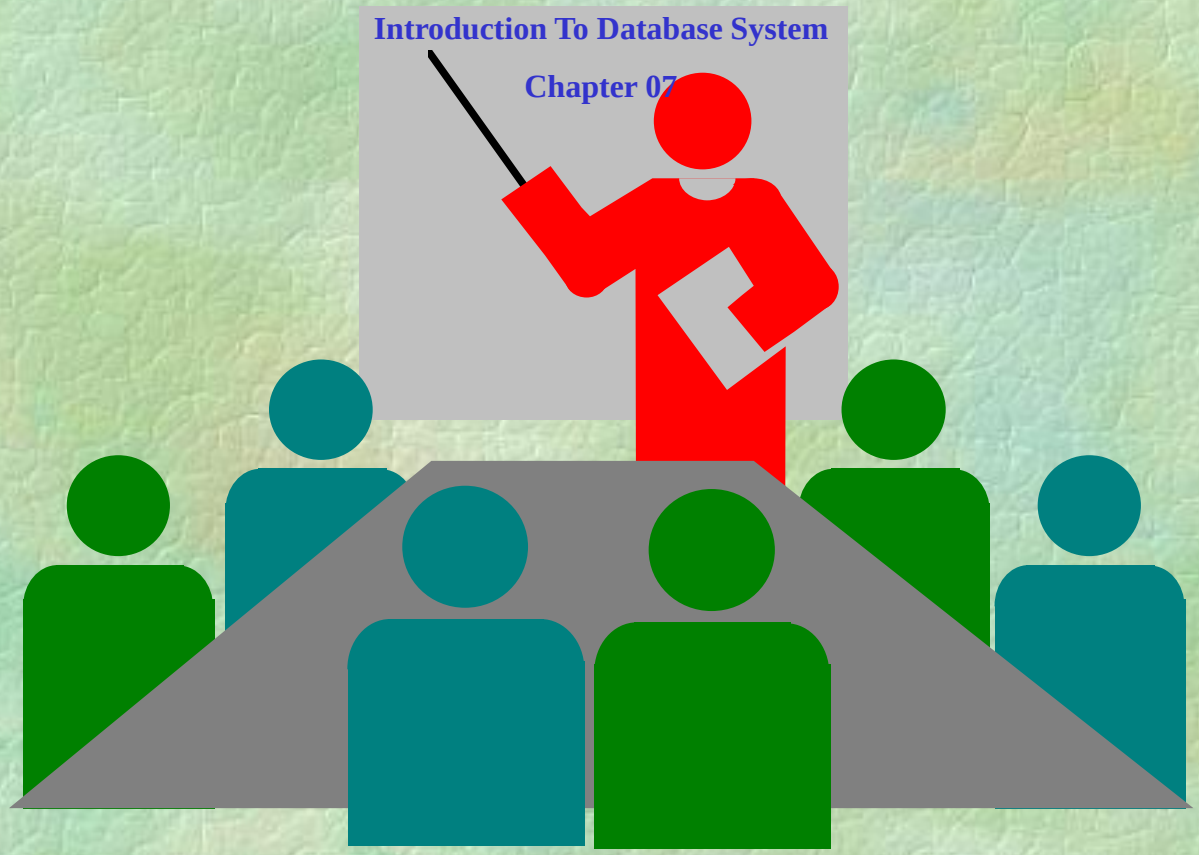
第七章　数据库恢复技术



哈爾濱工業大學 数据库系统概论 -- 系统篇

HARBIN INSTITUTE OF TECHNOLOGY

7.1 事务的基本概念

事务（Transcation）：

用户定义的一个数据库操作序列，这些操作要么全做要么全不做，是一个不可分割的工作单位。

在关系数据库中，一个事务可以是一条SQL语句，也可以是一组SQL语句或者整个程序。

注意：事务和程序是两个不同的概念。通常，一个程序中会包含有多个事务。

事务的开始与结束可以由用户显示控制，如果没有显示地定义事务，则由DBMS按照缺省规定自动划分事务。

SQL中定义事务的语句

在SQL-92规范中，定义事务的三条语句如下：

```
BEGIN TRANSACTION
COMMIT
ROLLBACK
```

在SQL Server 2000中，定义事务的相关语句如下：

```
BEGIN TRAN （事物名）
COMMIT TRAN （事物名）
ROLLBACK TRAN （事物名）
```

