



大型数据库应用技术

6-Oracle控制与安全

授课教师：王欢



□ Oracle数据库控制

□ 事务

□ 锁

□ Oracle数据库安全管理

□ 用户管理

□ 权限管理

□ 角色管理

□ 其它安全功能

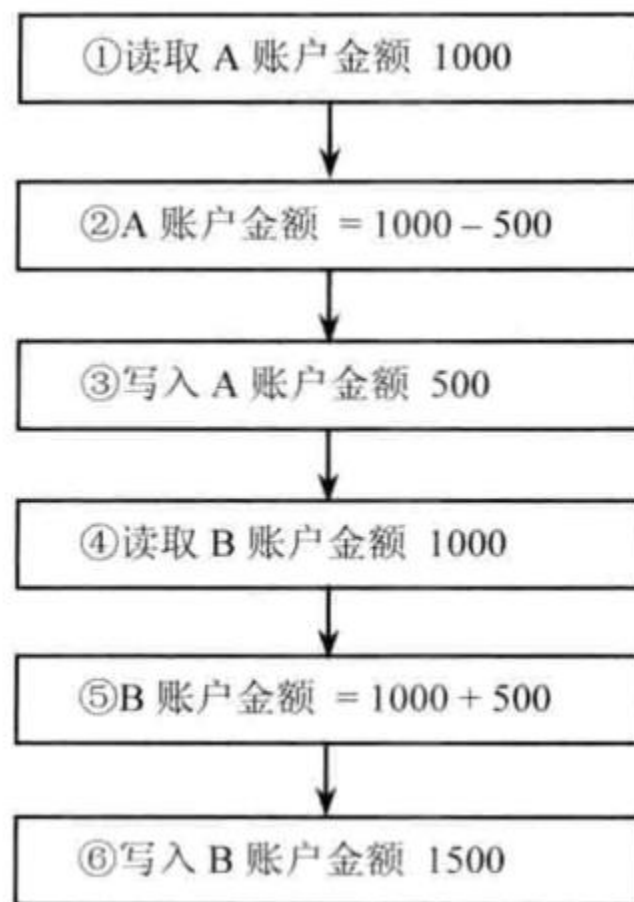


□ 基本概念

事务 (Transaction) 是由一系列语句构成的逻辑工作单元，通常是为了完成一定的业务逻辑而将一条或多条语句“封装”起来，是它们与其它语句之间存在一个逻辑上的边界。事务中的语句要么全部执行，完成整个工作单元的操作。要么全部不执行。

