         1303d         1303e
```

trace 文件如下，搜字符串“12f3d”：

oratest_ora_23540.trc

1、获取 T_INDEX_161113 表 mode=4 TM 锁

```
PARSING IN CURSOR #140719831671200 len=59 dep=1 uid=0 oct=26 lid=0 tim=1479709686366785 hv=3620741631 ad='7176cbc8' sqlid='chctu03bx08gz'
LOCK TABLE  FOR INDEX "IDX_TEST_LHR" IN SHARE MODE  NOWAIT
END OF STMT
PARSE #140719831671200:c=10999,e=29442,p=2,cr=80,cu=0,mis=1,r=0,dep=1,og=4,plh=0,tim=1479709686366785

*** 2016-11-21 14:28:06.366
ksqgtl *** TM-00012f3d-00000000 mode=4 flags=0x401 timeout=0 ***
ksqgtl: xcb=0x76209db0, ktcdix=2147483647, topxcb=0x76209db0
    ktcipt(topxcb)=0x0
```

2、获取 T_INDEX_161113 表 mode=3 DL 锁

```
*** 2016-11-21 14:28:06.370
ksqgtl *** DL-00012f3d-00000000 mode=3 flags=0x10001 timeout=0 ***
ksqgtl: xcb=0x76209db0, ktcdix=2147483647, topxcb=0x76209db0
    ktcipt(topxcb)=0x0
```

3、释放 T_INDEX_161113 表 DL 锁

```
*** 2016-11-21 14:28:10.938
```

```
ksqrcl: DL,12f3d,0
ksqrcl: returns 0
```

4、释放 T_INDEX_161113 表 TM 锁

```
*** 2016-11-21 14:28:10.947
ksqrcl: TM,12f3d,0
ksqrcl: returns 0
```

三、 create index ... online

```
SQL> conn / as sysdba
Connected.

SQL> drop index IDX_TEST_LHR;

Index dropped.

SQL> ALTER SESSION SET EVENTS '10704 trace name context forever,level 10';

Session altered.

SQL> ALTER SESSION SET EVENTS '10046 trace name context forever,level 12';

Session altered.

SQL> CREATE INDEX IDX_TEST_LHR ON T_INDEX_161113(OBJECT_NAME) ONLINE;

Index created.

SQL>  ALTER SESSION SET EVENTS '10704 trace name context off';

Session altered.

SQL> ALTER SESSION SET EVENTS '10046 trace name context off';

Session altered.

SQL> select value from v$diag_info where name like '%File%';

VALUE
--------------------------------------------------------------------------------
/u02/app/oracle/diag/rdbms/oratest/oratest/trace/oratest_ora_23672.trc

SQL> col object_name format a15
SQL> SELECT OBJECT_NAME,
  2         OBJECT_ID,
  3         DATA_OBJECT_ID,
  4         TO_CHAR(OBJECT_ID, 'xxxxxxxxxxxx') HEX_OBJECTID,
  5         TO_CHAR(DATA_OBJECT_ID, 'xxxxxxxxxxxx') HEX_DOBJECTID
  6    FROM DBA_OBJECTS
  7   WHERE OBJECT_NAME IN ('T_INDEX_161113', 'IDX_TEST_LHR');
OBJECT_NAME     OBJECT_ID DATA_OBJECT_ID HEX_OBJECTID  HEX_DOBJECTID
--------------- --------- -------------- ------------ ------------
T_INDEX_161113      77629          77629         12f3d         12f3d
IDX_TEST_LHR        77887          77887         1303f         1303f
```

trace 文件如下，搜字符串“12f3d”：

oratest_ora_23672.trc

1、获取 T_INDEX_161113 表 mode=2 TM 锁

```
*** 2016-11-21 15:14:44.397
ksqrcl: CU,717dfd90,0
ksqrcl: returns 0
=====================
PARSING IN CURSOR #140118279700704 len=46 dep=1 uid=0 oct=26 lid=0 tim=1479712484397029 hv=3395312659 ad='729e1628' sqlid='g95cs0g560r0m'
LOCK TABLE "T_INDEX_161113" IN ROW SHARE MODE
END OF STMT
PARSE #140118279700704:c=1999,e=1893,p=0,cr=19,cu=0,mis=1,r=0,dep=1,og=4,plh=0,tim=1479712484397029

*** 2016-11-21 15:14:44.397
ksqgtl *** TM-00012f3d-00000000 mode=2 flags=0x401 timeout=21474836 ***
ksqgtl: xcb=0x761eac90, ktcdix=2147483647, topxcb=0x761eac90
    ktcipt(topxcb)=0x0
*** 2016-11-21 14:21:45.055
```

2、获取 T_INDEX_161113 表 mode=3 DL 锁

