A Critique of ANSI SQL Isolation Levels 论文翻译

摘要：ANSI SQL-92 [MS，ANSI]根据phenomena（可译为现象，读时现象）定义了隔离级别：脏读，不可重复读和幻读。本文显示，这些phenomena和ANSI SQL定义无法正确表征几个流行的隔离级别，包括对于不同隔离级别标准的锁实现。本文调查了phenomena说明中的歧义，并提出了更正式的说明;此外，介绍了更好地表征隔离类型的新phenomena。最后，定义了一个称为快照隔离的重要的多版本隔离类型。

1.简介

在不同的隔离级别运行并发事务可以让程序员在保证正确性的条件下权衡并发和吞吐量。较低的隔离级别会增加事务并发性，从而有可能让事务遵守模糊或不正确的数据库状态。令人惊讶的是，可以在一些事务在最高的隔离级别（完全可串行化）执行时，并发执行在较低隔离级别运行的事务（读未提交或读过时数据）[GLPT]。当然，以较低隔离级别运行的事务可能会产生无效数据。程序员必须防止以较高隔离级别运行的后续事务访问这些无效数据并传播此类错误。

ANSI / ISO SQL-92规范[MS，ANSI]定义四个隔离级别：（1）读未提交（READ UNCOMMITTED），（2）读已提交（READ COMMITTED），（3）可重复读（REPEATABLE READ），（4）可串行化（SERIALIZABLE）。

这些级别是使用经典的串行化定义和三个禁止的操作子序列来定义的，被命名为phenomena，包括：脏读，不可重复读和幻读。phenomena的概念在ANSI规范中没有明确定义，但是规范表明phenomena是可能导致异常（anomalies）（也许是不可串行化的）行为的操作子序列。当提出对ANSI Phenomena进行添加的建议时，我们将参考这些异常（anomalies）。如后文所示，异常和phenomena之间有一个技术上的区别，但这个区别对于一般的理解并不重要。

ANSI隔离级别与锁调度器（lock schedulers）的行为有关。一些锁调度器允许事务改变其锁请求的范围和持续时间，从而偏离纯粹的两阶段锁。这个想法是由[GLPT]引入的，它以三种角度定义了一致性度（Degree of consistency）：锁，数据流图和异常。通过phenomena（或异常）定义隔离级别旨在允许不基于锁实现的SQL标准。

本文介绍了定义隔离级别的phenomena方法的一些缺点。三个ANSI phenomena是不明确的，即使在最宽松的解释中也不排除执行历史中可能出现的一些异常行为。这导致一些反直觉的结果。特别地，基于锁的隔离级别与等效ANSI phenomena有不同的特性。而商业数据库系统通常使用锁实现隔离级别，所以情况很尴尬。此外，ANSI phenomena不能区分出商业系统中许多类流行的隔离级别的行为。本文提出了表征这些隔离级别的其他phenomena。

第2节介绍了隔离级别的基本术语，定义了ANSI SQL和锁隔离级别。

第3节探讨了ANSI隔离级别的一些缺点，并提出了一个新的phenomena。还定义了其他流行的隔离级别。这些定义映射了ANSI SQL隔离级别和1977年[GLPT]中定义的一致性度。它们还包括Chris Date定义的游标稳定（Cursor Stability）和可重复读[DAT]。在统一框架下讨论隔离级别可以减少独立术语产生的误解。

第4节介绍了一种称为快照隔离的多版本并发控制机制，可以避免使用ANSI SQL phenomena，但不能串行化。快照隔离本身是有趣的，因为它提供了一个位于读已提交和可重复读之间隔离级别。有一种新的方法学（在本次会议文件的较长版本中提供）将多版本数据的隔离级别规约到经典的单版本锁定可串行化（single-version locking serializability）理论。

第5节探讨了一些新的异常来区分第3和第4节引入的隔离级别。这里提出的扩展ANSI SQL phenomena缺乏表征快照隔离和游标稳定的能力。

第6节提出了一个总结和结论。

2.隔离定义

2.1序列化概念

文献[BHG，PAP，PON，GR]中已经很好地说明了事务和锁的概念。接下来的几段会回顾这些文章里使用的术语。

事务（transaction）意为一组合并成组的操作，能将数据库从一个一致的状态转换到另一个状态。历史（history）意为将一组事务的交错执行建模成其操作排序后的线性操作序列，其中操作（operation）意为某特定数据项的读取和写入（如插入，更新和删除）。如果历史中对于相同数据项的两个操作由不同事务执行，并且至少其中一个操作写入数据项（item），则称这两个操作冲突（conflict）（译者注：可看作数据相关）。在[EGLT]之后，该定义对“数据项”的含义进行了广泛的解释：它可以是表行，页面上的空间，整个表或通信对象（例如队列上的消息）。冲突操作也可能发生在由谓词锁（predicate lock）保护的一组数据项上或者单个数据项上。

特定历史能生成出多个事务之间的以时间为轴的数据流依赖图（dependency graph）。历史中提交的事务的操作表示为点。如果事务T1的操作op1与历史中的事务T2的操作op2冲突，则<op1，op2>成为依赖图中的边。（译者注：也就是说以操作为点，冲突为边，构造出依赖图）。如果说两个历史是相等的，那么历史中提交的事务和依赖图都相同。历史是可串行化（Serializable）的，意味着历史可以等同于一个串行历史（serial history）——也就是说，它的依赖图（多个事务间基于时序的数据流图）与依次串行执行多个事务的历史的依赖图相同。

2.2 ANSI SQL隔离级别

ANSI SQL隔离设计人员寻求一种允许许多不同实现的定义，而不仅仅是基于锁的。他们使用以下三种phenomena定义了隔离：

P1（脏读）：事务T1修改数据项。另一个事务T2则在T1执行COMMIT或ROLLBACK之前读取该数据项。如果T1 然后执行ROLLBACK，则T2读取了一个从未提交的数据项。

P2（不可重复读或Fuzzy Read）：事务T1读取数据项。另一个事务T2则修改或删除该数据项并提交。如果T1尝试重新读取数据项，则它会看见修改后的值或发现数据项已被删除。

P3（幻读（Phantom））：事务T1读取满足一些<搜索条件>的一组数据项。事务T2然后创建满足T1的<搜索条件>并提交的数据项。如果T1然后使用相同的<搜索条件>重复读取，它将获得与第一次读取不同的组数据项。

这些现象都不会发生在串行历史上。因为通过串行化定理，它们不能发生在可串行化的历史中[EGLT, BHG Theorem 3.6, G R Section 7.5.8.2, PON Theorem 9.4.2].