在SQL Server2000中，也支持SQL-92规范中的说明方式。

关于事务定义的几点说明

事务通常是以BEGIN TRANSACTION 开始，以COMMIT或ROLLBACK结束。

其中，

COMMIT表示提交事务的所有操作，即：将事务中所有对数据库的更新写回到磁盘上的物理数据库中去，事物才算正常结束。

ROLLBACK表示回滚，即在事务运行的过程中发生了某种故障，事务不能继续执行，系统将事务中对数据库的所有已完成的操作（指更新操作）全部撤销，回滚到事务开始时的状态。

事务的ACID特性

事务的ACID特性是指：任何一个事务都具有如下的四个基本特点：

原子性（*A*tomicity）；

一致性（*C*onsistency）；

隔离性（*I*solation）；

持续性（*D*urability）；

原子性和一致性

原子性：

事务是数据库的逻辑工作单位，事务中包括的相关操作要么都做，要么都不做。

一致性：

事务执行的结果必须使数据库从一个一致性状态变到另一个一致性状态。因此，当数据库只包含成功事务提交的结果时，就认为数据库是处于一致性状态的。如果数据库系统运行过程中发生故障，有些事务尚未完成就中断，这些未完成的事务对数据库所做的修改有一部分已写入物理数据库，这时就认为数据库处于一种不正确的状态，或者说是不一致的状态。

（请参考教材中249给出的关于银行转账的实例）

隔离性和持续性

隔离性：

一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对其他并发事务是相隔离的，并发执行的各个事务之间不能互相干扰。

持续性（又称为永久性）：

一个事务一旦提交，它对数据库中数据的改变就是永久性的。后续的其他操作或者故障不应该对该事务的执行结果有任何影响。

关于事务的几点说明

事务是数据库恢复和并发控制的基本单位。保证事务的ACID特性是事务处理的重要任务。事务ACID特性可能遭到破坏的因素有：

（1）多个事务并发运行时，不同事务的操作交叉执行；

（2）事务在运行过程中被停止 。

在第一种情况下，DBMS必须保证多个事务的交叉执行不影响这些事务的原子性。

在第二种情况下，DBMS必须保证被停止的事务对数据库和其他事务没有任何影响。

7.2 数据库恢复概述

尽管数据库系统中采取了各种保护措施来防止数据库的安全性和完整性被破坏，保证并发事务的正确执行，但是计算机系统中硬件的故障、软件的错误、操作员的失误以及恶意的破坏仍是不可避免的，这些故障可能造成运行事务非正常中断，影响数据库中数据的正确性，严重时可能破坏数据库，使数据库中全部或部分数据丢失，因此**DBMS必须具有把数据库从错误状态恢复到某一已知的正确状态**（指一致状态或完整状态）**的功能，这就是数据库的恢复。**

数据库系统采用的恢复技术是否有效，将直接决定系统的可靠性，而且还将影响系统的运行效率。是衡量系统性能的重要指标。

7.3 故障的种类

数据库系统中可能发生各种各样的故障，大致可以分为以下几类：

事务内部的故障；

系统故障；

介质故障；

计算机病毒故障；

事务内部的故障

事务内部的故障有的是可以通过事务程序本身发现的，有的却是非预期的，不能由事务程序处理：

例：银行转账事务：将一笔资金从一个账户转账到另一个账户。

注意：

我们通过SQL Server2000中实现该转账事务。实现方法是：首先定义一个存储过程，然后在存储过程中实现转账事务。提供三个输入参数，如：转出和转入账号，转出的资金额。

- BEGIN TRANSACTION
- 读账户甲的余额BALANCE;
- BALANCE= BALANCE-AMOUNT (AMOUNT为转账金额)
- IF(BALANCE<0) THEN
- {打印“金额不足，不能转帐”；
- ROLLBACK; (撤销该事务)
- ELSE
- 写回BALANCE;
- {读账户乙的余额BALANCE1;
- BALANCE1= BALANCE1+AMOUNT;
- 写回BALANCE1;
- COMMIT;}

事务内部的故障（续）

事务内部更多的故障都是非预期的，是不能由应用程序处理的。如：运算溢出、并发事务发生死锁而被选中撤销该事务、违反了某些完整性限制等。

事务故障：

指事务没有达到预期的终点（即COMMIT或者显示的ROLLBACK）。

当发生事务故障时，数据库可能处于不正确状态。恢复程序要在不影响其他事务运行的情况下，强行回滚该事务，即撤销该事务已经做出的任何对数据库的修改，使得该事务好像根本就没有被执行过一样。我们把这类恢复操作称之为事务撤销（UNDO）。

系统故障

系统故障是指造成系统停止运转的任何事件，使得系统要重新启动。如：特定类型的硬件故障、操作系统故障、DBMS代码错误、突然停电等。这类故障影响正在运行的所有事务，但是不破坏数据库。