事务的提出主要是为了解决并发情况下保持数据一致性的问题。

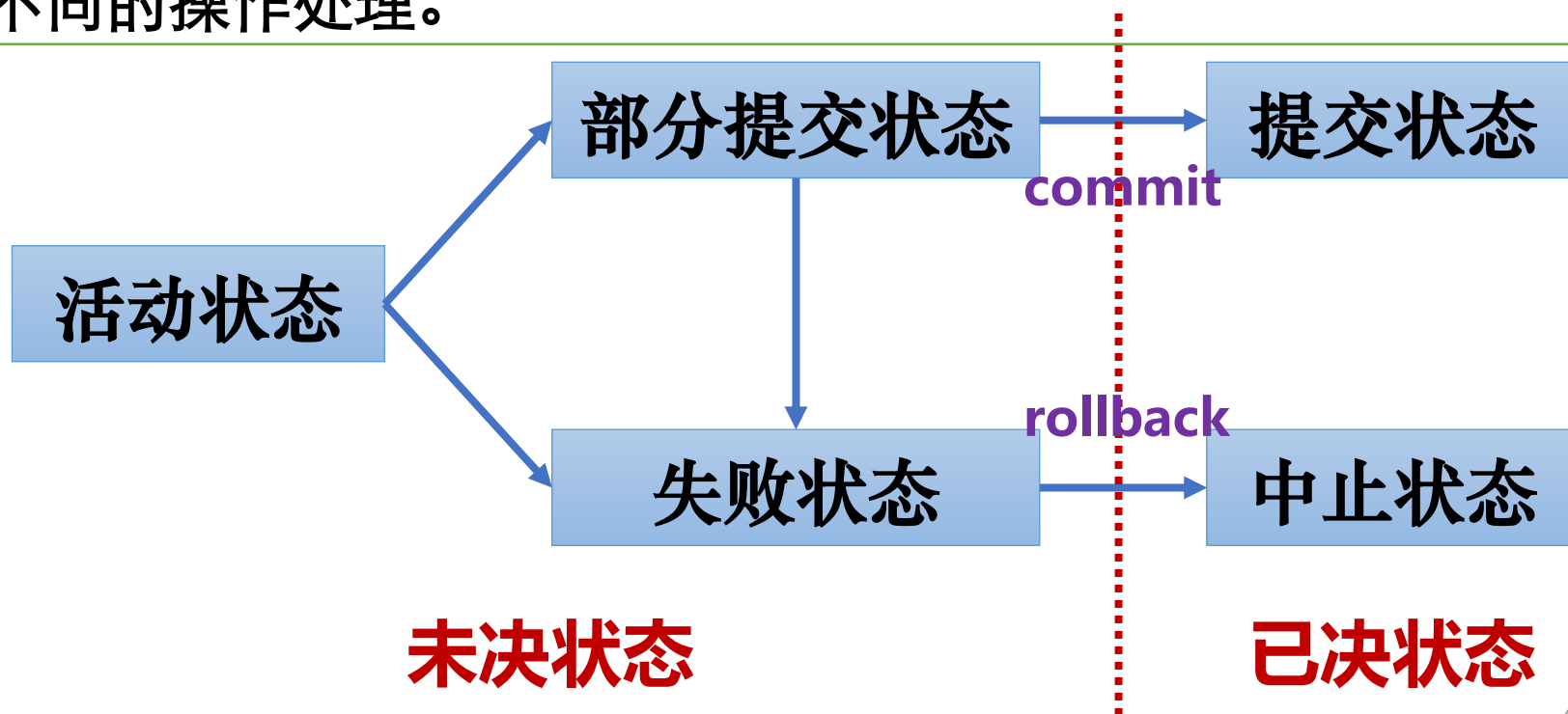
转账事务处理流程
初始A=1000, B=1000
A给B转账500





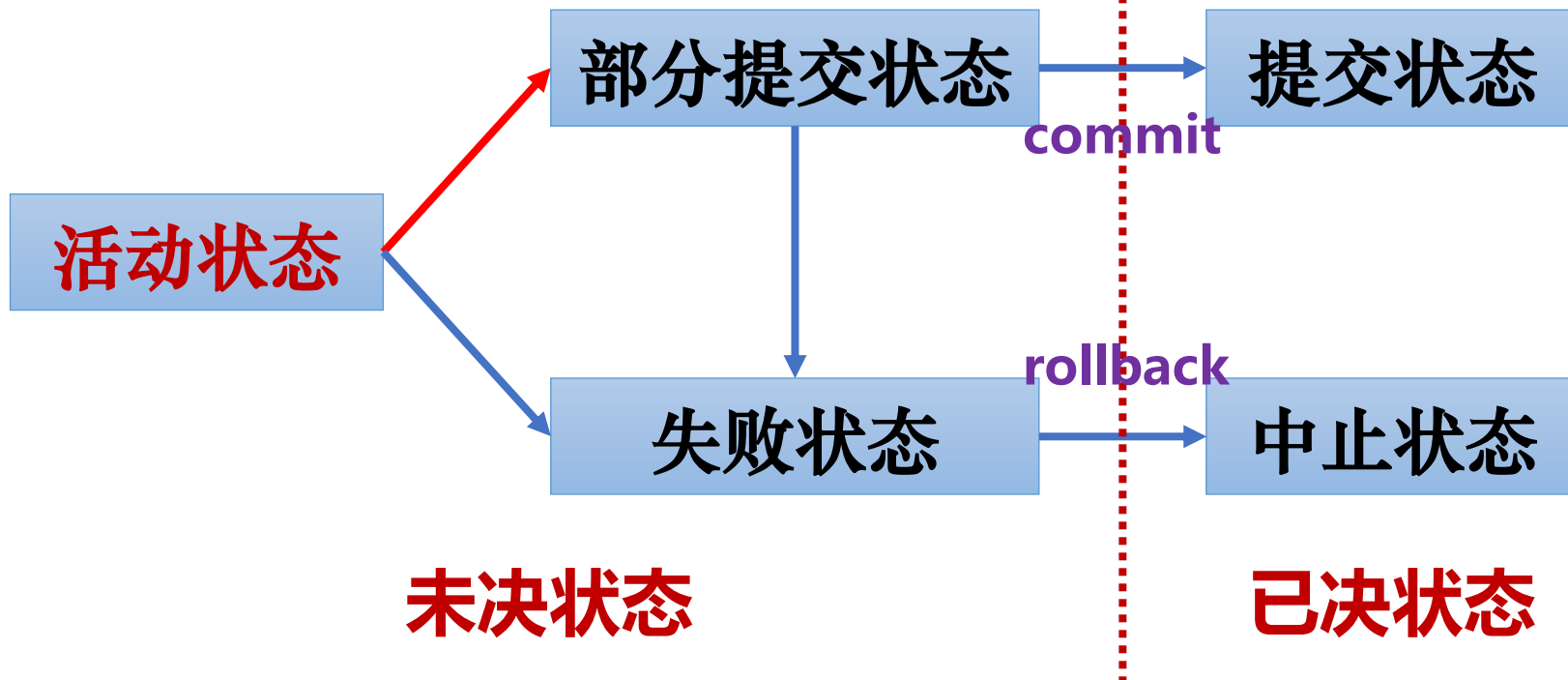
□ 事务的状态

每个事务从开始到结束的整个生命周期可以分为以下几种状态，DBMS记录每个事务的生命周期状态，以便恢复时进行不同的操作处理。





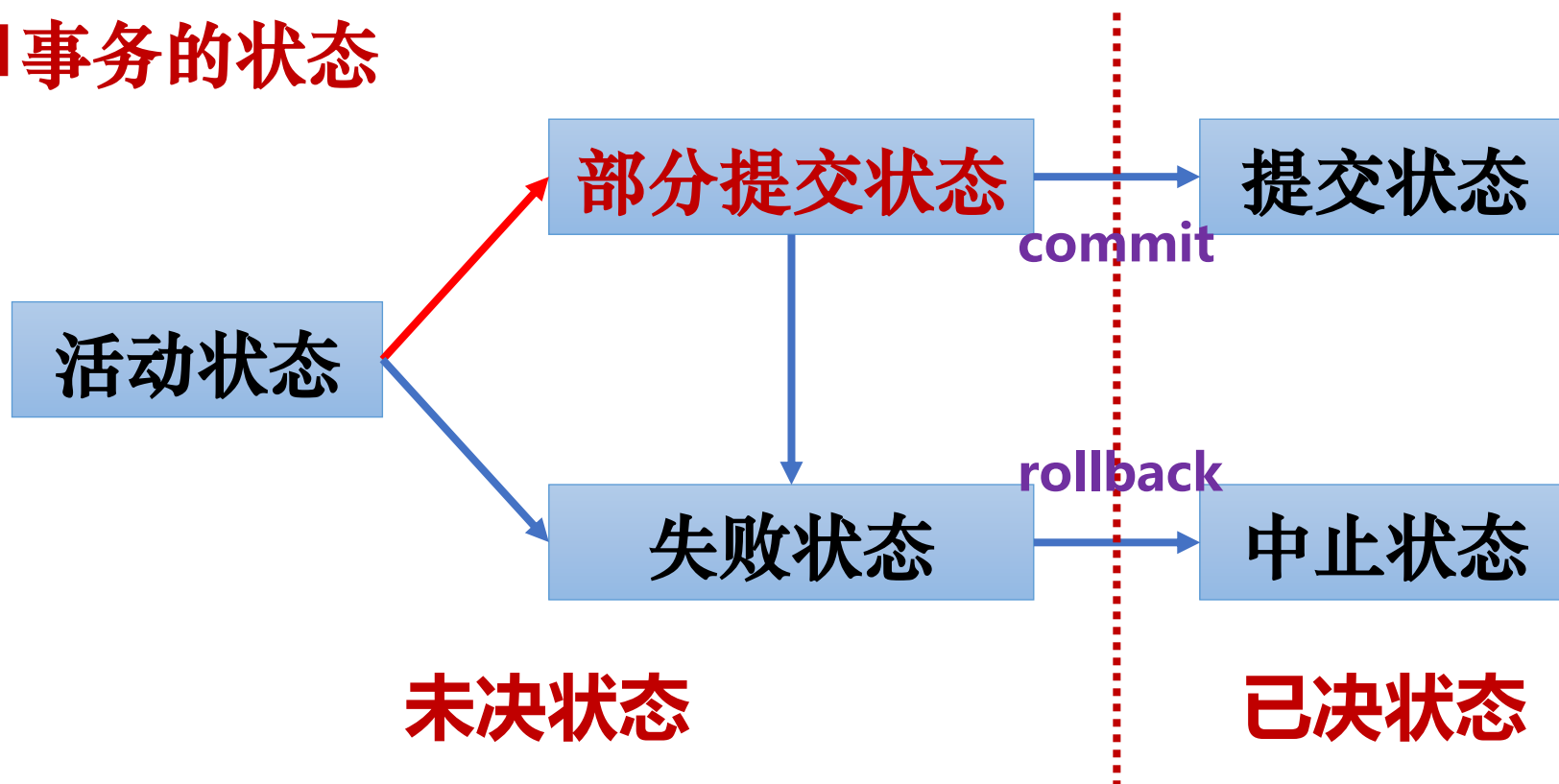
□ 事务的状态



活动状态： 事务在执行时的状态。



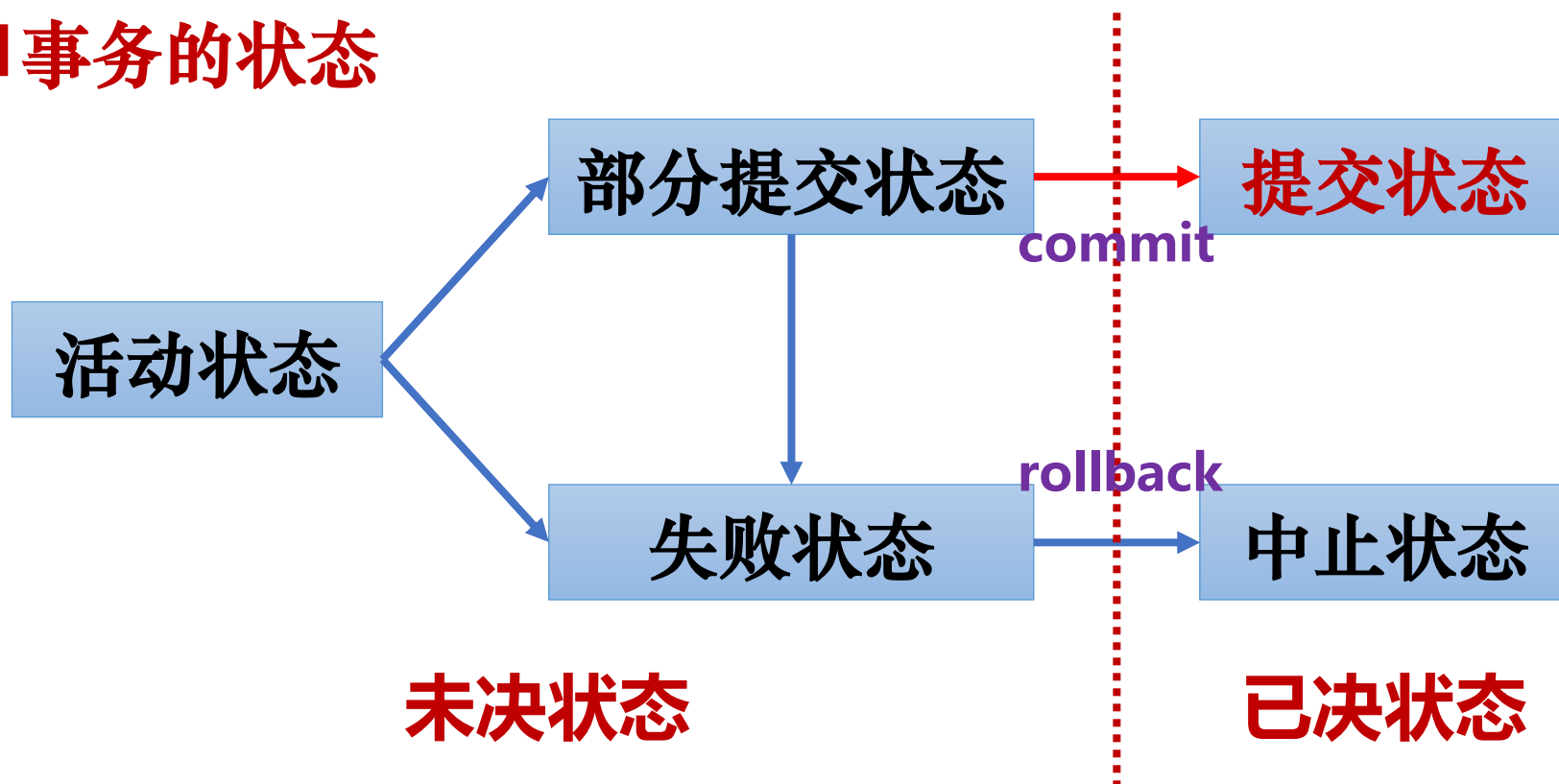
□ 事务的状态