由读取，写入，提交（commit）和中止（abort）组成的历史记录可以用简写符号表示：“w1 [x]”表示数据项x上的事务1的写入（数据项被“修改”），而“r2 [x]”表示事务2对x的读取。事务1满足谓词P的读取和写入一组记录分别由r1 [P]和w1 [P]表示。事务1的提交和中止（ROLLBACK）分别被记为“c1”和“a1”。

Phenomena P1可能会被重新表述为不允许以下情况：

（2.1）w1 [x]. . . r2 [x] . . . (a1 and c2 in either order)

P1的语义是不明确的（译者注：语义为第二个提交了的事务读到了第一个中止事务写的数据）。实际上并不强调T1一定要中止;它只是指出，如果这种情况发生可能会发生不一致。一些看过P1说明的人解释为：

（2.2） w1[x]...r2[x]...((c1 or a1) and (c2 or a2) in any order)

P1的（2.2）变体不允许历史中有T1修改数据项x，然后T2在T1提交或中止之前读取数据项这种操作序列。它不强调T1中止或T2提交。

定义（2.2）对P1比（2.1）更为宽松，因为它禁止所有四个可能的事务T1和T2的提交中止顺序，而（2.1）只禁止四个中的两个。解释（2.2）作为P1的含义，禁止在将来出现异常情况的执行顺序。我们称（2.2）是P1的宽松解释（loose interpretation），（2.1）是P1的严格解释。解释（2.2）规定了可能导致phenomena的现象，而（2.1）规定了实际的异常。分别表示为P1和A1。因此：

P1: w1[x]...r2[x]...((c1 or a1) and (c2 or a2) in any order)

A1: w1[x]...r2[x]...(a1 and c2 in any order)

类似地，phenomena P2和P3也具有严格和宽松的解释，P2和P3为宽松解释，A2和A3为严格解释（译者注：P表示异常的读现象的操作序列，A表示造成数据不一致的操作序列）：

P2: r1[x]...w2[x]...((c1 or a1) and (c2 or a2) in any order)

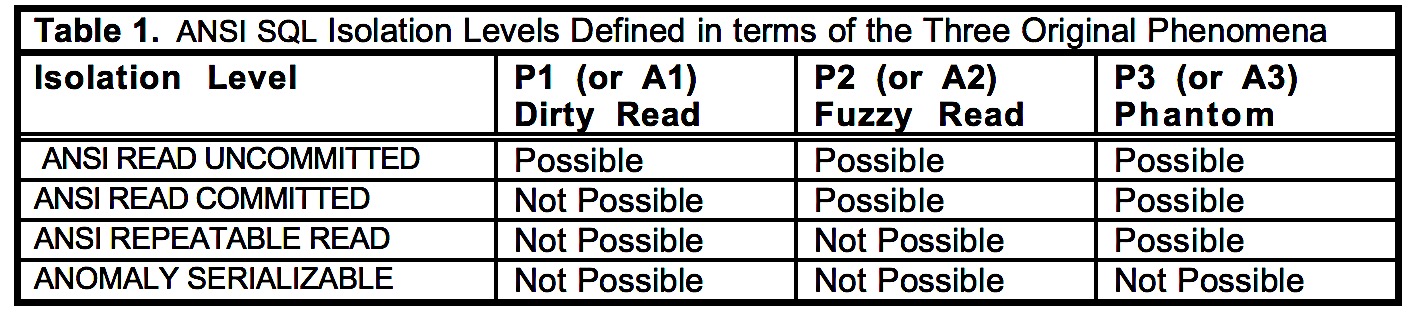
A2: r1[x]...w2[x]...c2...r1[x]...c1

P3: r1[P]...w2[y in P]...((c1 or a1) and (c2 or a2) any order)

A3: r1[P]...w2[y in P]...c2...r1[P]...c1

第3节分析了更多的已经开发出来的替代概念，并提出需要针对phenomena进行宽松的解释。注意到ANSI SQL P3的英文语义只是禁止谓词（范围内的）插入，但是上述P3完全禁止在读满足谓词的元组集合后，对满足谓词的元组进行任何写入（插入，更新，删除）。

本文稍后讨论多值历史（multi-valued history）的概念（MV-history for short，见[BHG]，第5章）。现在不详细介绍，多版本系统中可能会同时存在多个版本的数据项。任何读操作必须明确指定读取哪个版本。已经有文章尝试将ANSI隔离级别定义与多版本系统以及基于标准锁调度器（locking scheduler）的单版本系统（SV-histories）相关联。即使如此，P1，P2和P3 phenomena的语义也仅限于单版本的历史。我们在下一节中解释它们。



ANSI SQL通过表1的矩阵定义了四个级别的隔离。每个隔离级别的特征是事务中禁止发生的（松散或严格的解释的）phenomena。但是，ANSI SQL规范没有仅根据这些phenomena来定义可串行化隔离级别。 “ANSI”中的4.28“SQL事务”指出，可串行化隔离级别必须提供“共识的完全可串行化的执行”。与这个额外的附加条件相比，这个表导致了一个常见的误解，即不允许三个phenomena意味着可串行化。不允许这三种phenomena的历史在表1中被称为异常可串行化（ANOMALY SERIALIZABLE）（译者注：异常可串行化意为基于禁止异常（或phenomena）的可串行化，并非“真正的”可串行化）。

由于在历史中宽松解释远远多于严格解释，而且隔离级别是由它们被禁止经历的phenomena所定义的。我们在第3节中对宽松解释的讨论意味着我们想讨论更多的限制性隔离级别（更多种历史将被禁止）。但是第3节显示，即使对P1，P2和P3进行了宽松的解释，也禁止了这些phenomena，也并不能保证真正的可串行化。在ANSI中删除P3并仅使用4.28定义ANSI 可串行化会更简单。请注意，表1不是最终结果;它将被表3所取代。

2.3锁

大多数SQL产品都使用基于锁的隔离。因此，尽管存在某些问题，但从锁方面表征ANSI SQL隔离级别是有效的。

事务在基于锁调度下执行的读写会请求数据项或数据项集合上读（共享）和写（独占）锁。在两个不同的事务下的锁对应着同一个数据项的情况下，当至少一个是写锁的时候会冲突。

读取（或写入）的谓词锁（给定的<搜索条件>确定的一组数据项下）实际上是对满足<搜索条件>的所有数据项的锁。这可能是一个无限集，因为它包括数据库中存在的数据以及当前不在数据库中的所有幻影（phantom）数据项（如果它们被插入，或者当前数据项被更新以满足<搜索条件> ）。在SQL术语中，谓词锁覆盖满足谓词的所有数据项以及INSERT，UPDATE或DELETE后满足谓词的所有数据项。不同事务的两个谓词锁中如果一个是写锁，并且两个锁覆盖了相同的（可能是幻影）数据项，则两个谓词锁相冲突。数据项（item）锁（记录锁）是一个谓词锁，其中谓词指定特定记录。