在这种情况下，一些尚未完成的事务的结果可能已送入物理数据库，从而造成数据库可能处于不正确的状态。为保证数据的一致性，需要清除这些事务对数据库的所有修改。

恢复子系统必须在系统重新启动时，让所有非正常终止的事务回滚，强行撤销所有未完成的事务，或者重做所有已提交的事务。从而使数据库真正恢复到一致性状态。

介质故障与计算机病毒故障

介质故障是指外存故障，如：磁盘损坏、磁头碰撞、瞬时强磁场干扰等。

这类故障将破坏数据库或部分数据库，并影响正在存取这部分数据的所有事务。它的破坏性是最大的。甚至是不可恢复的。

计算机病毒故障是一种人为故障或破坏。它已成为计算机系统的主要威胁，也是数据库系统的主要威胁之一。在这种情况下，数据库一旦被破坏，也要用恢复技术对数据库进行恢复。

各种故障小结

总结各类故障，对数据库的影响有两种可能性：一是数据库本身被破坏；二是数据库没有破坏，但是数据库中的数据不正确。

恢复的基本原理十分简单，那就是：数据库中任何已部分被破坏的或者不正确的数据可以根据存储在系统别处的冗余数据来重建。

7.4 恢复的实现技术

数据库的恢复机制中涉及两个关键问题，它们分别是：

第一、如何建立冗余数据；

第二、如何利用冗余数据实施数据库恢复；

建立冗余数据最常用的技术是数据转储和登录日志文件。在应用系统中，通常是两种方法都采用。

7.4.1 数据转储

数据转储是数据库恢复技术中采用的基本技术。转储是指DBA定期地将整个数据库复制到磁带或者另一个磁盘上保存起来的过程。称这些备用的数据文本为后备副本。

当数据库遭到破坏后可以将后备副本重新装入，但重装后备副本只能将数据库恢复到转储时的状态，要想恢复到故障发生时的状态，必须重新运行自转储以后的所有更新事务。

转储十分耗费时间和资源，不能频繁的进行，因此DBA应该根据数据库使用情况确定一个适当的转储周期。

转储可分为静态转储和动态转储两种。

静态转储

静态转储：

它是在系统中无运行事务时进行的转储操作。即：转储操作开始的时刻，数据库处于一致性状态，而转储期间不允许（或不存在）对数据库的任何存取、修改活动。因此，静态转储得到的一定是一个数据一致性的副本。

静态转储相对简单，但是转储必须等待正在运行的用户事务结束才能进行，同样，新的事务必须等待转储结束后才能执行。因此，静态转储将会降低数据库系统的可用性。

那么，能不能有一种可以方法，可以使得转储过程中，同时还能执行事务呢？这就是动态转储。

动态转储

动态转储：

转储期间允许对数据库进行存取或者修改。即转储和用户事务可以并发执行。

因此，动态转储可以克服静态转储的缺点，它不用等待正在运行的用户事务结束，也不会影响新事务的允许，但是转储结束时得到的后备副本上的数据并不能保证正确有效。

所以**必须把转储期间各事务对数据库的修改活动登记下来，建立日志文件**。这样，**后备副本加上日志文件就能把数据库恢复到某一时刻的正确状态**。

数据转储（续）

数据转储还可以分为海量转储和增量转储两种。海量转储是指每次转储全部数据库。增量转储是指每次只转储上一次转储后更新过的数据。

从恢复角度来看，利用海量转储得到的后备副本进行恢复是会比较方便一些，但是如果数据库很大，事务处理又十分频繁，则增量转储方式更实用、更有效。

综上所述，数据转储有两种方式，分别可以在两种状态下进行，因此，数据转储方法可以分为四类：**动态海量转储**、**动态增量转储**、**静态海量转储**和**静态增量转储**。

7.4.2 登记日志文件

日志文件：

是用来记录事务对数据库的更新操作的文件。

常用的日志文件格式有两种：

以记录为单位的日志文件；

以数据块为单位的日志文件；

以记录为单位的日志文件的内容

对于以记录为单位的日志文件，日志文件中需要登记的内容包括：

- 各个事务的开始标记；
- 各个事务的结束标记；
- 各个事务的所有更新操作。

每个事务的开始标记、结束标记和每个更新操作均作为日志文件中的一个**日志记录**。日志文件中的一个日志记录的内容主要包括：

- 事务标识（标明是哪一个事务）；
- 操作的类型（插入、删除或修改）；
- 操作对象（记录内部标识）；
- 更新前数据的旧值（对插入操作，值为空）；
- 更新后数据的新值（对删除操作，值为空）；

