SQL Server和Oracle防止数据锁定的比较 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/224/2021_2022_SQL_Server_c102_224307.htm 数据库并行访问，也就是两个或两以上用户同时访问同一数据，这也是数据库引擎如何设计和实现适度反应所面临的最大问题。设计优良、性能卓越的数据库引擎可以轻松地同时为成千上万的用户服务。而“ 底气不足 ”的数据库系统随着更多的用户同时访问系统将大大降低其性能。最糟糕的情况下甚至可能导致系统的崩溃。 当然，并行访问是任何数据库解决方案都最为重视的问题了，为了解决并行访问方面的问题各类数据库系统提出了各种各样的方案。SQL Server和Oracle两大DBMS也分别采用了不同的并行处理方法。它们之间的实质差别在哪里呢？ 并行访问的问题 并行访问出现问题存在若干种情况。在最简单的情形下，数量超过一个的用户可能同时查询同一数据。就这种情况而言数据库的操作目标很简单：尽可能地为用户们提供快速的数据访问。 这对我们现在常见的数据库来说不成问题：SQL Server和Oracle 都采用了多线程机制，它们当然能够一次处理多个请求。 不过，在用户修改数据的情况下并行访问问题就变得复杂起来了。显然，数据库通常只允许唯一用户一次修改特定的数据。当某一用户开始修改某块数据时， SQL Server和Oracle 都能很快地锁定数据，阻止其他用户对这块数据进行更新，直到修改该数据的第1位用户完成其操作并提交交易（commit transaction）。但是，当某一位用户正在修改某块数据时假设另一位用户又正想查询该数据的信息时会发生什么情况呢？在这种情况下数据库管理系统又该如何动作呢

？Oracle 和 SQL Server针对这一问题采取了不同的解决方案。SQL Server方法 现在不妨假设有人开始修改SQL Server上存储的数据，于是这块数据立即被数据库锁定。数据锁定操作阻塞其他任何访问该数据的连接连查询操作都不会放过。于是，这块被锁定的数据只有在交易被提交或者回滚之后才能接受其他访问操作。 下面用SQL Server随带的pubs示例数据库做一个简单示范。在Query Analyzer内打开两个窗口。在第1个窗口中执行下列SQL操作语句，更新pubs数据库中某一图书的价格： use pubs go begin tran 0update titles set price = price * 1.05 where title_id = BU2075 由于代码中并没有执行commit语句，所以数据变动操作实际上还没有最终完成。接下来，在另一个窗口里执行下列语句查询titles数据表： 0select title_id,title,price from titles order by title_id. 你什么结果也得不到。窗口底部的小地球图标会转个不停。尽管我在先前的操作中仅仅更新了一行，但是，0select语句的执行对象却恰好包含了其数据正被修改的一行。因此，上面的操作不会返回任何数据，除非回到第1个窗口提交交易或者回滚。 SQL Server的数据锁定方案可能会降低系统的性能和效率。数据被锁定的时间越长，或者锁定的数据量越大，其他数据访问用户就越可能不得不等待其查询语句的执行。因此，从程序员的角度来看，对SQL Server编程的时候应该尽量地把交易代码设计得既小又快。 在SQL Server的最近版本中，微软对SQL Server进行了某些修改，使其一次锁定的数据量大大减少，这是数据库设计中的一大重要改进。在6.5版及以前版本中，最少的数据锁定量是一页。哪怕你只在修改一行数据，而该行数据位于包含10行数据的一页上，则整页10行数据都会被锁定。显

