所有信息系统的核心就是共享数据。随着系统的功能变得越来越高级，我们有时候会忘记有效的共享数据才是控制整体系统性能的底层主导因素。与此同时，数据库系统必须保护数据的完整性，因为这些数据的值与数据的正确性直接相关。数据库系统在为多用户访问提供更高性能的时候也必须保护数据的完整性。这两种因素有时候会相互冲突，它们构成了数据库系统的一些核心技术。

数据完整性必须放在首要位置。一个多用户数据库在某种意义上必须能够以某种方式处理并发执行的事务，从而“确保可预测的，可重现的结果”。这个目标就是数据库完整性的核心问题，同时也是所有数据库系统的基础。

当多用户访问同一个数据时，总是存在这样的可能性，即某个用户对数据某个部分的改变会被另一个用户的改变无意覆盖。如果这种情况发生，数据库信息的准确性就会受到影响，提交的数据可能会没用，甚至更糟，也就是会产生误解。同时，如果采用防止这种数据丢失的技术，也就是要求一个用户等待其他用户完成任务后才能继续他自己的工作，这样又会大大降低应用程序的性能。并不能通过增加资源的方法来解决这种性能问题。

尽管并发问题对应用的成功非常重要，但是同时也是最难以预知的问题，因为它们都是从负责的交互环境中滋生出来的。若并发用户数量增加，就会相应增加并发访问的难度。因为并发问题是由大量用户同时访问导致的，而这些用户可能无法使用测试环境，所有即使是一个非常健壮的调试和测试环境，在检测这类问题时可能也会失败。在一个应用程序运行的过程中，如果用户访问的模式发生改变，并发性问题也可能发生。

如果数据库软件不能对并发访问引起的问题进行适当处理，那么开发人员可能会发现它们需要面对许多问题。这将会浪费大量的开发时间。在开发的后期，他们经常需要增加代码并且围绕数据库可能存在的不足进行测试，从而可能改变应用程序的设计。更糟糕的是，他们不得不改变最优的数据结构以弥补底层数据库能力的不足。

只有一种方法能够解决并发问题，那就是提供服务的数据库必须改进自身策略从而透明地解决并发访问导致的问题。幸运的是，Oracle拥有处理并发访问的绝好方法。本章描述了并发访问的基础知识，并概述了Oracle处理并发访问问题的方法。

## 1、并发访问的基础知识

与并发相关的基本概念

### 1.1事务

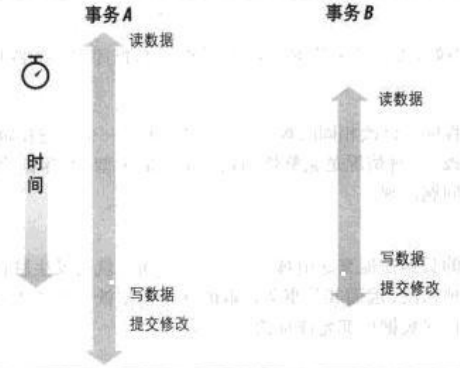
在多用户数据库环境中，事务是数据完整性的基础，也是所有并发模式的基础。事务是指用于某些数据的一个不可分割的操作。在一个事务中，所有对数据的修改都通过COMMINT语句统一提交到数据库中，或者将改变的数据通过ROLLBACK语句统一返回到初始状态。一旦事务被提交，那么该事物引起的改变就永久地存在，而且对其他事务和其他用户都是可见的。

由于事务引起的改变只有在事务被提交之后才能证实被数据库接收了，因此每个独立的事务必须不受其他事务的影响。实施事务独立性的机制就是锁。

### 1.2锁

数据库使用锁系统的方法来防止事务之间相互干扰。上锁可以防止其他用户修改数据。数据库系统使用锁来保证一个事务做的修改不被其他事务覆盖。

如图显示了如果系统不使用锁可能会出现的潜在问题。事务A读一段数据：事务B也读这段数据并且提交了对这段数据的修改，当事务A提交数据的时候，它的修改会不经意地覆盖事务B所做的修改，这样就导致了数据完整性的丢失。