```
*** 2016-11-21 15:14:44.398
ksqgtl *** DL-00012f3d-00000000 mode=3 flags=0x10001 timeout=0 ***
ksqgtl: xcb=0x761eac90, ktcdix=2147483647, topxcb=0x761eac90
    ktcipt(topxcb)=0x0
```

3、获取 T_INDEX_161113 表 mode=4 OD 锁

```
*** 2016-11-21 15:14:44.454
ksqgtl *** OD-00012f3d-00000000 mode=4 flags=0x10401 timeout=0 ***
ksqgtl: xcb=0x761eac90, ktcdix=2147483647, topxcb=0x761eac90
    ktcipt(topxcb)=0x0
```

4、释放 T_INDEX_161113 表 DL 锁

```
*** 2016-11-21 15:14:53.066
ksqrcl: DL,12f3d,0
ksqrcl: returns 0
```

5、释放 T_INDEX_161113 表 OD、TM 锁

```
*** 2016-11-21 15:14:55.327
ksqrcl: OD,12f3d,0
ksqrcl: returns 0

*** 2016-11-21 15:14:55.327
ksqrcl: TM,12f3d,0
ksqrcl: returns 0
```

四、 alter index ... rebuild online

```
SQL> conn / as sysdba
Connected.

SQL> ALTER SESSION SET EVENTS '10704 trace name context forever,level 10';

Session altered.

SQL> ALTER SESSION SET EVENTS '10046 trace name context forever,level 12';

Session altered.

SQL> ALTER INDEX IDX_TEST_LHR REBUILD ONLINE;

Index created.

SQL>  ALTER SESSION SET EVENTS '10704 trace name context off';

Session altered.

SQL> ALTER SESSION SET EVENTS '10046 trace name context off';

Session altered.

SQL> select value from v$diag_info where name like '%File%';

VALUE
--------------------------------------------------------------------------------
/u02/app/oracle/diag/rdbms/oratest/oratest/trace/oratest_ora_23792.trc

SQL> col object_name format a15

SQL> SELECT OBJECT_NAME,
  2         OBJECT_ID,
  3         DATA_OBJECT_ID,
  4         TO_CHAR(OBJECT_ID, 'xxxxxxxxxxxx') HEX_OBJECTID,
  5         TO_CHAR(DATA_OBJECT_ID, 'xxxxxxxxxxxx') HEX_DOBJECTID
  6    FROM DBA_OBJECTS
  7   WHERE OBJECT_NAME IN ('T_INDEX_161113', 'IDX_TEST_LHR');

OBJECT_NAME      OBJECT_ID DATA_OBJECT_ID HEX_OBJECTID  HEX_DOBJECTID
--------------- ---------- -------------- ------------ ------------
T_INDEX_161113       77629          77629        12f3d        12f3d
IDX_TEST_LHR         77887          77890        1303f        13042
```

trace 文件如下，搜字符串“12f3d”：

oratest_ora_23792.trc

1、获取 T_INDEX_161113 表 mode=2 TM 锁

```
PARSING IN CURSOR #139909890400672 len=55 dep=1 uid=0 oct=26 lid=0 tim=1479715165881556 hv=1263262788 ad='7167d4f8' sqlid='6dh4ubt5nrr24'
LOCK TABLE  FOR INDEX "IDX_TEST_LHR" IN ROW SHARE MODE
END OF STMT
PARSE #139909890400672:c=1000,e=1599,p=0,cr=8,cu=0,mis=1,r=0,dep=1,og=4,plh=0,tim=1479715165881555

*** 2016-11-21 15:59:25.881
ksqgtl *** TM-00012f3d-00000000 mode=2 flags=0x401 timeout=21474836 ***
ksqgtl: xcb=0x76228ed0, ktcdix=2147483647, topxcb=0x76228ed0
    ktcipt(topxcb)=0x0
```

```
    2、获取 T_INDEX_161113 表 mode=3 DL 锁
*** 2016-11-21 15:59:25.883
ksqgtl *** DL-00012f3d-00000000 mode=3 flags=0x10001 timeout=0 ***
ksqgtl: xcb=0x76228ed0, ktcdix=2147483647, topxcb=0x76228ed0
    ktcipt(topxcb)=0x0