以数据块为单位的日志文件

对于以数据块为单位的日志文件，日志记录的内容包括事务标识和被更新的数据块。由于将更新前的整个块和更新后的整个块都放入日志文件中，操作的类型和操作对象等信息就不必放入日志记录中。

日志文件的作用

日志文件在数据库的恢复中有着非常重大的作用。可以用来进行事务故障恢复和系统故障恢复，并协助后备副本进行介质故障恢复。

日志文件的具体作用是：

- 事务故障恢复和系统故障恢复必须用日志文件。
- 在动态转储方式中必须建立日志文件，后援副本和日志文件综合起来才能有效地恢复数据库。
- 在静态转储方式中，也可以建立日志文件。

当数据库毁坏后可重新装入后援副本把数据库恢复到转储结束时刻的正确状态，然后利用日志文件，把已完成的事务进行重做处理，对故障发生时尚未完成的事务进行撤消处理。这样不必重新运行那些已完成的事务程序就可把数据库恢复到故障前某一时刻的正确状态。

登记日志文件

为保证数据库是可恢复的，登记日志文件时必须遵循两条原则：

- 登记的次序严格按并发事务执行的时间次序；
- 必须先写日志文件，后写数据库。

把对数据的修改写到数据库中和把表示这个修改的日志记录写到日志文件中是两个不同的操作。有可能在这两个操作之间发生故障，即这两个写操作只完成了一个。如果先写了数据库修改，而在运行记录中没有登记这个修改，则以后就无法恢复这个修改了。如果先写日志，但没有修改数据库，按日志文件恢复时只不过是多执行一次不必要的撤消操作，并不会影响数据库的正确性。所以**为了安全，一定要先写日志文件，即首先把日志记录写到日志文件中，然后写数据库的修改**。

7.5.1 事务故障的恢复

事务故障是指事务在运行至正常终止点前被终止，这时恢复子系统应**利用日志文件撤销此事务已对数据库进行的修改**。事务故障的恢复是由系统自动完成的，对于用户来说是透明的。

系统的恢复步骤请同学们参考P_{255}。（略）

7.5.2 系统故障的恢复

系统故障造成数据库不一致的原因有：

一、未完成事务对数据库的更新可能已写入数据库；

二、已提交事务对数据库的更新可能还留在缓冲区没来得及写入数据库。

因此，系统故障的恢复操作就是要撤销故障发生时未完成的事务，重做已完成的事务。

系统故障的恢复是由系统在重新启动时完成的，**不需要用户干预**。

具体的恢复步骤，请同学们参考P255。（略）

7.5.3 介质故障的恢复

发生介质故障后，磁盘上的物理数据和日志文件被破坏，这是最严重的一种故障，**恢复方法是重装数据库，然后重做已完成的事务**。（恢复时，需要有DBA的介入，才能完成数据库的恢复。）

更具体的解释，请同学们参考P_{256}。（略）

7.6 具有检查点的恢复技术

利用日志技术进行数据库恢复时，恢复子系统必须搜索日志，确定哪些事务需要重做，哪些事务需要撤销。一般来说，需要检查所有日志记录。这样做存在两个弊端：

一、搜索整个日志将耗费大量的时间；

二、很多需要重做处理的事务已经将它们的更新操作结果写倒数据库中了，然而恢复子系统又将重新执行这些操作，这浪费了大量时间。

为了最大限度地减少数据库完全恢复时所必须执行的日志部分，SQL-92提出了具有检查点的恢复技术。这种技术将在**日志文件**中增加一类新的纪录——**检查点记录**，增加一个重新开始文件，并让恢复子系统在登录日志文件期间动态地维护日志。

关于检查点的其他内容，我们不再详细介绍，留给同学们自学。

7.7 数据库的镜像

为了避免磁盘介质出现故障影响数据库的可用性，许多数据库管理系统提供了数据库镜像功能用于数据库恢复。即根据DBA的要求，自动把整个数据库或其中的关键数据复制到另一个磁盘上。每当主数据库更新时，DBMS自动把更新后的数据复制过去，即DBMS自动保证镜像数据与主数据的一致性。这样，一旦出现介质故障，可由镜像磁盘继续提供使用，同时DBMS自动利用镜像磁盘数据进行数据库的恢复，不需要关闭系统和冲撞数据库副本。在没有出现故障时，数据库镜像还可以用于并发操作。**实际应用中，往往只选择对关键数据和日志文件镜像，而不是对整个数据库进行镜像。**