部分提交状态：事务中最后一条语句被执行后的状态。事务虽然已经完成，但由于实际输出可能在内存中，未被写回，可能还会发生硬件故障而失败，并进入中止状态。



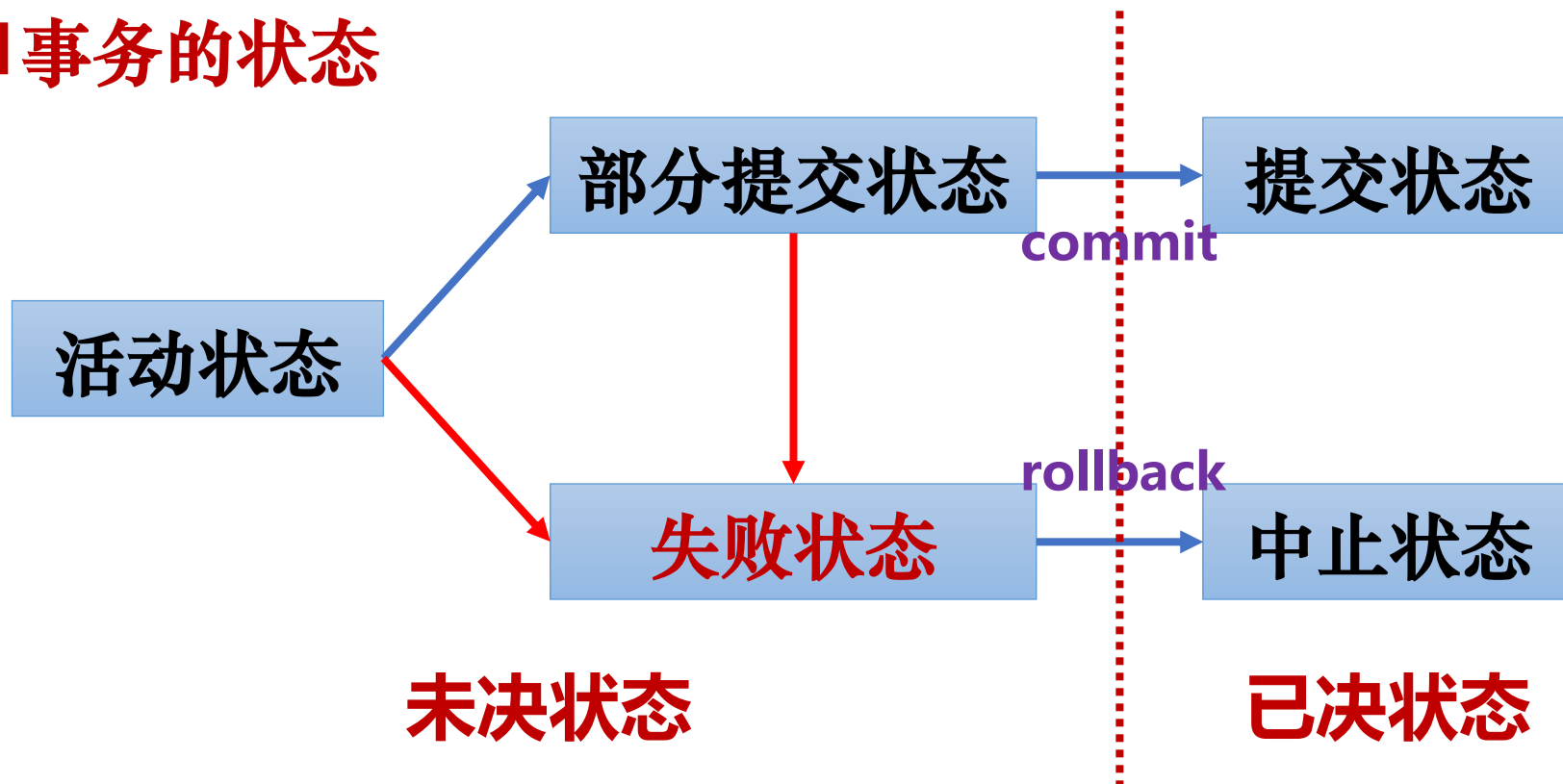
□ 事务的状态



提交状态：事务在部分提交后，将往磁盘中写入数据，最后一条信息写入后的状态称为提交状态，进入提交状态的事务就成功完成了。



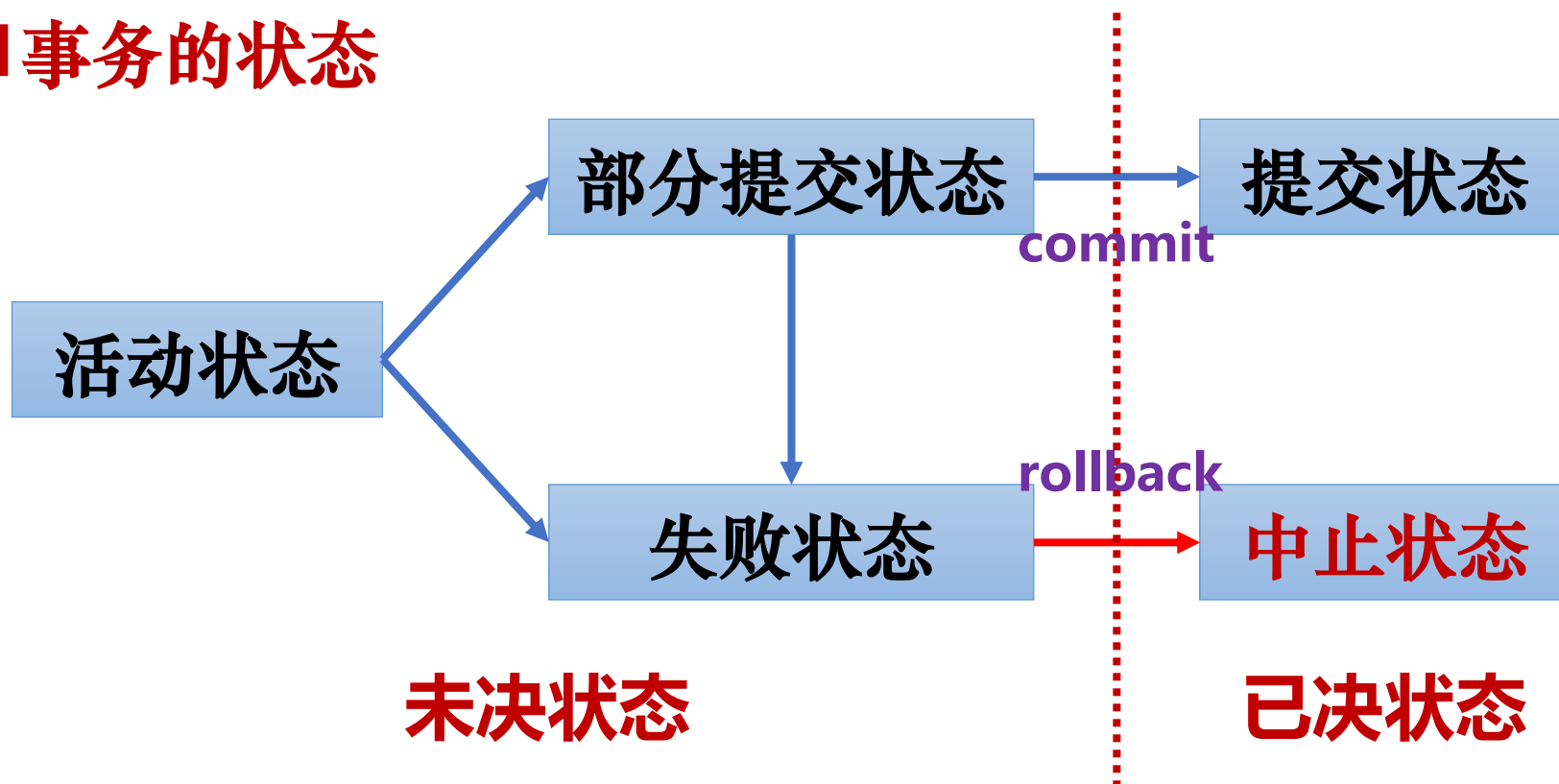
□ 事务的状态



失败状态：事务不能正常执行的状态。导致失败状态的发生可能原因有硬件问题或逻辑错误，这时事务必须回滚，进入中止状态。



□ 事务的状态



中止状态: 事务回滚，并且数据库已经恢复到事务开始执行前的状态。



□事务的四个特性 (ACID)