事务具有好形式的写（读）（well-formed writes (reads)）要求在写（读）该数据项或谓词定义的数据项集之前，每个数据项或谓词请求写锁（读锁）（译者注：也就是说在读（写）时对指定数据项集进行有且仅有一次的加读（写）锁）。事务是好形式（well-formed）的，要求事务有好形式的读与写。事务具有两阶段写（读）（two-phase writes (reads)）要求在释放写（读）锁之后，在数据项上没有设置新的写（读）锁。事务是两阶段（two-phase）的，要求事务在释放一些锁之后不会请求任何新的锁（读或写锁）。

长锁，要求锁到事务提交或中止为止。否则，为短锁。实际上，短锁通常在操作完成后立即释放。

如果一个事务持有一个锁，另一个事务请求一个冲突的锁，那么在前一个事务的冲突锁已经被释放之前，新的锁请求是不被授予的。

基本的串行化定理是好形式的两阶段锁（well-formed two-phase locking）保证可串行化 ——两阶段锁下的每个历史等同于一些串行历史。相反，如果一个事务不是好形式的或两阶段的，那么（除了在退化的情况下）可能出现不可串行化的执行历史 [EGLT]。 [GLPT]论文定义了四个一致性度，试图说明锁，依赖和基于异常的表征的等价性。异常定义（见定义1）太模糊。其作者在定义上持续受到批评（同见[GR]）。只有在历史和依赖图或锁方面更多的数学定义经受住了时间的考验。

|  |  |  |
| --- | --- | --- |
| **表2 基于锁定义的一致性度和隔离级别** | | |
| 一致性级别=隔离级别 | 数据项和谓词上的读锁 | 数据项和谓词上的写锁 |
| 度（Degree）0 | 不需要 | 好形式的写 |
| 度1=锁读未提交（locking READ UNCOMMITTED） | 不需要 | 好形式的写，长写锁 |
| 度2=锁读已提交（locking READ COMMITTED） | 好形式的读，短读锁 | 好形式的写，长写锁 |
| 游标稳定（Cursor Stability见 4.1 节） | 好形式的读，游标持有读锁，短谓词锁 | 好形式的写，长写锁 |
| 锁可重复读（locking REPEATABLE READ） | 好形式的读，长读锁（对于数据项），短读锁（对于谓词） | 好形式的写，长写锁 |
| 度3=锁可串行化（locking SERIALIZABLE） | 好形式的读，长读锁 | 好形式的写，长写锁 |

表2根据锁定范围（数据项项或谓词），模式（读或写）及其持续时间（短或长）定义了多个隔离类型。我们认为基于锁的隔离级别 锁读未提交、锁读已提交、锁可重复读、锁可串行化是满足ANSI SQL隔离级别要求的，但如图所示，它们与表1完全不同。因此，它是必须将根据锁定义的隔离级别与基于ANSI SQL phenomena的隔离级别进行区分。为了区分，表2中的级别标有“锁”前缀，而不是表1的“ANSI”前缀。

[GLPT]定义了0级一致性以允许脏读和写：只需要操作原子性。度1，2和3分别对应于锁读未提交，锁读已提交和锁可串行化。没有隔离度与锁可重复读匹配。

Date和IBM最初使用名称“Repeatable Reads”[DAT，DB2]表示可串行化或锁可串行化。这似乎是一个比[GLPT]术语“隔离度3”更易理解的名称，尽管它们是相同的。可重复读的ANSI SQL含义与Date的原始定义不同，我们认为这是不幸的。Phenomena P3是ANSI SQL 可重复读隔离级别不考虑的，但从P3的定义可以看出，读取是不可重复的！（译者注：P3 违反了“可重复读”中文词义）我们在表2中仍然误用“锁可重复读”术语，来对应ANSI定义。类似地，Date将术语“游标稳定”作为一个更易于理解的度2隔离名称，增强了对丢失游标更新的保护，如下面4.1节所述。

定义：如果符合L2标准的所有不可串行化历史都满足L1，并且至少存在一个不可串行化的历史发生在L1级，但不在L2级，则隔离级别L1比隔离级别L2弱（weaker）（或L2比L1强（stronger）），表示为L1<L2。当满足L1和L2的不可串行化历史集合相同时，两个隔离级L1和L2是相等的，表示为L1 == L2。 如果L1<L2或L1 == L2表示为L1<=L2。两个隔离级别是无法比较的，表示为L1><L2。比如当每个隔离级别允许另一个不允许的不可串行化的历史。

在比较隔离级别时，我们仅在不可串行化的历史上区分它们，比如一些历史可能发生在一个级别而不可能发生在另一个级别。两个隔离级别在他们允许出现的可串行化历史方面也可能不同。即使众所周知，锁调度程序不接受所有可能的可串行化历史，但是我们认为锁可串行化==可串行化。由于不允许太多种可串行化的历史，这种隔离级别可能是不切实际的，但是我们在这里不考虑这些。

结论如下，这些定义和一些简单的例子表明<关系是合理的。

结论1：锁读未提交<锁读已提交<锁可重复读<锁可串行化

在下一节中，我们将重点比较ANSI和基于锁的隔离级别定义。

3.分析ANSI SQL隔离级别

从好的结论开始，锁隔离级别符合ANSI SQL要求。

结论2：表2的锁协议定义了至少与表1相应的基于phenomena的隔离级别一样强大的锁隔离级别。参考[OOBBGM]的证明。

因此，锁隔离级别至少与同名ANSI级别隔离相当。它们隔离度更强吗？答案是肯定的，即使在最低级别也一样。锁读未提交提供长写锁，以避免我们称之为“脏写”的现象，但ANSI SQL并不排除其基于ANSI 可序列化的异常行为。

P0（脏写）：事务T1修改数据项。另一个事务T2然后在T1执行COMMIT或ROLLBACK之前进一步修改该数据项。如果T1或T2然后执行ROLLBACK，则不清楚正确的数据值应该是什么。宽松的解释是：

P0: w1[x]...w2[x]...((c1 or a1) and (c2 or a2) in any order)