7.8 小结

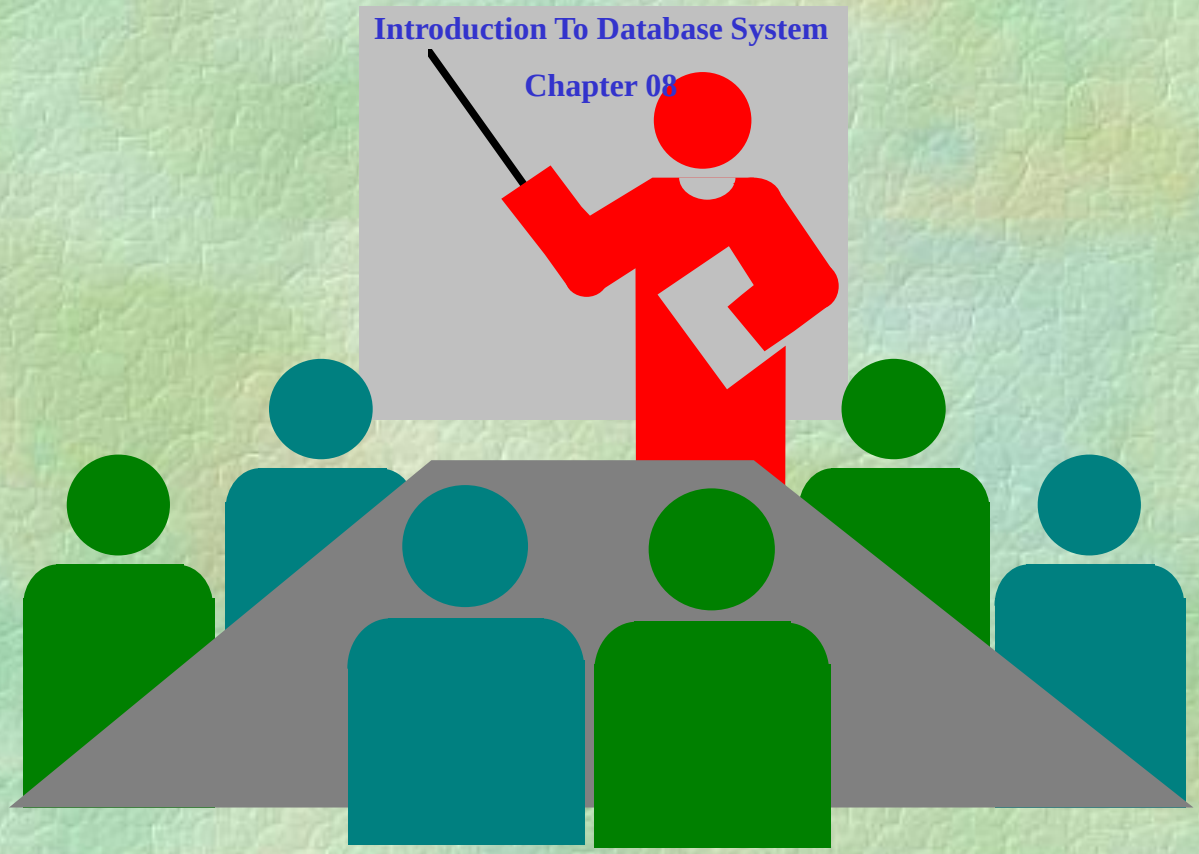
保证数据的一致性是对数据库的最基本要求。事务是数据库的逻辑工作单位，只要DBMS能够保证系统中一切事务的ACID特性，也就保证了数据库处于一致状态。为了保证事务的ACID特性，DBMS必须对事务故障、系统故障和介质故障进行恢复。数据库转储和登记日志文件是恢复中最经常使用的技术。恢复的基本原理就是利用存储在后援副本、日志文件和数据库镜像中的冗余数据来重建数据库。

事务不仅是恢复的基本单位，也是并发控制的基本单位，为了保证事务的ACID特性，DBMS必须对并发控制进行控制。

Chapter 07 is over

Thanks for all!!!

第八章 并发控制


Introduction To Database System
Chapter 08


Introduction to Database System

章节前言

数据库的特点之一就是资源共享，可以供多个用户使用。允许多个用户同时使用的数据库系统称为多用户数据库系统。如：售票系统、银行数据库系统等。

事务可以一个一个地串行执行，即每个时刻只有一个事务运行，其他事物必须等到这个事务结束以后方能运行。然而，如果事务串行执行，则许多系统资源将处于空闲状态。因此，为了充分利用系统资源，发挥数据库共享资源的特点，应该允许多个事务并行地执行。

章节前言（续）

在单处理机系统中，事务的并行执行是这些并行事务的操作轮流交叉执行。这种并行执行方式称为交叉并发方式（Interleaved Concurrency）。尽管没有真正地并行运行，但是减少了处理机的空闲时间，提高了系统的效率。