有两种类型的锁可用于避免这类问题。

写锁（write lock）或互斥锁（exclusive lock）

在一个事务过程中，当数据发生修改时，就加上一个互斥锁，当事务通过COMMIT或ROLLBACK语句终止时，互斥锁就被释放。在某个时间，一个写锁只能被唯一一个用户所占有，所以只有一个用户能够改变数据。

读锁（read lock），又称共享锁（shared lock）

由于数据的某个部分是多用户共享的，因而一个读锁运行多个只读数据的用户同时拥有。但是由于写锁是互斥的，所以读锁不允许同时在该数据上加写锁。上图中，如果当事务A开始时在数据上加一把读锁，那么事务B还是可以读同样的数据，不过在事务A终止之前事务B不能请求写锁。

当SQL操作通过在SELECT语句中的FOR UPDATE子句中明确请求读锁时，Oracle才会使用读锁。通常情况下，你不应该使用FOR UPDATE子句，因为它会过度地增加读数据用户干扰写数据用户的几率。在Oracle中这种情况是不应该出现的，很快你就可以看到这一点。

### 1.3并发与竞争

利用锁来隔离并发用户各自的数据，这种方法具有其自身的问题。从上面的例子中可以看到单个事务在数据库上加锁会妨碍其他事务的完成，从而引发严重的性能问题。由于锁冲突而带来的干扰又叫做竞争。数据库中竞争过多会导致响应时间变长，系统整体吞吐量下降。Oracle的多版本读操作一致性技术能够在很大程度上减少竞争。

### 1.4完整性问题

如果事务隔离做得不好，还会带来基本的完整性问题。数据库常见的四种完整性问题有：

更新丢失：

当两个写进程同时修改相同的数据时，往往会出现一个写进程做的修改覆盖另一个写进程的修改。这种情况是完整性问题最常见的类型。互斥锁的目的就是为了解决这种类型的问题。

脏读：

一个事务修改的数据在提交之前被另一个事物读取，就会发生脏读。由于事务的提交修改可能被回滚，因而其他事务读取的这个数据就可能不正确。为了避免读锁导致的竞争，许多数据库都允许脏读。

不可重复读：

由于其他事务的修改，可能出现不可重复读。例如，一个事务根据某个条件做查询。当这个事务得到返回结果集之后，但是这个事务完成之前，另一个事物修改了数据，使得某些数据不再满足前一个事务的查询条件。如果这个时候前一个事务重复这个查询操作，那么它将得到一个不同的结果集，因而针对最初的结果集所做的修改可能就无效了。简言之，根据同样的条件在同一事务中读两次数据可能会产生不同的结果。

幻影读：

这类型的问题也是由于其他事务对数据的修改而导致的。例如，一个事务根据某个条件做查询。在前一个事务得到返回结果集之后，但是在它完成之前，另一个事物在数据库中插入了一个新行，并且该行数据符合前一个事务的选择条件。如果事务中的第一条SQL语句返回了最初满足选择条件的行，之后该事物对满足条件的这些行执行操作，这个操作所涉及到的行与之前查询到的行不同，这就因为包含了幻影行而导致的。

### 1.5串行化

一个完整的并发解决方案的目标就是在访问同一数据的不同用户操作之间提供最高级别的独立性。在SQL92标准中这个最高的级别叫做可串行化。顾名思义，可串行化的事务就是一组有序的事务。当一个事务开始执行时，它与其后事务中产生的数据改变是相互独立的。

对用户来说，一个可串行化的事务在执行期间就好像是独占数据库的。可串行化的事务是可预测的和可再生的，这正是数据完整性的两个重要特点。

## 2、Oracle与并发用户访问

Oracle采用了多版本读操作一致性技术来解决并发访问的问题，这项技术又叫做MVRC。它能够保证每个用户都可以看到所要请求数据的完整视图。如果其他用户在查询过程中修改了底层数据，Oracle会在查询期间另外维护一个数据版本。如果在这个查询操作过程中有事务是修改数据但尚未提交的，Oracle将会确保该查询能够忽略当时事务所做的修改。查询的返回结果集将反映查询开始时所有提交的事务。