脏写不好的一个原因是它可以违反数据库一致性。假设在x和y之间存在约束（例如，x = y），并且如果单独运行，则T1和T2每个维持约束的一致性。但是，如果两个事务以不同的顺序写入x和y，那么这个约束很容易被违反，这只有在有脏写时才会发生。例如，如果历史是w1 [x] w2 [x] w2 [y] c2 w1 [y] c1，则T1到y和T2到x间修改都是“存活”的。如果T1在x和y两者中写入1同时T2写入2，则结果将为x = 2、y = 1。违反了x = y约束。

如[GLPT，BHG]和其他地方所述，自动事务回滚是说明P0重要性的另一个原因。在没有P0保护的情况下，系统无法通过恢复映像（image）来撤消更新。考虑历史：w1 [x] w2 [x] a1。您不想通过恢复x的前一个映像来撤消w1 [x]，因为这会覆盖w2的更新。但是，如果您不还原其前映像，并且事务2稍后中止，则无法通过还原其前映像来撤消w2 [x]！这就是为什么即使是最弱的锁系统也要持有长写锁。否则，他们的恢复系统将失效。

结论3：ANSI SQL隔离应修改为要求所有隔离级别至少有P0。

我们认为，需要对三种ANSI phenomena进行宽松的解释。回想一下，严格的解释是：

A1: w1[x]...r2[x]...(a1 and c2 in either order) (Dirty Read)

A2: r1[x]...w2[x]...c2...r1[x]...c1 (Fuzzy or Non-Repeatable Read)

A3: r1[P]...w2[y in P]...c2....r1[P]...c1 (Phantom)

之前的表1显示，读已提交隔离的历史禁止异常A1，可重复读隔离的历史禁止异常A1和A2，可串行化隔离的历史禁止异常A1，A2和A3。考虑历史H1，在银行余额行x和y之间转移$ 40：

H1：r1 [x = 50] w1 [x = 10] r2 [x = 10] r2 [y = 50] c2 r1 [y = 50] w1 [ y = 90] c1

H1是不可串行化的，这是一个经典的不一致分析（inconsistent analysis）问题。其中事务T1将40元从x转移到y，要求保持余额总数为100，但T2读到了总余额为60的不一致状态。历史H1不违反任何异常A1，A2或A3。在A1的情况下，两个事务之一将不得不中止（译者注：也就是读未提交要求一个事务读，另一个事务终止提交，但在H1中两个事务都提交了）;对于A2，数据项将必须由相同的事务读取第二次; A3需要谓词范围内的值改变。这些事情都不会发生在H1。考虑到A1的宽松解释，phenomena P1：

P1: w1[x]...r2[x]...((c1 or a1) and (c2 or a2) in any order)

H1显然违反了P1。因此，我们认为ANSI定义意图说明的是解释P1而不是A1，宽松的解释是正确的解释。

类似的论点表明，P2应该被视为ANSI的定义而不是A2。区分这两个解释的历史是：

H2：r1 [x = 50] r2 [x = 50] w2 [x = 10] r2 [y = 50] w2 [y = 90] c2 r1 [y = 90]

H2是不可串行化的——这是另一个经典不一致分析，其中T2看到总余额为140.这一次，交易都没有读取脏（即未提交）的数据。因此P1满足。并且，没有任何数据项被读取两次，也没有谓词范围内的数据被更改。 H2的问题是，当T1读取y时，x的值已过期。如果T2再次读取x，则会被更改;但由于T2不会读两次，A2不适用。用P2代替A2，宽松解释解决了这个问题。

P2: r1[x]...w2[x]...((c1 or a1) and (c2 or a2) any order)

当w2 [x = 20]发生时，由于覆盖了r1 [x = 50]，H2将不再正确。最后，考虑A3和历史H3：

A3：r1 [P] ... w2 [y in P] ... c2 ... r1 [P] ... c1（Phantom）

H3：r1 [P] w2 [insert y到P] r2 [z] w2 [z] c2 r1 [z] c1

这里T1执行<搜索条件>以找到雇员的列表。然后T2执行新的员工的插入，然后更新公司中的员工数量z。此后，T1将员工的数量读出，并看到差异。这个历史显然是不可串行化的，但由于谓词范围没有被访问两次，所以它是被A3所允许的。同样，宽松解释P3解决了这个问题。

P3: r1[P]...w2[y in P]...((c1 or a1) and (c2 or a2) any order)

如果P3被禁止，历史H3无效。这显然是ANSI的意图。上述讨论有助于证明以下结果。

结论4.严格的解释A1，A2和A3有意想不到的缺点。正确的解释是宽松的。我们认为ANSI意在定义P1，P2和P3。

结论5. ANSI SQL隔离phenomena不完整。还有一些异常仍然可能出现。必须定义新的phenomena来完成锁的定义。此外，必须重新进行定义P3。在以下定义中，我们删除不加强历史限制的操作（c2 or a2）。

P0: w1[x]...w2[x]...(c1 or a1) (Dirty Write)

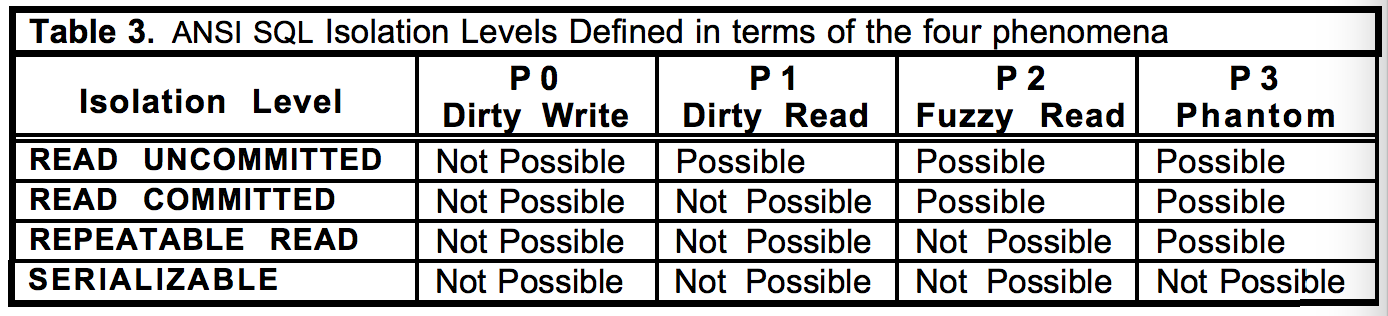
P1: w1[x]...r2[x]...(c1 or a1) (Dirty Read)