为了确保数据库共享访问的数据正确性，要求DBMS的事务管理机制维护事务的ACID特性。

- 原子性 (Atomic)
- 一致性 (Consistent)
- 隔离性 (Isolated)
- 持久性 (Durability)

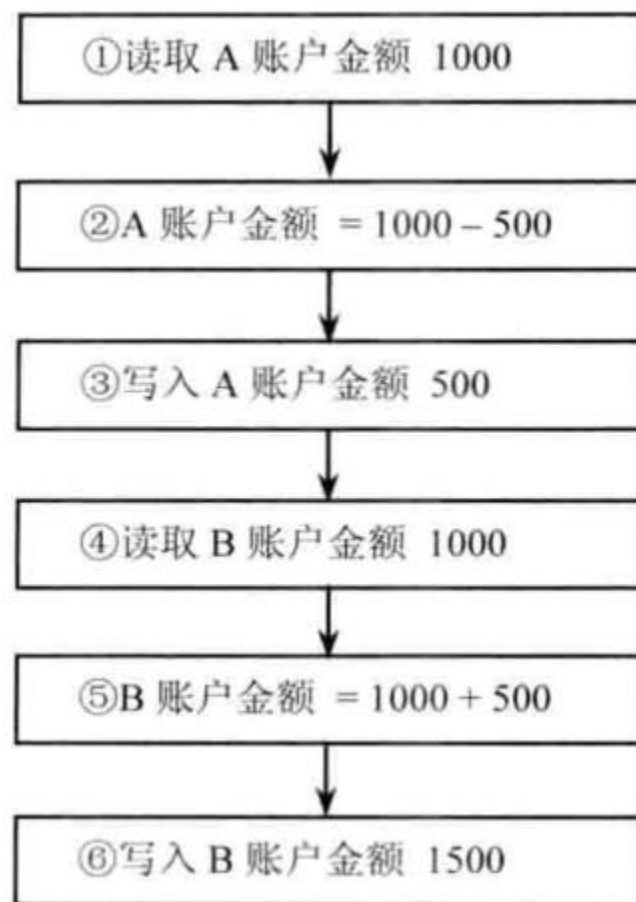


□事务的四个特性 (ACID)

1、原子性 (Atomicity)： 一个事务里面所有包含的 SQL 语句是一个执行整体，不可分割，要么都做，要么都不做。

假如在处理过程中，在完成了步骤3，尚未完成步骤6的过程中突然发生电源或软硬件故障，这样数据库中的数据就变成了A=500，B=1000，数据的一致性已被破坏，必须将数据恢复到初始状态，这称为数据库的回滚 (ROLLBACK) 操作。