这个特性对数据库的查询有两个显著的影响。

首先，对于读操作，Oracle对数据不加任何锁。这就意味着读操作永远不会干扰写操作。即使一个数据库在读操作过程中对某一行加了一把锁，这仍然会导致数据库的竞争，特别是因为大部分数据库表都集中于对某些热点数据进行更新。

其次，用户得到一个数据的完整“快照”视图，它反映出在查询开始时的数据状态。其他数据库可能通过只在读数据的时候锁定记录，而不是整个事务的执行期间锁定行的方式来减少竞争的总量，在这种方式下，结果记录集的某行可能和获取结果集开始的时候不一样。因为在查询过程中要读的行没有加锁，可能会被其他用户修改，从而导致产生不一致的数据视图。

## 3、Oracle的隔离级别

Oracle使用隔离级别来描述一个事务是如何和其他事务交互的，以及如何与其他事务相隔离。隔离级别在本质上是一种数据库加锁的方案，可用来确保某个特定类型的事务独立。

应用程序员在会话或事务级别可以使用SQL语句设置一个隔离级别，例如，使用ALTER SESSION或SET TRANSACTION来设置。对于一个更为严格的隔离级别而言，开发人员应该在其可能带来的潜在冲突与增加数据完整性保护所带来的利益之间进行权衡。

在Oracle中的两个经常使用的基本隔离级别：READ COMMITTED和SERIALIZABLE（第三种级别READ ONLY将在后面介绍）。这两个隔离级别都可以用于创建可串行化的数据库操作。两个级别不同之处在于实施串行化操作的时间段不同：

READ COMMITTED

在语句级别实施串行化。这意味着每个语句将得到一个数据的完整视图，该视图存在于语句的开头。但是，由于一个事务可以包含多个语句，这样在事务上下文中可能会发生不可重复读和幻影读。READ COMMITTED的独立级别是Oracle默认的。

SERIALIZABLE

在事务级别实施串行化。这意味着事务中每个语句将得到事务开始时的同一个完整视图。

因为两者控制时间段不同，因此，如果有一个事务通过在某条请求记录上用加锁的方式来阻塞它们的操作，这两个隔离界别反应也是不同的。一旦阻塞事务释放了锁，READ COMMITTED隔离级别的其他正在执行的操作将会重新执行。因为这个操作只关心语句开始的数据状态，这是一个很完美的逻辑方法。

另一方面，如果阻塞事务提交了对数据的修改，SERIALIZABLE独立级别上其他正在执行的操作将会返回一个错误，以表示它不能进行串行化操作。这个错误表明，因为阻塞事务已经改变了SERIALIZABLE事务开始时的数据状态，这样使得在被修改的行中不能进行任何写操作。在这种情况下，应用程序员必须在程序中增加逻辑语句，以便返回SERIALIZABLE事务的开始状态，然后重新执行。

Oracle还支持另外一种隔离级别READ ONLY，在一个会话或事务中还可以声明为这种隔离级别。显而易见，这个级别明确拒绝任何写操作，从而在一个事务开始时提供所有数据的准确视图。

## 4、Oracle的并发特性

Oracle使用是哪个特性来实现多版本读操作的一致性：

回滚段：

系统改变号（SCN）：

数据块中的锁：

以上几种特性都是与多版本读操作的一致性直接相关的。Oracle还为用户众多的环境提供了一种更高级别的并发特性：

非升级行锁：

Oracle并发服务器上曾经使用过一个分布式锁管理器（DLM）的锁管理器，它可用于跟踪多个Oracle实例的锁。这是一种完全不同的独立加锁的模式，并不会影响Oracle处理行锁。Oracle并发服务器使用的这种技术在Oracle 9i中得到了改进，并且被集成进入Oracle 9i的核心组件实时应用程序集群。

## 5、Oracle如何处理锁

在了解并发的概念以及明确了有关Oracle如何处理多用户并发访问的特性。为了更好理解这些特性是如何作用的，我们将用三个例子来讲述：

数据库一次简单的写操作，两个用户试图对同一个表中的同一条记录进行写操作的情况，以及在冲突的更新操作中发生的读操作。

### 5.1一次简单的写操作

本例描述了一次简单的写操作，即一个用户要写数据库中的一行。在这个例子中，一个HR职员想要更新一个员工的名字。假定这个HR职员在屏幕上已经有了该员工的记录。下面就是从这点开始的步骤：