P2: r1[x]...w2[x]...(c1 or a1) (Fuzzy or Non-Repeatable Read)

P3: r1[P]...w2[y in P]...(c1 or a1) (Phantom)

需要注意的是，ANSI SQL P3只禁止对谓词插入（和更新），而上面的P3的定义禁止任何满足谓词的写被读取——写可以是插入，更新或删除。

关于这些phenomena的提出的ANSI隔离级别的定义在表3中给出。



对于单个版本历史，容易得出P0，P1，P2，P3 phenomena是“假”的锁版本phenomena。例如，禁止P0排除了在第一个事务写入数据项后第二个事务的写，相当于在在数据项（和谓词）上持有长写锁。所以脏写是不可能的。类似地，禁止P1相当于对数据项进行了好形式的读取。禁止P2表示数据项加上长读锁。最后，禁止P3意味着持有长谓词读锁。因此，由这些phenomena定义的表3的隔离级别提供与表2的锁隔离级别相同的行为。

结论6.表2的锁定隔离级别和表3基于phenomena的定义是等效的。换句话说，P0，P1，P2和P3是对于锁版本隔离级别的重新定义。

接下来，我们将通过表3中的名称来引用表3中列出的隔离级别，相当于表2的这些隔离级别的锁定版本。当我们参考ANSI READ UNCOMMITTED，ANSI READ COMMITTED，ANSI REPEATABLE READ和ANOMALY SERIALIZED，我们指的是表1的ANSI定义（因不包括P0而不完备）。

下一节显示，许多商业上可用的隔离实现提供了落在读已提交和可重复读之间的隔离级别。为了实现区分这些实现的有意义的隔离级别，我们以P0和P1作为基础，然后添加新的phenomena。

4.其他隔离类型

4.1游标稳定（Cursor Stability）

游标稳定旨在防止丢失更新现象。

P4（丢失更新（Lost Update））：当事务T1读取数据项，然后T2更新数据项（可能基于先前的读取），然后T1（基于其较早的读取值）更新数据项并提交时，会发生丢失的更新异常。在从历史上考虑，这是：

P4: r1[x]...w2[x]...w1[x]...c1 (Lost Update)

（译者注：注意P4只是基于P0脏写和P1脏读，从锁隔离的角度上来说只是持有短读锁和长写锁，没有达到锁可重复读P2（也就是说不持有长读锁））

如历史H4所示，问题是即使T2提交，T2的更新也会丢失。

H4：r1 [x = 100] r2 [x = 100] w2 [x = 120] c2 w1 [x = 130] c1

x的最终值为T1写入的x+30=130。 P4至少在读已提交隔离级别，因为禁止P0（事务执行第一次写操作的数据项被另一个事务第二次写入）或P1（写后提交前被读取）的情况下允许出现H4。当然，禁止P2也排除了P4，因为P2是r1[x],w2[x]，（c1 or a1），包括了P4。因此，P4可用于作为区分读已提交和可重复读强度中间的隔离级别。

游标稳定扩展了读已提交隔离级别下对于SQL游标的锁行为。其提出游标上的Fetching操作rc（意思是读取游标）。rc要求在游标的当前数据项上保持长读锁，直到游标移动或关闭（可能通过提交关闭）。当然，游标上的Fetching事务可以更新行（wc），即使游标在随后的Fetch上移动，写锁也将保持在行上直到事务提交。 rc1 [x]和以后的wc1 [x]排除了介入的w2 [x]。因此，针对游标上的情况，提出phenomena P4C。

P4C：rc1 [x] ... w2 [x] ... w1 [x] ... c1（丢失更新）

结论7：读已提交<游标稳定<可重复读

游标稳定被SQL系统广泛实现，以防止丢失通过游标读取的行的更新。在某些系统中的读已提交，实际上仅是更强的游标稳定。ANSI标准允许这一点。

将游标放在数据项上以保持其值稳定的技术可以用于多个数据项，代价是使用多个游标。因此，程序员可以将访问稳定性调整为锁读已提交隔离，以便访问少量固定数量的数据项。然而，这种方法不方便也不通用。因此，总是有历史适合P4（当然适用于更通用的P2）phenomena，处于游标稳定隔离级别上。

4.2快照隔离（Snapshot Isolation，SI）

在快照隔离下执行的事务始终从事务开始时起的（已提交）数据的快照中读取数据。事务开始时获取的时间戳称为其开始时间戳（Start-Timestamp）。这一个时间戳可能为事务第一次读之前的任何时间。事务运行在快照隔离中时，只要可以维护其开始时间戳对应的快照数据，在就不会阻塞读。事务的写入（更新，插入和删除）也将反映在此快照中，如果事务第二次访问（即读取或更新）数据，则能再次读到。这个事务开始时间戳之后的其他事务的更新对于本次事务是不可见的。

快照隔离是一种多版本并发控制（Multiversion Concurrency Control，MVCC）。它扩展了[BHG]中描述的多版本混合方法（Multiversion Mixed Method）。其允许通过只读事务进行快照读（snapshot read）。

当事务T1准备好提交时，它将获得一个提交时间戳（Commit-Timestamp），该值大于任何现有的时间戳。当其他事务T2提交了的数据的提交时间戳在T1事务的间隔[Start-Timestamp，Commit-Timestamp]中，只有T1, T2数据不重叠，事务才成功提交。否则，T1将中止。这个功能叫做先提交者成功（First-committer-wins），防止丢失更新（P4）。当T1提交时，其更改对于开始时间戳大于T1的提交时间戳的所有事务都可见。

快照隔离是一种多版本（MV）方法，因此单值（SV）历史不能正确地反映时间上的操作序列。在任何时候，每个数据项可能有多个版本，由活动的和已提交的事务写入。事务必须读取合适的版本。考虑第3节开始的历史H1，其表明在单值执行中需要P1。在快照隔离下，相同的操作序列将导致多值历史：

H1.SI： r1 [x0 = 50] w1 [x1 = 10] r2 [x0 = 50] r2 [y0 = 50] c2

r1 [y0 = 50] w1 [y1 = 90] c1

（译者注：事务1读到x0=50，写x1=10，之所以事务2读的是x0的值，是因为事务1还没有获得提交时间戳并提交x1，从而事务2的开始时间戳在一定在事务1的提交时间戳之前。最终没有脏读）

但是H1.SI具有可串行化执行的数据流。在[OOBBGM]中，我们说明了所有快照隔离历史都可以映射到单值历史，同时保留数据流依赖性（MV历史被视为与SV历史视图等效，转换的方法在[BHG]第5章中）。例如，MV历史H1.SI将映射到可串行化的SV历史：