在多处理机系统中，每个处理机可以运行一个事务，多个处理机可以同时允许多个事务，实现多个事务真正的并行运行。这种并行执行方式称为同时并发方式（Simultaneous Concurrency）。下面，我们将以单处理机系统中的交叉并发事务处理为基础介绍并发控制的相关技术。

当多个用户并发地存取数据库时就会产生多个事务同时存取同一数据的情况。如果对并发操作控制不当，就会造成存取和存储不正确的数据，从而破坏数据库的一致性。所以，DBMS必须提供并发控制机制。它是衡量DBMS性能的重要标志之一。

8.1 并发控制概述

事务是并发控制的基本单位，保证事务的ACID特性是事务处理的重要任务，而事务ACID特性可能遭到破坏的原因之一是多个事务对数据库的并发操作造成的。为了保证事务的隔离性更一般，为了**保证数据库的一致性**，DBMS需要**对并发操作进行正确的调度**。这就是DBMS中数据库并发控制机制的责任。

并发操作实例

考虑飞机订票系统中的一个活动序列：

① 甲售票点（甲事务）读出某航班的机票余额A，设A=16；

② 乙售票点（乙事务）读出同一航班的机票余额A，也有A=16；

③ 甲售票点卖出一张机票，修改余额后A为15，把A写回数据库；

④ 乙售票点也卖出一张机票，修改余额后A为15，把A写回数据库；

显然，数据库中剩下的机票数应该为14，但按照上述活动序列，机票数为15，因此，该活动序列造成了数据库的不一致性。

常见的三类数据不一致性

从并发操作带来的数据不一致性主要可以分为三类，它们分别是：

丢失修改；

不可重复读；

读“脏”数据。

下面，我们将详细介绍这三种数据不一致性。

丢失修改（Lost Update）

丢失修改：

两个事务T_1和T_2 读入同一数据并修改，T_2提交的结果破坏了T_1提交的结果，导致T_1的修改被丢失。如下图所示：

T_1	T_2
（1） 读入A = 16	
（2）	读入A = 16
（3） A= A - 1 写回A = 15	
（4）	A= A - 1 写回A = 15

不可重复读（Non-Repeatable Read）

不可重复读：

事务T_1读取数据后，事务T_2执行更新操作，使T_1无法再现前一次读取结果。包括三种情况：

（1）T1读取某一数据后，T2对其做了修改，当T1再次读取数据时，得到与前一次不同的值。

（2）T_1按照一定条件从数据库中读取了某些数据记录后，T_2删除了其中的部分记录，当T_1再次按照相同条件读取数据时，发现某些记录已经不存在。

（3）T_1按一定条件从数据库中读取某些数据记录后，T_2插入了一些记录，当T_1再次按相同条件读取数据时，发现多了一些记录。

称后面的两种不可重复读为**幻影现象**（Phantom Row）。

不可重复读举例

T_1	T_2
（1） 读入A = 50 读入B = 100 求和 = 150	
（2）	读入B = 100 B = B * 2 写回 B = 200
（3） 读入A = 50 读B = 200 和 = 250	

读“脏”数据（Dirty Read）

读“脏”数据：

事务T_1修改某一数据，并将其写回磁盘，事务T_2读取同一数据后，T_1由于某种原因被撤销，这时T_1已修改过的数据恢复原值，T_2读到的数据就与数据库中的数据不一致，则T_2读到的数据就是“脏”数据，即数据不正确。

T_1	T_2
（1） 读入C = 100 C = C * 2 写回C	
（2）	读入C = 200
（3） RollBack C恢复为100	

对数据不一致性的几点补充说明

产生上面提到的三种数据不一致性的主要原因是：并发操作破坏了事务的隔离性，并发控制就是要用正确的方式调度并发操作，使一个用户事务的执行不受其他事务的干扰，从而避免造成数据的不一致性。

另一方面，对数据库的应用优势允许某些不一致性，这时可以降低对一致性的要求，从而减少系统的开销。

并发控制的主要技术是**封锁**。下面，我们将详细介绍这个概念。

8.2 封锁（Locking）

封锁：

就是事务T在对某个数据对象（如表、记录等）操作之前，先向系统发出请求，对其加锁。加锁后事务T就对该数据对象有了一定的控制，在事务T释放它上面的锁之前，其他的事务不能更新此数据对象。

基本的封锁类型有两种：

排他锁　　和　　共享锁

排它锁（Exclusive Locks）

排它锁，也称为写锁（简称为X锁）。定义如下：

若事务T对数据对象A加上了排它锁，则只允许T读取和修改A，其他任何事务都不能再对A加任何类型的锁，直到T释放A上的锁。**排它锁保证了**其他在事务T释放A上的锁之前不能再读取和修改A。