1. 客户在屏幕上修改雇员姓名。客户进程通过网络将一个SQL UPDATE语句发送到服务器进程。
2. 服务器进程获得系统改变号，读取包括目标行的数据块。
3. 服务器记录数据块中的行锁信息。
4. 服务器将数据中旧的镜像写入内存中的redo缓存，然后将修改写入回滚段，并修改员工数据，在Oracle 10g及以后，还要将SCN写入ORA\_ROWSCN伪列。
5. 服务器进程将redo日志写入磁盘，然后将回滚段和被修改的数据写入磁盘。回滚段的修改时redo日志的一部分，因为redo日志存储了事务产生的所有修改。
6. HR职员提交事务。
7. 日志写进程（LGWR）将整个事务的日志信息（包括标志事务提交时间的SCN）从redo日志缓存写入磁盘中的当前redo日志文件。当操作系统确认对redo日志的写操作已经成功完成，这个事务才被认为已提交。
8. 服务器进程给客户发送正是提交的信息。

从Oracle 10g开始，服务器进程能够直接将控制返回给用户而不需要所有的redo信息都被写入硬盘。该特性的一个正面作用就是，它使得高覆盖的OLTP应用程序性能得到提高。负面作用是给系统留下了一个脆弱的窗口——如果在事务提交之后，写入redo日志之前，数据库宕机，那么就无法恢复之前已提交的事务，因而使用该特性时需谨慎。

### 5.2冲突的写操作

在这个例子中，有两个用户，客户A和客户B，它们正试图同时修改同一行数据。步骤如下：

1. 客户A在屏幕上修改员工姓名。客户A通过网络将SQL UPDATE语句发送到服务器进程。
2. 服务器进程获得该语句的SCN，并读取目标行所在的数据块。
3. 服务器在数据块中记录行锁信息。
4. 服务器进程将修改写入redo日志缓存。
5. 服务器进程将即将被修改的员工数据写入回滚段。一旦服务器进程完成操作，就开始修改员工数据，在Oracle 10g及之后，还要将SCN写入ORA\_ROWSCN伪列。
6. 客户B在屏幕上修改该员工姓名，并通过网络将SQL UPDATE语句发送到服务器进程。
7. 服务器进程获得该语句的SCN，并读取目标行所在的数据块。
8. 服务器进程根据数据块的头信息中发现目标行被上锁，它将采取两种动作之一。如果客户B事务的隔离级别为READ COMMITTED，那么服务器进程等待阻塞事务的完成。如果客户B事务的隔离级别为SERIALIZABLE，那么将客户返回一个错误。
9. 客户A提交事务，服务器进程据此采取适当的操作，然后服务器向客户A发送证实提交的信息。
10. 如果客户B使用READ COMMITTED隔离级别执行SQL语句，那么这条SQL语句将继续执行。

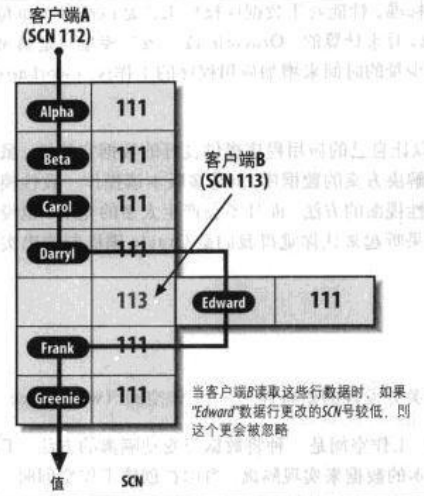
上述例子描述的是当Oracle检测到由潜在的更新丢失而导致的问题时，Oracle采用默认的操作。因为SERIALIZABLE隔离级别比READ COMMITTED级别有着更加明显的反应，所以许多开发人员宁愿采用后者。总而言之，通过在提交一个更新操作之前检查改变（比较行中的数据，Oracle 10g及之后使用SCN），或者通过在SQL中使用SELECT FOR UPDATE语法来避免问题的发生，就可以避免某些可能的冲突。