H1.SI.SV：r1 [x = 50] r1 [y = 50] r2 [x = 50] r2 [y = 50] c2

w1 [x = 10] w1 [y = 90] c1

将MV历史映射到SV历史是在隔离层次中放置快照隔离的关键。

快照隔离是不可串行化的，因为事务的读在一个时刻，写在另一个时刻。例如，考虑单值历史：

H5：r1 [x = 50] r1 [y = 50] r2 [x = 50] r2 [y = 50] w1 [y = -40] w2 [x = -40] c1 c2

H5是不可串行化的（译者注：但可以看出，如果SI读写数据项集相同，则可串行化），并且具有与快照隔离下事务相同的事务间数据流（事务读取的版本没有选择）。这里我们假设为x和y写入一个新值的每个事务有保持x + y>0的约束，而T1和T2两者都是隔离的，所以约束不能保持在H5中。

约束违反（Constraint violation）是一种通用和重要的并发异常类型。个别数据库满足多个数据项的约束（例如，键的唯一性，引用完整性，两个表中的行的复制关系等）。它们一起形成数据库不变量约束谓词C（DB）。如果数据库状态DB与约束一致，则不变量为TRUE，否则为FALSE。事务必须保留约束谓词以保持一致性：如果数据库在事务启动时保持一致，则事务提交时数据库将一致。如果事务读取到违反约束谓词的数据库状态，则事务将受到约束违反并发异常的影响。这种约束违反在[DAT]中称为不一致分析（inconsistent analysis）。

A5（数据项约束违反）。假设C（）是数据库中两个数据项x和y之间的数据库约束。这里提出两种由于约束违反引起的异常。

A5A 读偏（Read Skew）假设事务T1读取x，然后第二个事务T2将x和y更新并提交。如果现在T1读取y，它可能会看到不一致的状态。就历史而言，我们有异常：

A5A：r1 [x] ... w2 [x] ... w2 [y] ... c2 ... r1 [y] ...（c1 or a1）（读偏）

A5B 写偏（Write Skew）假设T1读取与C（）一致的x和y，然后T2读取x和y，写入x和提交。然后T1写y。如果x和y之间存在约束，则可能会被违反。在历史方面：

A5B：r1 [x] ... r2 [y] ... w1 [y] ... w2 [x] ...（c1 and c2 occur）（写偏）

Fuzzy Read（P2）是读偏的退化形式，其中x = y。更典型地，事务读取两个不同但相关的项目（如引用完整性）。写偏（A5B）可能来自银行业务语义的约束，如只要总共持有的余额保持非负，账户余额才能变为负值。如历史H5中出现的异常。

在排除P2的历史中，A5A和A5B都不会出现，因为A5A和A5B都有T2写入一个先前未被提交的T1读取的数据项的情况。因此，phenomena A5A和A5B仅用于区分低于可重复读取的隔离级别。

ANSI SQL定义中可重复读的在其严格的解释中捕获了行约束的退化形式，但在概念上有缺失。具体来说，表2的锁可重复读提供了对行约束违反的保护，但表1的ANSI SQL定义仅仅禁止异常A1和A2，不提供对行约束违反的保护。

现在回到快照隔离，这个隔离级别十分强，比读已提交更强。

结论 8.读已提交<快照隔离

证明：在快照隔离中，first-committer-wins排除了P0（脏写入），并且时间戳机制阻止了P1（脏读），因此快照隔离不比读已提交弱。此外，A5A可能在读已提交下，但不在快照隔离与时间戳机制下。因此读已提交<快照隔离。

注意到，在单值解释中，难以描述快照隔离历史如何违反现象P2。异常A2不能发生，因为快照隔离下的事务即使在另一个事务更新数据项之后也会只读取数据项的相同版本对应的值。然而偏写（A5B）显然会发生在快照隔离下（比如H5），并且在单值历史解释中已经提到，禁止了P2也会排除A5B。因此，快照隔离承认可重复读没有历史异常。

快照隔离下不会发生A3异常（幻读）。在一个事务更新数据项集时，另一个事务多次谓词读的将始终看到相同的旧数据项集。但是可重复读隔离级别可能会遇到A3异常。快照隔离禁止具有异常A3的历史，但允许A5B，而可重复读则相反（允许A3禁止A5B）。因此：

结论9：可重复读><快照隔离。

但是，快照隔离（能排除A3却）并不排除P3（谓词读事务提交前谓词范围内被另一事务写入）。考虑一个约束，表示由谓词确定的一组作业任务不能有大于8的小时数。T1读取此谓词，确定总和只有7小时，并添加1小时持续时间的新任务，而并发事务T2做同样的事情。由于两个事务正在插入不同的数据项（以及不同的索引条目（如果有的话）），因此First-Committer-Wins不排除此情况，并且可能发生在快照隔离中。但是在任何等价的串行历史中，在这种情况下会出现P3现象。

也许最引人注目的是，快照隔离没有幻读（在ANSI定义A3的严格意义上）。每个事务都不会看到并发事务的更新。所以，可以说出以下令人惊讶的结果（请注意，表1定义了异常可串行化（ANOMALY SERIALIZABLE）作为ANSI SQL定义的SERIALIZABLE），而没有[ANSI]中的4.28中的额外限制：

结论10：快照隔离历史排除了异常A1，A2和A3 。因此，在表1 异常可串行化的异​​常解释中：可串行化<快照隔离。

快照隔离允许以非常老的时间戳运行事务，从而从数据库的历史允许进行“时间旅行”——过程中写入不会被阻塞。当然，具有非常旧的时间戳的事务，如果尝试更新任何由比较新的事务更新过的数据项，将会中止。

快照隔离的简单实现模仿了Reed [REE]。这种多版本数据库有几种商业实现。 Borland的InterBase 4 [THA]和基于Microsoft Exchange系统的引擎都提供了快照隔离与First-committer-wins功能。 First-Committer-wins要求系统记住属于每个活动事务的开始时间戳之后提交的任何事务的所有更新（写入锁信息）。如果其更新与系统记忆中其他人的已有的更新冲突，则中止该事务。