共享锁（Share Locks）

共享锁又称为读锁（简称S锁），定义如下：

若事务T对数据对象A加上S锁，则事务T可以读取A但是不能修改A，其他事务只能再对A加S锁，而不能加X锁，直到T释放A上的S锁。共享锁保证了其他事务可以读A，但在T释放A上的S锁之前不能对A做任何修改。

排它锁与共享锁的控制方式可以用相容矩阵来描述，请同学们自学参考。

8.3 封锁协议

封锁协议（Locking Protocol）：

在运用X锁和S锁这两种基本封锁对数据对象加锁时，对锁约定的一些基本规则：如何时申请X锁或S锁、持锁时间、何时释放等。

对封锁协议定义不同的规则，就形成了各种不同的封锁协议。

针对并发操作的正确调度可能带来的丢失修改、不可重复读和读“脏”数据等不一致性问题，提出了**三级封锁协议**，它们分别在一定程度上解决了这些问题，为并发操作的正确调度提供了一定的保证。

一级封锁协议

一级封锁协议是：

事务T在修改数据R之前必须先对R加X锁，直到事务T结束才释放。事务结束包括正常结束（Commit）和非正常结束（Rollback）。

一级封锁协议的**特点**：

可防止丢失修改，并保证事务T是可恢复的。

不足：在一级封锁协议中，如果仅仅是读数据而不对其进行修改，是不需要加锁的，因此，它不能保证可重复读和不读“脏”数据。

请参考教材中关于图8.3(a)事务T_1加X锁的例子。

二级封锁协议

二级封锁协议是：

一级封锁协议加上事务T在读取数据R之前必须先对R加S锁，读完后即可释放S锁。

二级封锁协议的**特点**：

除防止了丢失修改，还防止了读“脏”数据。

不足：在二级封锁协议中，由于读完数据后即可释放S锁，所以它不能保证可重复读。

请参考教材中关于图8.3(c)事务加X锁和S锁的例子。

三级封锁协议

三级封锁协议是：

一级封锁协议加上事务T在读取数据R之前必须先对其加S锁，直到事务结束才释放。

三级封锁协议的特点：

除防止了丢失修改和不读“脏”数据外，还进一步防止了不可重复读。

请参考教材中关于图8.3(b)事务加X锁和S锁的例子。

三级封锁协议小结

三级封锁协议的主要区别是：

什么操作需要申请封锁，以及何时释放锁（即持锁时间）。

不同级别的封锁协议

	X锁		S锁		一致性保证		
	操作结束释放	事务结束释放	操作结束释放	事务结束释放	不丢失修改	不读“脏”数据	可重复读
一级封锁协议		√			√		
二级封锁协议		√	√		√	√	
三级封锁协议		√		√	√	√	√

8.4 活锁

活锁：

如果事务T_1封锁了数据R，事务T_2又请求封锁R，于是T_2等待。T_3也请求封锁R，当T_1释放了R上的封锁之后系统首先批准了T_3的请求，于是T_2仍然是等待。然后T_4又请求封锁R，……，T_2有可能永远等待，这就是活锁。

避免活锁的方法：（队列策略）

采用先来先服务的策略。当多个事务请求封锁同一数据对象时，封锁子系统按照请求封锁的先后次序对事务排队，数据对象上的锁一旦释放就批准申请队列中第1个事务获得锁。

死锁

死锁：

如果事务T_1封锁了数据R_1，T_2封锁了数据R_2，然后T_1又请求封锁R_2，因T_2已封锁了R_2，于是T_1等待释放R_2上的锁，接着T_2又申请封锁R_1，因T_1已封锁了R_1，T_2也只能等待T_1释放R_1上的锁。这样就会出现T_1在等待T_2，而T_2又在等待T_1的局面，T_1和T_2两个事务永远不能结束，形成死锁。

数据库中解决死锁问题，主要有两类方法：

一类方法是采取一定措施来**预防死锁**的发生；

另一类方法是**允许发生死锁**，采用一定手段定期**诊断系统中有无死锁**，若有则解除。

死锁的预防

数据库中产生死锁的原因：

两个或多个事务都已封锁了一些数据对象，然后又都请求对已为其他事务封锁的数据对象加锁，从而出现死等待。因此，防止死锁的发生就是要破坏产生死锁的条件。

预防死锁发生，通常有两种方法：

一次封锁法；

顺序封锁法。

一次封锁法

一次封锁法要求每个事务必须一次将所有要使用的数据全部加锁，否则就不能继续执行。

一次封锁法虽然可以有效地防止死锁的发生，但也如下存在问题：

第一、一次就将以后要用到的全部数据加锁，因此扩大了封锁的范围，从而降低了系统的并发度。

第二、数据库中数据是不断变化的，原来不要求封锁的数据，在执行过程中可能会变成封锁对象，所以很难事先精确地确定每个事务所要封锁的数据对象，因此只能扩大封锁范围，将事务在执行过程中可能要封锁的数据对象全部加锁，所以，这将降低系统的并发度。