    3、获取 T_INDEX_161113 表 mode=4 OD 锁
*** 2016-11-21 15:59:25.884
ksqgtl *** OD-00012f3d-00000000 mode=4 flags=0x10401 timeout=0 ***
ksqgtl: xcb=0x76228ed0, ktcdix=2147483647, topxcb=0x76228ed0
    ktcipt(topxcb)=0x0

    4、释放 T_INDEX_161113 表 DL 锁
*** 2016-11-21 15:59:30.334
ksqrcl: DL,12f3d,0
ksqrcl: returns 0

    5、释放 T_INDEX_161113 表 OD、TM 锁
*** 2016-11-21 15:59:30.363
ksqrcl: OD,12f3d,0
ksqrcl: returns 0

*** 2016-11-21 15:59:30.363
ksqrcl: OD,1303f,0
ksqrcl: returns 0

*** 2016-11-21 15:59:30.363
ksqrcl: TM,12f3d,0
ksqrcl: returns 0
```

2.11.4.3 实验 SQL

```
ALTER SESSION SET EVENTS '10704 trace name context forever,level 10';
ALTER SESSION SET EVENTS '10046 trace name context forever,level 12';
--CREATE INDEX IDX_TEST_LHR ON T_INDEX_161113(OBJECT_NAME) ;
--ALTER INDEX IDX_TEST_LHR REBUILD;
--CREATE INDEX IDX_TEST_LHR ON T_INDEX_161113(OBJECT_NAME) ONLINE;
ALTER INDEX IDX_TEST_LHR REBUILD;
ALTER SESSION SET EVENTS '10704 trace name context off';
ALTER SESSION SET EVENTS '10046 trace name context off';

SELECT OBJECT_NAME,
       OBJECT_ID,
       DATA_OBJECT_ID,
       TO_CHAR(OBJECT_ID, 'xxxxxxxxxxxx') HEX_OBJECTID,
       TO_CHAR(DATA_OBJECT_ID, 'xxxxxxxxxxxx') HEX_DOBJECTID
  FROM DBA_OBJECTS
 WHERE OBJECT_NAME IN ('T_INDEX_161113', 'IDX_TEST1_LHR');

select value from v$diag_info where name like '%File%';
```

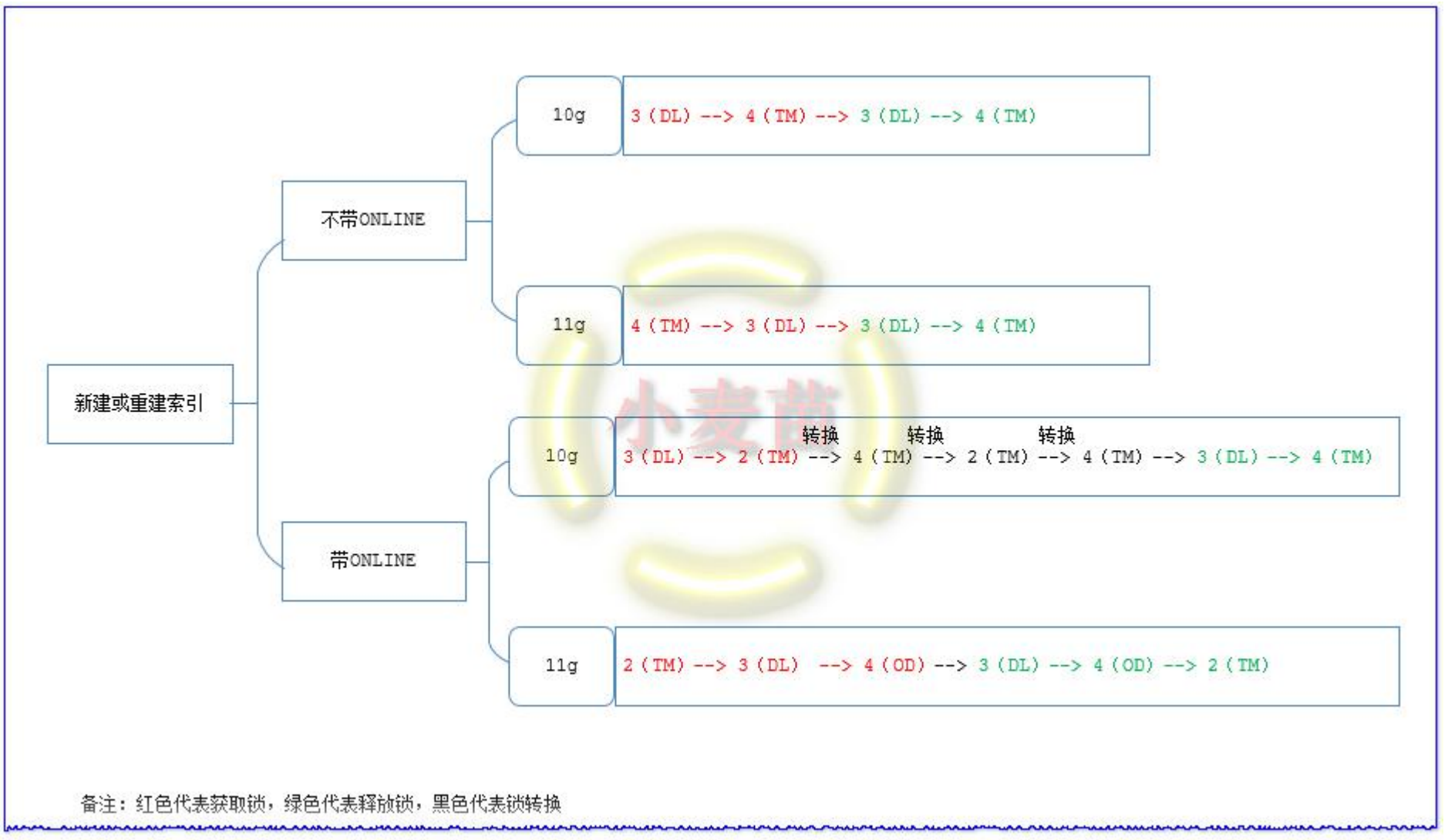
运行如下 SQL 来创建视图：