快照隔离的“乐观”并发控制方法对于只读事务具有明显的并发优势，但其对更新事务的好处仍然存在争议。这可能不利于长时间运行的与高竞争性短事务执行，因为长期运行的事务不太可能是他们写的一切的第一作者，因此可能会被中止。（请注意，这种情况也会成为锁实现中的一个真正的问题，如果解决方案是不允许长期运行会持有短事务锁的更新事务，快照隔离也是可以接受的。）当然，当在短期更新事务冲突最小化，长时间运行的事务大多为只读事务的情况下，快照隔离应该有较好的效果。在高数据冲突的长事务中，快照隔离提供了一种古典乐观的方法。对此种情况下的快照隔离的价值存在不同的看法。

4.3其他多版本系统

还有其他模式的多版本并发控制。一些商业数据库维护对象的版本，但将快照隔离限制为只读事务（例如，仅在某些其他数据库中的SQL-92，Rdb和[MS，HOB，ORA]的SET TRANSACTION READ; Postgres和Illustra [STO，ILL]长期维护多版本，并提供“时间旅行”查询）。其他产品允许更新事务，但不保证first-committer-wins（例如，Oracle读一致性隔离[ORA]）。

Oracle读一致性（Read Consistency）隔离在每个SQL语句开始后，根据事务开始时间戳，给出最近提交的数据库值。就好像在每个SQL语句中事务的开始时间戳都是最新的。启动游标的时间作为游标集的成员。底层机制根据语句时间戳重新计算行的适当版本。写锁保护了行插入，更新和删除，以提供first-writer-wins而不是first-committer-wins策略。读一致性比读已提交（它禁止游标丢失更新（P4C））更强，但允许不可重复读取（P3），丢失更新（P4）和读偏（A5A）。快照隔离不允许P4或A5A。

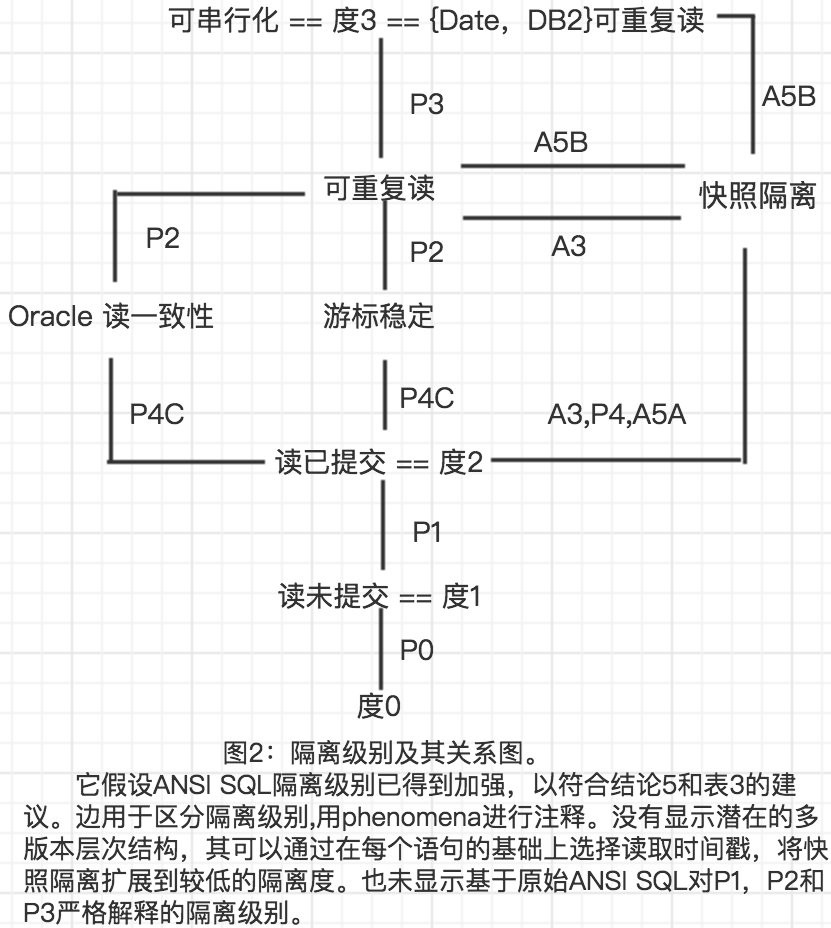
如果仔细查看SQL标准，它将每个语句定义为原子操作。它在每个语句的开头都有可串行化的子事务（或时间戳）。可以想象出一种隔离级别层次，其通过为语句分配时间戳而定义的隔离级别（例如，在Oracle中，一次游标fetch具有其启动的时间戳）。

5.总结

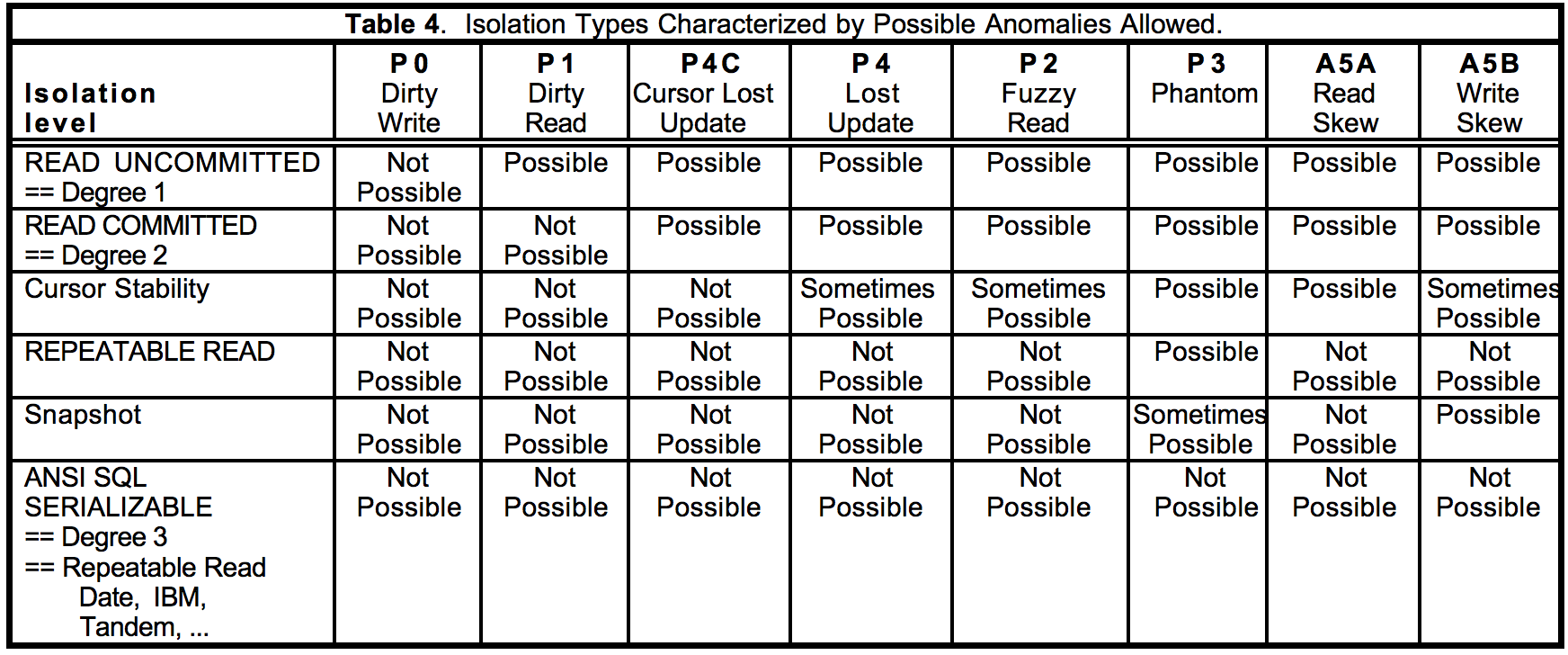
总之，原始ANSI SQL隔离级别的定义存在严重问题（如第3节所述）。英文文字上的定义是模糊和不完整的。脏写（P0）没有被排除。结论5是我们建议清理ANSI 隔离级别以对应锁隔离级别[GLPT].