转账事务处理流程
初始A=1000，B=1000
A给B转账500



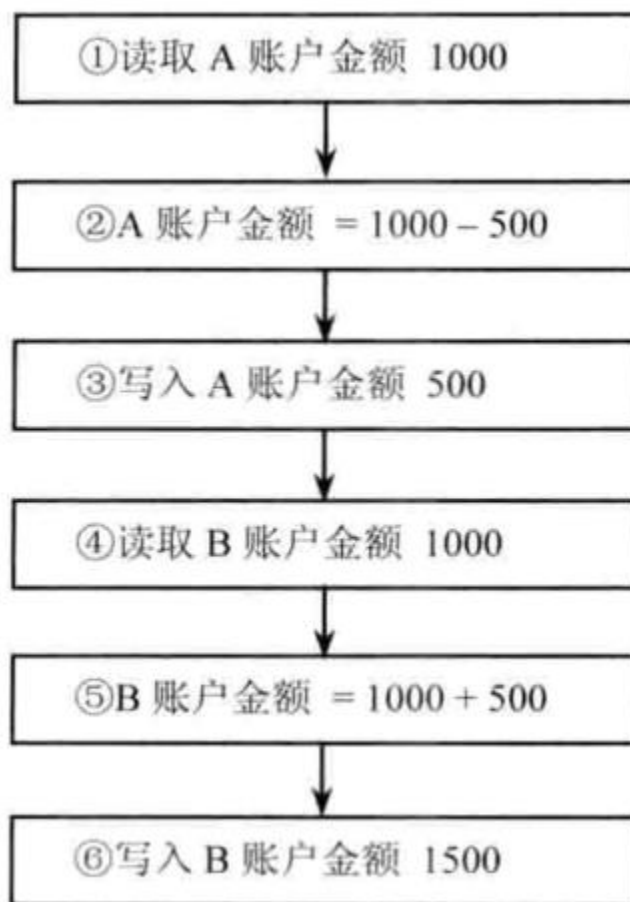


□事务的四个特性 (ACID)

2、一致性 (Consistent)：是指事务应该正确的转换系统状态。事务开始时，数据库中的数据是一致的，事务结束时，数据库的数据也必须是一致的。

银行转帐过程中，要么转帐金额从一个帐户转入另一个帐户（在不考虑转账费用的情况下，转账方减少的金额与收账方增加的金额是相等）要么两个帐户都不变，没有其他的情况。

转账事务处理流程
初始A=1000, B=1000
A给B转账500



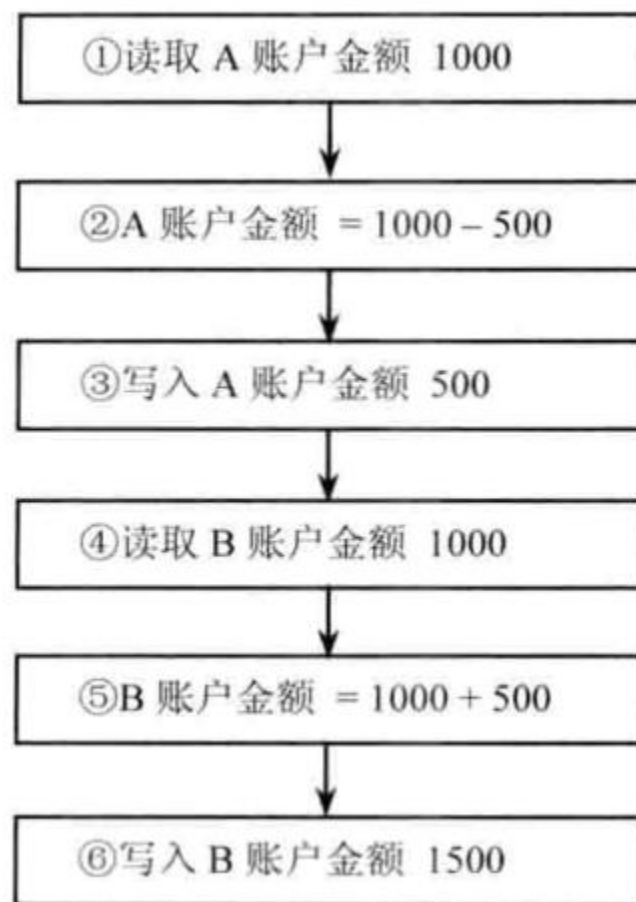


□事务的四个特性 (ACID)

3、隔离性 (Isolation)：是指数据库允许多个并发事务同时对数据进行读写和修改的能力，而不会导致数据不一致状态，即不同事务的执行不能互相干扰。

银行的每笔转账都不能被其它的转账干扰，对相同账户的不同转账操作不会同时进行。

转账事务处理流程
初始A=1000, B=1000
A给B转账500



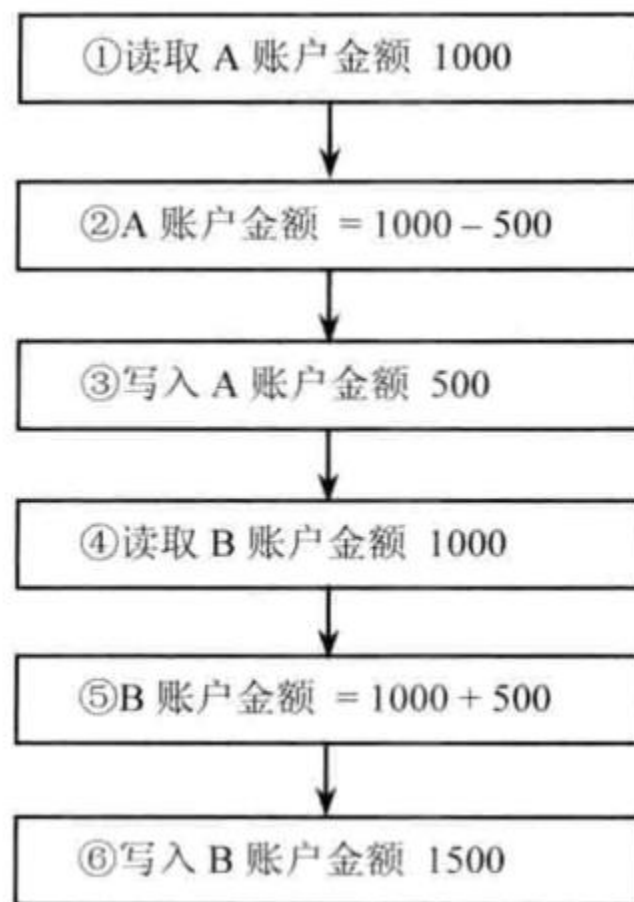


□事务的四个特性 (ACID)

4、持久性 (Durability)： 是指事务一旦被提交，它对数据库中数据的改变就是永久性的，后续的操作和故障不应该对其有任何影响。

银行的每笔转账结果都会被永久保存下来，后续的转账和故障不应该对这笔结果有所改变。

转账事务处理流程
初始A=1000, B=1000
A给B转账500





□事务处理——事务的开始与结束：

事务是用来分割数据库操作的逻辑单元，事务既有起点，也有终点。Oracle的特点是没有“开始事务处理”语句，但有“结束事务处理”语句。

Oracle提供的事务控制是隐式自动开始的，不需要用户显式的执行事务开始的命令。

● 当发生如下事件时，事务就自动开始了：

- (1) 连接到数据库，并开始执行第一条DML语句（INSERT、UPDATE或DELETE）
- (2) 前一个事务结束，又输入另一条DML语句。



□事务处理——事务的开始与结束

- 事务的结束处理，需要用户进行指定的操作，当发生如下事件时，事务就结束了：
 - (1) 用户执行COMMIT语句提交事务，或者执行ROLLBACK语句撤销了事务；
 - (2) 用户执行一条数据定义语句/数据控制语句（DDL/DCL），如成功 Oracle 会自动 COMMIT，失败会自动 ROLLBACK；
 - (3) 正常断开数据库连接、正常退出 SQL Plus 环境，Oracle 会自动 COMMIT，失败会自动 ROLLBACK；