```
CREATE OR REPLACE VIEW VW_SQL_TRACE_NAME_LHR AS
SELECT D.VALUE || '/' || LOWER(RTRIM(I.INSTANCE, CHR(0))) || '_ora_' ||
       P.SPID || '.trc' TRACE_FILE_NAME
  FROM (SELECT P.SPID
          FROM V$MYSTAT M, V$SESSION S, V$PROCESS P
         WHERE M.STATISTIC# = '1'
           AND S.SID = M.SID
           AND P.ADDR = S.PADDR) P,
       (SELECT T.INSTANCE
          FROM V$THREAD T, V$PARAMETER V
         WHERE V.NAME = 'thread'
           AND (V.VALUE = '0' OR TO_CHAR(T.THREAD#) = V.VALUE)) I,
       (SELECT VALUE FROM V$PARAMETER WHERE NAME = 'user_dump_dest') D;
```

创建公共同义词：

```
CREATE OR REPLACE PUBLIC SYNONYM SYN_TRACENAME_LHR FOR VW_SQL_TRACE_NAME_LHR;
```

2.11.5 总结



不带 ONLINE 的新建或重建索引的 SQL 语句获取的是 4 级 TM 锁，它会阻塞任何 DML 操作。

在 Oracle 10g 中，带 ONLINE 的新建或重建索引的 SQL 语句在开始和结束的时候获取的是 4 级 TM 锁，而在读取表数据的过程中获取的是 2 级 TM 锁，所以，在 Oracle 10g 中，即使加上 ONLINE 也会阻塞其它会话的 DML 操作。

在 Oracle 11g 中，带 ONLINE 的新建或重建索引的 SQL 语句在整个执行过程中获取的是 2 级 TM 锁，并不会阻塞其它会话的 DML 操作，但是在创建或重建索引的过程中，其它的会话产生的事务会阻塞索引的创建或重建操作，所以必须结束其它会话的事务才能让创建或重建索引的操作完成。

在 Oracle 11g 加上 ONLINE 的情况下：

(1) 过程中会持有 OD(ONLINE DDL)、DL(Direct Loader Index Creation)两种类型的锁，在 Oracle 10g 下只有 DL 锁没有 OD 锁

(2) 表级锁 TM 的持有模式为 row-S (SS)，与 row-X (SX)类型的锁互相兼容，因此不会在表级发生阻塞

(3) 阻塞发生在行级锁申请阶段，即请求的 share(S)类型的锁与执行 DML 的 session 已经持有的 exclusive(X)锁之间存在不兼容的情况；相比非 online 方式的表级锁，锁的粒度上更加细化，副作用更小

(4) 新增以 SYS_JOURNAL_为前缀的 IOT 表，记录与索引创建动作同时进行的其它 DML 操作修改过的记录，等到索引创建完成前将 IOT 表里的记录合并至索引中并删除 IOT 表

2.12 锁用到的 SQL 语句