### 5.3读操作

在下面这个最常见的例子中，我们可以欣赏到Oracle读操作一致性模型的巧妙之处。在这个例子中，一位用户在读数据，另一位用户在向同一行写数据。假设客户A正在从EMP表中读取若干行数据，而客户B在客户A读之前且在客户A开始事务之后写数据：

1. 客户A通过网络将SQL SELECT语句发送到服务器进程。
2. 服务器进程获得该语句的SCN，并开始读查询请求的数据。对于读到的每个数据块，它都会拿该SELECT语句的SCN与涉及到数据块中行数据的所有事务的SCN做比较。如果服务器发现存在比当前SELECT语句更新的事务，它就会用回滚段中的数据为数据块创建一个“读操作一致性”的版本，该版本的时间与SELECT语句发起的时间一致。这个就是多版本读操作一致性（MVRC）的工作原理，它使得Oracle不需要在数据上使用读锁。如果事务启动之后行数据被更新，Oracle只需简单地获取早些事件的一致性视图。
3. 客户B发送SQL UODATE语句来更新EMP表中客户A的SELECT语句尚未读取的行。服务器进程获得该语句的SCN，并开始操作。
4. 客户B提交自己的更新。服务器进程完成整个操作，包括记录包含被修改行的数据块信息，以便Oracle能够确定这个更新事务的SCN。
5. 客户A读操作的服务器进程达到最近更新过的块。它发现，包含事务所做更新的数据块的SCN比SELECT语句的SCN要新。服务器进程在数据块头中查找一个指向回滚段的指针；该回滚段保存着客户A事务启动时的数据。回滚段使用数据的旧版本为该数据块创建一个版本，记录SELECT语句启动时刻的数据。客户A的SELECT语句从数据块的一致性版本中读取需要的行。

如图显示了多版本读操作一致性的读过程：



Oracle的并发处理能力在高负荷的情况下可以显著地避免竞争的产生和性能的衰退——事实上，工作负荷越大，MVRC的优势越明显。

## 6、并发访问与性能

在Oracle实现多版本读操作一致性的过程中，它通过最小化和延迟不必要的I/O操作来保证最佳的性能。为了确保数据库中数据的完整性，当系统发生故障的时候数据库必须能够恢复，这就意味在数据库发生故障的时候，必须有一种方法来确保数据库中的数据能够准确反映提交数据的状态。Oracle通过在事务提交的时候向数据库中写入被修改的数据来做到这一点。但是，redo日志包括的信息比数据块中的信息要少得多，因而将它写入磁盘是一种“便宜”的做法。每当一个事务被提交，Oracle就将日志信息写入磁盘，且暂时不将被修改的数据写入磁盘，直到有几个被修改的数据块能够一起写入磁盘时才执行数据写磁盘操作。Oracle可以通过redo日志来恢复数据库，这些过程减少了I/O操作耗费的时间。

然而，在考虑数据库性能的时候，不能仅仅考虑简单的I/O操作。如果你的事务正在等待另一个事务释放锁，此时不论你的数据库运行得如何快也没有用。一个速度快一点的数据库可能会更快地完成阻塞事务，但是你的事务仍然会处于停止状态，一直到阻塞事务的完成。

由于大部分数据库均执行读和写的混合操作，并且由于Oracle是市场上仅有的不用读锁的数据库，因而Oracle本质上总是尽量减少数据竞争的数量。对于一个混合型的应用程序来说，竞争越少就意味着速度越快。

对性能的度量有不同的标注。性能对于数据库操作来说都是以毫秒为单位测量的，而对于应用程序开发人员来说真是以月来计算的。Oracle的读一致性模型只是带来很少的竞争，开发人员还是不得不花费少量的实际来增加应用程序的工作区（workaround），已处理竞争结果。

有了并发解决方案可以让自己的应用程序提供良好的数据完整性。虽然Oracle不是唯一一种可以提供一致性解决方案的数据库，但是多版本读操作一致性模型确实提供了一种简单地获得数据一致性视图的方法，而且不会产生太多的竞争，也没必要在应用程序中写相应的工作区。如果听起来让你觉得我们是Oracle锁机制的忠实支持者。