□ 事务处理——事务管理的命令

事务管理的命令包括：

1. **COMMIT** 语句提交事务
2. **ROLLBACK** 语句回退全部事务
3. **设置保存点** 回退部分事务



□事务管理的命令——COMMIT语句

1、COMMIT语句：用于事务的提交处理，即该语句执行后，将事务中所有对数据库的更新写回到磁盘上永久保存。

(1) 事务提交前，SQL 语句执行完毕，Oracle已经完成如下操作：

- **回滚缓冲区**生成回滚记录，包含所有已经修改值的旧值。
- 在**SGA日志缓冲区**生成该事务的日志。
- 修改后的数据写入**SGA数据缓冲区**。



□事务管理的命令——COMMIT语句

1、COMMIT语句：用于事务的提交处理，即该语句执行后，将事务中所有对数据库的更新写回到磁盘上永久保存。

(2) 在使用COMMIT提交事务时，Oracle将执行如下操作：

- 在回退段的事务表内记录这个事务已经提交，并且生成一个唯一的系统变更号（SCN）保存到事务表中，用于唯一标识这个事务。
- SGA数据缓冲区内的数据写入物理数据表
- 启动LGWR后台进程，将SGA区重做日志缓存在的重做记录写入联机重做日志文件，并且将该事务的SCN也保存到联机重做日志文件中。
- 释放该事务中各个SQL语句所占用的系统资源。
- 通知用户事务已经成功提交。



□ 事务管理的命令——COMMIT语句

1、COMMIT语句：用于事务的提交处理，即该语句执行后，将事务中所有对数据库的更新写回到磁盘上永久保存。

(3) 事务的提交方式：

- **显示提交：**使用 `commit` 命令使当前事务生效
- **自动提交：**SQL Plus中执行 `set autocommit on;` 语句，只针对当前连接有效。
- **隐式提交：**
 - 正常执行完成 DDL/DCL 语句。
 - 正常退出 SQLPlus 或者 SQL Developer 等客户端。
 - 正常关闭数据库。



□事务管理的命令——ROLLBACK

2、 ROLLBACK语句：用于事务的回滚处理，撤销未提交的事务所完成的操作，数据库被恢复到事务执行之前的状态。

Oracle通过回退段（或撤销表空间）存储数据修改前的数据，通过重做日志记录对数据库所做的修改。如果回退整个事务，Oracle将执行以下操作：

- Oracle通过使用回退段中的数据撤销事务中所有SQL语句对数据库所做的修改。
- 释放事务处理所使用的资源，解锁相关记录和表。
- 通知用户事务回滚成功。



□ 事务管理的命令——SAVEPOINT

3、设置保存点 (savepoint) : 类似于调式程序的断点, 利用保存点可以将事务划分为若干部分, 这样回滚时就不必回滚整个事务, 增加灵活性。回滚到指定保存点将完成如下工作:

- 回滚保存点之后的部分事务。
- 删除在该保存点之后建立的全部保存点, 但保留该保存点, 以便多次回滚。
- 解锁保存点之后涉及的记录和表。

savepoint *sp_name*;

--设置保存点

release savepoint *sp_name*;

--删除保存点

rollback to *sp_name*;

--回滚至保存点



□事务管理的命令——SAVEPOINT

SAVEPOINT是指在事务中设置一个保存点，它可以使一个事务内的部分操作回退。

```
.....                               /* A组语句序列 */
SAVEPOINT 保存点1
.....                               /* B组语句序列 */
ROLLBACK TO 保存点1                /* 回滚到保存点1 */
COMMIT                             /* 此时只提交了A组语句 */
```

在A组语句后设置一个保存点1，然后执行B组语句后回滚到保存点1，再提交，此时B组语句已被回滚，只有A组语句的操作被提交生效。



□ 事务控制

【示例1】 HR模式中，演示使用保存点回滚事务。

```
select * from dept;  --4条记录
```

```
insert into dept values(15, 'MARKET', 'BEIJING');
```

```
savepoint sp01;
```

```
insert into dept values(25, 'HR', 'SHANGHAI');
```

```
savepoint sp02;
```

```
select * from dept;  --6条记录
```

```
delete from dept where deptno = 15;
```

```
select * from dept;  --5条记录
```

```
rollback to sp02;
```

```
select * from dept;  --6条记录
```

```
rollback to sp01;
```

```
select * from dept;  --5条记录
```

```
rollback;
```

```
select * from dept;  --4条记录
```




□ 数据并发性与一致性

在单用户数据库中，用户不必考虑其他用户修改相同的数据的情况，但是在多用户操作统一数据库的情况下，多个事务可能同时对同一数据进行更新，对这些事务不加以控制，会导致产生数据的不一致。

多用户数据库必须保证**数据的并发性和数据的一致性**。并发性是指允许多用户访问同一数据，一致性是指多用户数据库中每个用户看到的数据都是一致的，用户对数据进行修改和提交事务所产生的的变化对于其他用户也是可见的。



□数据不一致问题

为了改善系统的资源利用率并减少事务运行的平均等待时间，在数据库系统中，事务需要并发执行。事务并发执行时如果不加控制约束，可能会产生若干数据不一致问题，如：

- 脏数据库读取（Dirty read）
- 不可重复读取（Nonrepeatable read）
- 幻象读取（phantom read）



□数据不一致问题

1、脏读：事务 A 对数据进行了更新但没有提交，事务 B 读到了事务 A 更新后（未提交）的数据。如果事务 A 进行了回滚，那么刚刚事务 B 看到的数据就是脏数据。事务 B 进行了脏读。

示例：甲给乙转账500，但是甲还没有提交，但是此时乙查询自己账户多了500。然后甲发现转错人了，回滚了转账。然后乙多出的500就没了。在这个过程中乙查到了没有提交的数据（多出的500），这就是脏读。



□数据不一致问题

2、不可重复读：事务 A 读取数据库中的数据后，事务 B 更新了数据，当 A 再次读取数据时，就会发现数据已经发生了改变，这就是不可重复读取。不可重复读取所导致的结果就是一个事务前后两次读取的数据不相同。

示例：甲查询自己银行余额为1万，乙这个时候用甲的账号取走了3000，甲再一次查询余额，变成了7000。对甲而言两次结果不一致就是不可重复读。



□数据不一致问题

3、幻读：如果事务 A 基于某个条件读取数据后，另一个事务则更新了同一个表中的数据，这时第一个事务再次读取数据时，根据搜索的条件返回了不同的行，这就是幻读。

示例：甲查询自己银行明细中与乙的资金往来，之后乙向甲转了一笔账，甲再次查询发现比之前的查询多了一条新的记录，这就是幻读。


不可重复读重点在于update，而幻读的重点在于insert。



□ 数据的隔离级别(Transaction Isolation Level)

为了应对上述数据不一致问题，事务隔离级别的概念就产生了。隔离级别越高，上述问题就会越少，但是性能消耗也会越大。所以在实际生产过程中，要根据需求去确定隔离级别。

1、在 SQL 92 标准中定义了 4 个事务的隔离级别：

- Read uncommitted （读未提交）
 - Read committed （读已提交）
 - Repeatable read （可重复读）
 - Serializable （串行读）
- 