```
SELECT * FROM V$LOCK A WHERE A.SID IN (16,27) AND A.TYPE IN ('TX','TM') ORDER BY a.SID,a.TYPE;
SELECT * FROM V$LOCK A WHERE A.SID IN (16,27)  ORDER BY a.SID,a.TYPE;
SELECT * FROM DBA_DML_LOCKS D WHERE D.SESSION_ID IN (16,27) ORDER BY d.SESSION_ID;
SELECT * FROM DBA_DDL_LOCKS D WHERE D.SESSION_ID IN (16,27) AND D.name NOT IN
('ALERT_QUE_R','AQ$_ALERT_QT_E','AW_DROP_PROC','DATABASE','DBMS_APPLICATION_INFO','DBMS_BACKUP_RESTORE','DBMS_HA_ALERT
S_PRVT','DBMS_OUTPUT','DBMS_PRVT_TRACE','DBMS_RCVMAN','DBMS_SQL','DBMS_STANDARD','DBMS_SYS_SQL','DBMS_TRANSACTION','DB
MS_UTILITY','DBMS_XDBZ0','DICTIONARY_OBJ_NAME','DICTIONARY_OBJ_OWNER','PLITBLM','SCHEDULER$_INSTANCE_S','STANDARD','SD
O_GEOR_DEF','SQL_TXT');
SELECT A.TADDR,
       A.LOCKWAIT,
       A.ROW_WAIT_OBJ#,
       A.ROW_WAIT_FILE#,
       A.ROW_WAIT_BLOCK#,
       A.ROW_WAIT_ROW#,
       (SELECT D.OWNER || '|' || D.OBJECT_NAME || '|' || D.OBJECT_TYPE
          FROM DBA_OBJECTS D
         WHERE D.OBJECT_ID = A.ROW_WAIT_OBJ#) OBJECT_NAME,
       A.EVENT,
       A.P1,
       A.P2,
           A.P3,
       CHR(BITAND(P1, -16777216) / 16777215) ||
       CHR(BITAND(P1, 16711680) / 65535) "LOCK",
```

```
       BITAND(P1, 65535) "MODE",
       TRUNC(P2 / POWER(2, 16)) AS XIDUSN,
       BITAND(P2, TO_NUMBER('FFFF', 'XXXX')) + 0 AS XIDSLOT,
       P3 XIDSQN,
       A.SID,
       A.BLOCKING_SESSION,
       A.SADDR,
       DBMS_ROWID.ROWID_CREATE(1, 77669, 8, 2799, 0) REQUEST_ROWID,
       (SELECT B.SQL_TEXT
          FROM V$SQL B
         WHERE B.SQL_ID = NVL(A.SQL_ID, A.PREV_SQL_ID)) SQL_TEXT
  FROM V$SESSION A
 WHERE A.SID IN (143);

SELECT * FROM v$lock a WHERE a.KADDR='000000007620A7C0';
SELECT * FROM v$transaction a WHERE a.ADDR='000000007620A7C0';

SELECT * FROM V$LOCK_TYPE D WHERE D.TYPE IN ('AE','DL','OD','TO','TX');
SELECT D.OWNER, D.OBJECT_NAME, D.OBJECT_ID, D.OBJECT_TYPE
       FROM DBA_OBJECTS D
    WHERE D.OBJECT_ID IN (77665, 77629);

SELECT D.PARAMETER1,D.PARAMETER2,D.PARAMETER3 FROM V$EVENT_NAME  D WHERE D.NAME='enq: TX - row lock contention';
```

About Me

- 本文作者：小麦苗，只专注于数据库的技术，更注重技术的运用
- 本文在 itpub（http://blog.itpub.net/26736162）、博客园（http://www.cnblogs.com/lhrbest）和个人微信公众号（xiaomaimiaolhr）上有同步更
- 本文 itpub 地址：http://www.cnblogs.com/lhrbest/p/6091277.html
- 本文博客园地址：http://www.cnblogs.com/lhrbest/p/6091277.html
- 本文 pdf 版及小麦苗云盘地址：http://blog.itpub.net/26736162/viewspace-1624453/
- QQ 群：**230161599**　　微信群：私聊
- 联系我请加 QQ 好友(642808185)，注明添加缘由
- 于 2016-10-21 09:00 ~ 2016-11-22 22:00 在泰兴公寓完成
- 文章内容来源于小麦苗的学习笔记，部分整理自网络，若有侵权或不当之处还请谅解
- 版权所有，欢迎分享本文，转载请保留出处

手机长按下图识别二维码或微信客户端扫描下边的二维码来关注小麦苗的微信公众号：xiaomaimiaolhr,免费学习最实用的数据库技术。