### 6.1工作区间

Oracle 9i引入了一个有关并发性的新特性——工作空间（workspace）管理器。

在一般的数据库环境中，工作空间是一种将数据与变动隔离的方法。工作空间管理通过为工作空间创建特定版本的数据来实现隔离。当你在创建工作空间时，实质上就是为指定时间点的数据在工作空间中创建快照。之后工作空间外对数据的改动并不会影响工作空间中数据的视图，在工作空间内对数据所做的修改，工作空间之外的用户是无法看到的。工作空间内的数据修改只对工作空间的其他用户可见。

工作空间允许你为特定的应用创建本质上独立的数据环境。可以捕捉某个时间点的数据用于历史分析，也可以执行各种类型的“what-if”分析，来观察变动是如何影响数据的各个部分而又不干扰主产品数据库的。这两种功能在普通情况下都要求创建数据库副本，而工作空间可以为你节省时间和资源。

### 6.2工作空间的实现

实现工作空间的关键就是要支持同一个数据的多个版本。要让工作空间支持表中数据的多个版本，必须首先要这个表支持多版本。工作空间管理器可以让数据库中的一个或多个用户表支持多版本。版本化的单位是一行。当一个表支持多版本，那么该表中的所有行都能够支持多个版本的数据。版本化的行和原始行一样都存储在同一个表中。版本化的基础架构对数据库用户来说是不可见的，对支持多版本的表进行选择、插入、修改以及删除操作与之前没有两样。工作空间管理器为了支持多版本采用了下面的步骤：重命名表、在表中增加若干列来存储版本化的元数据，在支持多版本的表上创建视图，并使用原始的表名对之命名，在该视图上为SQL DML操作定义INSTEAD OF触发器。

工作空间保存对数据的修改，从而将工作空间的数据最小化，同时避免数据重复。

可以创建一个工作空间的层次结构，一个工作空间可以有多个父工作空间。所有工作空间的操作将会影响当前工作空间及其父工作空间。后面的几节将会讲述工作空间的操作。使用多级工作空间的活动。这个选项可以和保存点一起工作，记录对保存点创建的每个行版本的修改历史。这个时间戳允许工作空间的用户返回到任意时间点，观察数据库从某次变动到下一个时间点之前的变动。可以把这个特性看做是用于有限表集合的一种类型的Flashback。

另外，可以指定数据空间中某个版本的数据只能在一个指定的时间段内有效。例如，可以对工作空间中的数据做修改，并让它在随后的24小时内对工作空间的其他用户时可见的，而之后又是不可见的。

工作空间有自己的锁机制，不过它们仅适用于该工作空间的其他用户访问。可以互斥的锁住表空间的一行数据，但是这个锁只能放置这行数据被工作空间的其他用户访问。其底层的数据仍然可以被此工作空间之外的用户访问或修改。由于工作空间和这样的锁都是用于隔离数据的修改，因而这样的锁机制是可行的。工作空间存在于标准的数据库之外，因而工作空间的锁与标准数据库的锁并不会相互作用。

### 6.3工作空间的操作

有三种类型的基本操作可应用于工作空间：

回滚：

可以回滚对工作空间所做的修改，并将工作空间的状态回滚到创建时刻。你也可以指定存储点，以便能够将工作空间的状态回滚到随后的时间点。

刷新：

刷新工作空间意味着将工作空间内的数据与整个数据库中的对应数据保持一致。如果你决定在每天结束时创建一个具有数据快照的工作空间，那么可以使用这项操作。午夜时，可以刷新工作空间，从而让工作空间反映前一天的数据。

合并：

合并操作将负责将工作空间中的变动滚动到其父工作空间中。

你可以想到，刷新操作和合并操作一起作用可能会导致工作空间中的数据与其父工作空间的数据库冲突。工作空间管理能够跟踪每个表的冲突；可以据此手动解决这些冲突。

### 6.4工作空间的加强

工作空间管理器与Oracle数据库紧密地集成在一起。Oracle 10g对工作空间管理器做了加强，增加了导出导入支持多版本表的功能，利用SQL Loader将大量数据导入多版本表的功能、根据规则空间操作触发事件的功能以及定义持续被刷新的工作空间的功能。

Oracle 11g继续加强了工作空间的功能，开始支持优化器提示，并为支持工作空间的表提供更多的数据维护操作。