ANSI SQL希望定义可重复读隔离以排除除幻读之外的所有异常。表1的异常定义没有达到这个目标，但是表2的锁版本定义做到了。 ANSI对术语“可重复读”的选择是非常不合适的：（1）可重复的读取不会产生可重复的结果，（2）行业已经使用该术语来表示：可重复的读取意味着可序列化——几种产品术语中都是这么认为的。我们建议为此找到另一个术语。

许多商业产品中流行的隔离级别处于表3的可重复读和可串行化之间。第4节中进行了阐述。这里命名的所有隔离级别已经被表征如图2所示和表4，如下。图2中较高级别的隔离度在强度上较高（参见4.1节开头的定义），连接线标有区别它们的phenomena和异常。



值得注意的是，从来没有文章讨论过降低多版本系统的隔离级别，尽管其已经在几个产品中实现。许多应用程序通过使用游标稳定性或Oracle的一致性隔离来避免锁争用。这样的应用程序会发现快照隔离比两者更好：它避免了丢失的更新异常，一些幻读异常（例如ANSI SQL所定义的）。它不会阻塞只读事务，并且读不阻塞更新。



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 表4：由可以允许的异常与隔离级别 | | | | | | | | |
| 隔离级别 | P0  脏写 | P1  脏读 | P4C  游标丢失更新 | P4  丢失更新 | P2  不可重复读 | P3  幻读 | A5A  读偏 | A5B  写偏 |
| 读未提交==度1 | 不可能 | 可能 | 可能 | 可能 | 可能 | 可能 | 可能 | 可能 |
| 读已提交==度2 | 不可能 | 不可能 | 可能 | 可能 | 可能 | 可能 | 可能 | 可能 |
| 游标稳定 | 不可能 | 不可能 | 不可能 | 有时可能 | 有时可能 | 可能 | 可能 | 有时可能 |
| 可重复读 | 不可能 | 不可能 | 不可能 | 不可能 | 不可能 | 有时可能 | 不可能 | 不可能 |
| ANSI SQL 可串行化==度3==可重复读(Date, IBM,Tandem…) | 不可能 | 不可能 | 不可能 | 不可能 | 不可能 | 不可能 | 不可能 | 不可能 |

声明（略）

引用

[ANSI] ANSI X3.135-1992, American National Standard for Information Systems — Database Language — SQL, November, 1992

[ABJ] V. Atluri, E. Bertino, S. Jajodia, "A Theoretical Formulation for Degrees of Isolation in Databases," Technical Report, George Mason University, Fairfax, VA, 1995

[BHG] P. A. Bernstein, V. Hadzilacos, N. Goodman, “Concurrency Control and Recovery in Database Systems,” Addison-Wesley 1987.

[DAT] C. J. Date, “An Introduction to Database Systems,” Fifth Edition, Addison-Wesley, 1990 [DB2] C. J. Date and C. J. White, “A Guide to DB2,” Third Edition, Addison-Wesley, 1989.

[EGLT] K. P. Eswaran, J. Gray, R. Lorie, I. Traiger, “The Notions of Consistency and Predicate Locks in a Database System,” CACM V19.11, pp. 624- 633, Nov. 1978

[GLPT] J. Gray, R. Lorie, G. Putzolu and, I. Traiger, “Granularity of Locks and Degrees of Consistency in a Shared Data Base,” in Readings in Database Systems, Second Edition, Chapter 3, Michael Stonebraker, Ed., Morgan Kaufmann 1994 (originally published in 1977).

[GR] J. Gray and A. Reuter, “Transaction Processing: Concepts and Techniques,” Corrected Second Printing, Morgan Kaufmann 1993, Section 7.6 and following.

[HOB] L . Hobbs and K. England, “Rdb/VMS, A Comprehensive Guide,” Digital Press, 1991.

[ILL] Illustra Information Technologies, "Illustra User's Guide," , Illustra Information Technologies, Oakland, CA. 1994.

[MS] J. Melton and A. R. Simon, “Understanding The New SQL: A Complete Guide,” Morgan Kaufmann 1993.

[OOBBGM] P. O'Neil, E. O'Neil, H. Berenson, P. Bernstein, J. Gray, J. Melton, “An Investigation of Transactional Isolation Levels,” UMass/Boston Dept. of Math & C.S. Preprint.

[ORA] “PL/SQL User’s Guide and Reference, Version 1.0,” Part. 800-V1.0, Oracle Corp., 1989.

[PAP] C. Papadimitriou, “The Theory of Database Concurrency Control, Computer Science Press, 1986.

[PON] P. O’Neil, “Database: Principles, Programming, Performance, Morgan Kaufmann 1994, Section 9.5.

[REE] D. Reed, “Implementing Atomic Actions On Decentralized Data,” ACM TOCS 1.1,1981, pp. 3- 23.

[STO] M. J. Stonebraker, “The Design of the POSTGRES Storage System,” 13th VLDB, 1987, reprinted in Readings in Database Systems, Second Edition, M. J. Stonebraker, Ed., Morgan Kaufmann 1994

[THA] M. Thakur, “Transaction Models in InterBase 4,” Proceedings of the Borland International Conference, June 1994.