□ 数据的隔离级别

- **Read uncommitted**：能够读取没有被提交的数据。
- **Read committed**：即能够读取已经提交的数据。
- **Repeatable read**：在数据读出来之后加锁，事务不结束，别的事务就不可以改读出的数据。
- **Serializable**：最高的事务隔离级别，不管有多少事务，必须运行完一个事务的所有语句之后才可以执行另外一个事务中的语句，如果事务尝试更新由另一个事务更新并未提交的资源，则事务将失败。



□ 数据的隔离级别

事务4个隔离等级与数据异常的关系

隔离级别	脏读	不可重复读	幻读
Read uncommitted (读未提交)	是	是	是
Read committed (读已提交)	否	是	是
Repeatable read (可重复读)	否	否	是
Serializable (串行读)	否	否	否

避免不可重复读需要锁行就行，避免幻影读则需要锁表



□ 数据的隔离级别

2、Oracle 支持 SQL92 中 Read committed 和 Serializable 两种级别。

- Oracle 默认的隔离级别是 Read committed。
- Oracle 中还定义 Read only 和 Read write 隔离级别
 - Read only：事务中不能有任何修改数据库中数据的操作语句，是 Serializable 的一个子集。
 - Read write：这是默认设置，表示事务中可以有访问语句和修改语句。



□ 数据的隔离级别

设置事务隔离级别的语句：

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

SERIALIZABLE

READ ONLY

READ WRITE

四种级别是互斥的，不能同时设置两个或两个以上的级别。



□ 数据的隔离级别

【示例1】 SCOTT模式中，演示默认的Read committed级别。

会话1	会话2	说明
使用scott用户连接到PDBORCL。 sqlplus scott/password@//localhost:1521/pdborcl	sysdba登录，将容器切换为PDBORCL。 sqlplus / as sysdba alter session set container= <i>pdborcl</i> ;	同时创建两个会话： 打开两个命令行窗口， 或分别使用SQLPlus和 SQL Developer。
select empno,ename,sal from scott.emp where empno=7369;		返回结果相同。
update scott.emp set sal=1000 where empno=7369;		会话1修改7369员工的 工资为1000。



□ 数据的隔离级别

【示例1】 SCOTT模式中，演示默认的Read committed级别。

会话1	会话2	说明
	<code>select empno,ename,sal from scott.emp where empno=7369;</code>	会话2中发现修改未生效。
<code>commit;</code>		会话1进行提交。
	<code>select empno,ename,sal from scott.emp where empno=7369;</code>	会话2中发现修改已生效。



□ 事务

□ 锁



□ 基本概念

锁 (Lock) 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。锁在多用户的环境下可以保证数据库的完整性和一致性。

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。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。当用户提交或回滚一个事务后，Oracle 将释放此事务内各个 SQL 语句获得的锁。



□ 锁的级别

- **共享锁（Share Locks，S 锁，读锁）**：加了共享锁的数据对象可以被其它事务读取，但不能被修改。

若事务T获得了数据对象A的S锁，则事务T只能读A；其它事务只能获得A的S锁（读A），不能获得X锁，直到T释放A上的S锁。

- **排它锁（eXclusive Locks，X 锁，独占锁，互斥锁，写锁）**：加了排它锁的数据对象，其它事务不能对其进行加锁。

若事务T获得数据对象A的X锁，其它事务不能再获得A的任何类型的锁，获得X锁的事务既能读数据，又能修改数据。



□ 锁的类型——按是否需用户干预

- **隐式锁（自动锁）**：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，根据执行的语句和对象自动确定锁的类型和级别。
- **显式锁**：较少使用，用户使用命令明确对某一对象加锁。

LOCK TABLE 表名 IN **锁模式** [NOWAIT | WAIT <time>]

- ROW SHARE
- ROW EXCLUSIVE
- SHARE UPDATE
- SHARE
- SHARE ROW EXCLUSIVE
- EXCLUSIVE



□ 锁的类型——按操作和保护对象的不同

- **DML 锁 (DML Locks, 数据锁)**：用于控制并发事务中的数据操纵，保证数据的一致性和完整性。
例如，DML 锁保证表的特定行能够被一个事务更新，同时保证在事务提交之前，不能删除表。
- **DDL 锁 (Dictionary Locks, 字典锁)**：用于保护数据库对象的结构，如表、索引等的结构定义。
- **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。包括闕锁、互斥体、和内部锁。



□ DML 锁

在Oracle数据库中，**DML锁主要包括TM锁和TX锁**，其中TM锁称为表级锁，TX锁称为事务锁或行级锁。

- **行锁 (Row Locks, TX)**：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。
- **表锁 (Table Locks, TM)**：对 DML 执行时并发的 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。TM锁包括了SS、SX、S、X等多种模式，在数据库中使用0~6来表示。



□ DML 锁

当 Oracle 执行 DML 语句时，**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了，而只需检查TM锁模式的相容性即可，从而提高了系统的效率。

锁 □ DML 锁



模式	所模式别名	锁类型	对应SQL操作
0、1	None(没有)、NULL(空)		select
2	行共享表锁 Row Share (RS)	TM	lock in row share
3	行排他表锁 Row Exclusive(RX)	TM	insert、update、delete、 select for update、lock in row exclusive mode
4	共享表锁 Share(S)	TM	lock in share mode
5	共享行排他表锁 Share Row Exclusive (SRX)	TM	lock in share row exclusive mode
6	排他锁 Exclusive(X)	TM/TX	lock in exclusive mode

锁 □ DML 锁



模式	所模式别名	锁类型	对应SQL操作
0、1	None(没有)、NULL(空)		select
2	行共享表锁 Row Share (RS)	TM	lock in row share
3	行排他表锁 Row Exclusive(RX)	TM	lock in row exclusive mode
4	共享表锁 Share(S)	TM	lock in share mode
5	共享行排他表锁 Share Row Exclusive (SRX)	TM	lock in share row exclusive mode
6	排他锁 Exclusive(X)	TM/TX	lock in exclusive mode

行共享表锁(RS): 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。某个事务拥有了表的RS锁后只会禁止其他事务对相同表获取X锁

锁 □ DML 锁



模式	所模式别名	锁	
0、1	None(没有)、NULL(空)		
2	行共享表锁 Row Share (RS)		
3	行排他表锁 Row Exclusive(RX)	TM	insert、update、delete、select for update、lock in row exclusive mode
4	共享表锁 Share(S)	TM	lock in share mode
5	共享行排他表锁 Share Row Exclusive (SRX)	TM	lock in share row exclusive mode
6	排他锁 Exclusive(X)	TM/TX	lock in exclusive mode

通常表示持有锁的事务已更新了表行。
一个RX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，RX锁允许多个事务对同一个表同时获得RX和RS表锁。某个事务拥有了表的RX锁后只会禁止其他事务对相同表加S、SRX和X锁

锁 □ DML 锁



模式	所模式别名	锁类型	对应SQL操作
0、1	None(没有)、NULL(空)		select
2	行共享表锁 Row Share (RS)	TM	lock in row share
3	行排他表锁 Row Exclusive(RX)	TM	insert、update、delete、 lock in row exclusive mode
4	共享表锁 Share(S)	TM	lock in share mode
5	共享行排他表锁 Share Row Exclusive (SRX)	TM	lock in share row exclusive mode
6	排他锁 Exclusive(X)	TM/TX	lock in exclusive mode