然，这么大的数据锁定量增加了其他数据访问连接不得不等待数据修正完成的概率。在SQL Server 7中，微软引入了行锁定技术，这样，目前的SQL Server只锁定实际正被改变的数据行。 SQL Server的解决方案听起来很简单，但实际上其幕后为提供足够的系统高性能而采取了很多措施。例如，如果你在同时修改多行数据，SQL Server则会把数据锁定范围提升到页级别乃至锁定整个数据表，从而不必针对每一记录跟踪和维护各自的数据锁。 Oracle方法 下面我们再看看Oracle数据库是如何实施类似操作的。首先，我打开一个SQLPlus实例执行下列查询语句（这个例子可以在Oracle 9i中示例中找到）。这个实例称做查询实例： 0select first_name, last_name, salary from hr.employees where department_id = 20. 代码返回两行数据，如下所示： 然后，再打开另一个SQLPlus实例更新实例来执行以下命令： SQL> 0update hr.employees 2 set salary = salary * 1.05 3 where 4 department_id = 20 5 / 代码执行后回复消息称两行数据已被更新。 注意，以上代码中并每有像在SQL Server示例那样键入“ begin tran ”字样的代码。Oracle 的SQLPlus隐含启用交易（你还可以模仿SQL Server的行为，设置“ autocommit to on ”自动地提交交易）。接下来我们在SQLPlus更新实例中再执行同查询实例一样的0select语句。 结果清楚地表明：Michael和Pat的薪水都增加了，然而这个时候我还没有提交数据变更交易。 现在转到第1个SQLPlus查询实例重新运行查询，结果如下： Oracle不需要用户等待数据更新实例中操作被提交，它径直返回Michael和Pat的查询信息，但实际上返回的是数据更新开始之前的数据视图！ 这时候，熟悉SQL Server的人可能会说了，在查询中设置（NOLOCK）不也能达到同样的效果

吗？可是，对SQL Server而言，在数据映像之前是不能获取数据的。指定（NOLOCK）实际上只是得到了没有提交的数据。Oracle的方法则提供了数据的一致视图，所有的信息都是针对交易的、基于存储数据快照的。 如果在SQLPlus的更新实例中提交更新交易在查询实例中就能看到薪水数据发生变化。如果在查询实例中重新运行先前的查询语句，那么Oracle将返回新的薪水数值。 存储数据快照 说了半天，在给用户显示先前版本的数据同时，Oracle是如何允许其他用户修改数据的呢？其实，只要某一用户启动了一宗修改数据的交易，之前的数据映像就会被写到一个特殊的存储区域。这种“ 前映像 ”用来向任何查询数据的用户提供一致的数据库视图。这样，当其他用户在修改数据的时候，在以上的测试中我们就能看到尚未发生变更的薪金数据。 这个特殊的存储区域在哪里呢？这个问题的答案就跟你正在使用的Oracle版本有关了。在Oracle 8i及其以前版本中会为这一目的创建特殊的回滚段。然而，这种举措会给数据库管理员（DBA）带来管理和调整数据段的工作负担。例如，DBA必须确定为此需要的数据段的数量以及大小等。假如回滚段没有正确配置，那么对交易而言它们就可能不得不排队等待回滚段中出现必要的数据空间。 Oracle 9i就不同了，这是Oracle的最新版本，Oracle实现了一种新特性，这就是所谓的undo表空间，它有效地消除了以上的管理复杂性。虽然回滚段仍然可以继续使用，但是，DBA现在可以选择创建undo表空间的方式令Oracle自己管理“ 前映像 ”的复杂空间分配。 Oracle的这种方法对程序员具有重要意义。因为回滚空间不是无限的，所以，更新交易的数据快照会取代先前交易的映像。因此，如果必要的回滚

段被其他交易的映像覆盖的话。运行时间较长的查询操作就可能产生“ snapshot too old ”错误。 下面举个可能发生的案例。假设在上午11:59的时候某位职员开始更新John Doe帐务的交易。这宗交易在下午12:01被提交。同时，下午12:00某财务经理开始查询所有的客户帐务报表和当月收费总计。因为客户很多，所以这一查询操作很费了点时间，但是不论这次操作到底执行了多久，反正它检索出的结果就是下午12:00数据库中存在的数据。如果包含John Doe帐务前映像的回滚空间在查询执行到该客户名字的时候被覆盖则查询返回错误消息。 Oracle的解决方案当然更为合理，在抽象意义上提供了相比SQL Server更佳的数据一致性。在执行Oracle查询的时候无须担心较长的查询操作会锁定重要的交易。但是，在两种数据库同时支持海量用户的情况下也很难证明Oracle是否就能真正实现具体条件下的数据一致性。 