阻止其他DML事务操作

锁 □ DML 锁



模式	所模式别名	锁类型	对应SQL操作
0、1	None(没有)、NULL(空)		select
2	行共享表锁 Row Share (RS)	TM	lock in row share
3	行排他表锁 Row Exclusive(RX)	<p>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SRX锁。某个事务拥有的SRX锁允许其它事务查询该表（除SELECT...FOR UPDATE）但不能更新该表。</p>	delete、lock in row
4	共享表锁 Share(S)		de
5	共享行排他表锁 Share Row Exclusive (SRX)		lock in share row exclusive mode
6	排他锁 Exclusive(X)	TM/TX	lock in exclusive mode

锁 □ DML 锁



模式	所模式别名	锁类型	对应SQL操作
0、1	None(没有)、NULL(空)		select
2	行共享表锁 Row Share (RS)	TM	lock in row share
3	行排他表锁 Row Exclusive(RX)	TM	insert、update、delete、 select for update、lock in row exclusive mode
4	共享表锁 Share(S)	TM	lock in share mode
5	共享行排他表锁 Share Row Exclusive (SRX)	TM/TX	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。 Oracle只有表级别的共享锁没有行级别的共享锁，行级别的是排他锁
6	排他锁 Exclusive(X)		



□DML 锁

不同 TM 锁的兼容模式

模式	Get Held	NULL	RS	RX	S	SRX	X
0、1	NULL	✓	✓	✓	✓	✓	✓
2	RS	✓	✓	✓	✓	✓	
3	RX	✓	✓	✓			
4	S	✓	✓		✓		
5	SRX	✓	✓				
6	X						



□ DML 锁

不同 SQL 语句的锁模式

SQL语句	TX 模式	TM 模式	允许是否其它事务获取TM				
			RS	RX	S	SRX	X
select...from table...		NULL	Y	Y	Y	Y	Y
select...from table for update...	X	RS/ RX	Y*	Y*	Y*	Y*	N
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

Y*表示当不与其它事务的TX锁冲突时才允许，否则将被阻塞。



□DML 锁

不同 SQL 语句的锁模式

SQL语句	TX 模式	TM 模式	允许是否其它事务获取TM				
			RS	RX	S	SRX	X
lock table in row share mode		RS	Y	Y	Y	Y	N
lock table in share update mode		RS	Y	Y	Y	Y	N
lock table in row exclusive mode		RX	Y	Y	N	N	N
lock table in share mode		S	Y	N	Y	N	N
lock table in share row exclusive mode		SRX	Y	N	N	N	N
lock table in exclusive mode		X	N	N	N	N	N



□ DDL 锁

DDL 锁的作用是在执行 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。例如，用户创建一个存储过程时，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删除。

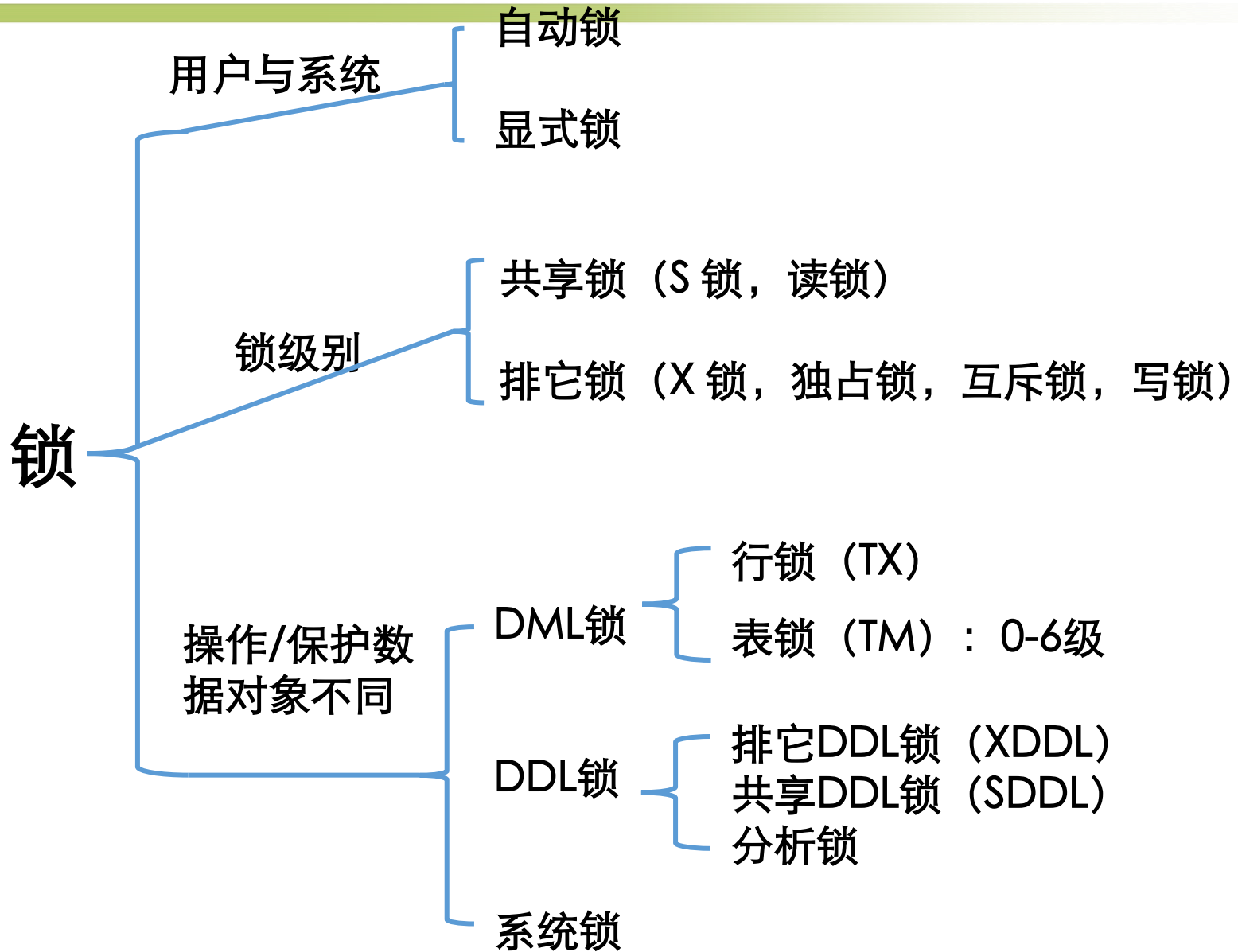
- DDL 锁只锁定 DDL 操作所涉及的对象，而不会锁定数据字典中的所有对象。
- DDL 锁由 Oracle 自动加锁和释放，不能显式地给对象加 DDL 锁，即没有加 DDL 锁的语句。



□ DDL 锁

- **排它 DDL 锁 (XDDL)** : 大多数 DDL 语句都带有一个排它 DDL 锁, 如 Alter Table。XDDL 锁可防止其它会话获取 DDL 或 DML 锁。
- **共享 DDL 锁 (SDDL)** : 如当 CREATE PROCEDURE 语句运行时, 所在事务将为所有被引用的表获取 SDDL 锁
- **分析锁 (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分析锁。获取分析锁的目的是, 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。

锁总结





1. 自行完成课件中事务的两个演示示例。
2. 自学v\$log
3. 关于锁的学习参考：
<http://blog.itpub.net/29108856/viewspace-2147524/>