顺序封锁法

顺序封锁法是预先对数据对象规定一个封锁顺序，所有事务都按这个顺序实行封锁。

顺序封锁法可以有效地防止死锁，但也存在如下问题：

第一、数据库系统中封锁的数据对象很多，并且随数据的插入、删除等操作而不断变化，要维护这些资源的封锁顺序非常困难，成本很高。

第二、事务的封锁请求可以随着事务的执行而动态地决定，很难实现确定每一个事务要封锁哪些对象，因此也很难按规定的顺序去施加封锁。

由以上可知，在数据库系统中预防死锁的策略不是很适合，为此，DBMS通常提供诊断并解除死锁的方法来避免死锁的发生。

死锁的两种诊断方法

数据库系统中**诊断死锁的方法**一般有：**超时法**和**事务等待图法**。

一、超时法：

如果一个事务的等待时间超过了规定的时限，就认为发生了死锁。

缺点：（1）可能误判死锁；（2）时限若设置太长，死锁发生后不能及时发现。

二、事务等待图法：

事务等待图是一个有向图G=（T，U），其中T为节点的集合，每个节点表示正运行的事务；U表示边的集合，每条边表示事务等待的情况。若T_1等待T_2，则从T_1到T_2有一条有向边。事务等待图动态地反映了所有事务的等待情况。并发控制子系统周期性地检测事务等待图，如果发现存在回路，则表示系统已经发生了死锁。

死锁的解除方法

DBMS的并发控制子系统一旦检测到系统中存在死锁，就要设法解除。通常采用的方法是选择一个处理死锁代价最小的事务，将其撤销，释放此事务持有的所有的锁，使其他事务得以继续运行下去。当然，对撤销的事务所执行的数据修改操作必须加以恢复。

SQL Server2000中的死锁检测

检测和结束死锁

在 Microsoft® SQL Server™ 2000 中，单个用户会话可能有一个或多个代表它运行的线程。每个线程可能获取或等待获取各种资源，如：

- 锁。
- 与并行查询执行相关的资源（与交换端口相关联的处理协调器、发生器和使用者线程）。
- 线程。
- 内存。

上述这些资源除内存外都参与 SQL Server 检测方案。对于内存，SQL Server 使用基于超时的机制来检测。

SQL Server2000中的死锁检测-续

在 SQL Server 2000 中，死锁检测由一个称为**锁监视器线程**的单独的线程执行。在出现下列任一情况时，锁监视器线程对特定线程启动死锁搜索：

- **线程已经为同一资源等待了一段指定的时间。**

锁监视器线程定期醒来并识别所有等待某个资源的线程。如果锁监视器再次醒来时这些线程仍在等待同一资源，则它将对等待线程启动锁搜索。

- **线程等待资源并启动急切的死锁搜索。**

注意：SQL Server 通常只执行定期死锁检测，而不使用急切模式。因为系统中遇到的死锁数通常很少，**定期死锁检测有助于减少系统中死锁检测的开销。**

SQL Server2000中的死锁检测-续

当锁监视器对特定线程启动死锁检测时，它识别线程正在等待的资源。然后，锁监视器查找特定资源的拥有者，并递归地继续执行对那些线程的死锁搜索，直到找到一个循环。**用这种方式识别的循环形成一个死锁**。

在识别死锁后，系统自动选择可以打破死锁的线程（死锁牺牲品）来结束死锁。**SQL Server 回滚作为死锁牺牲品的事务**，通知线程的应用程序取消线程的当前请求，然后允许不间断线程的事务继续进行。

SQL Server 通常**选择运行撤消时花费最少的事务的线程作为死锁牺牲品**。另外，用户可以使用 SET 语句将会话的 DEADLOCK_PRIORITY 设置为 LOW。DEADLOCK_PRIORITY 选项控制在死锁情况下如何衡量会话的重要性。如果会话的设置为 LOW，则当会话陷入死锁情况时将成为首选牺牲品。

8.5 并发调度的可串行性

计算机系统对并发事务中并发操作的调度是随机的，而不同的调度可能会产生不同的结果，那么哪个结果是正确的，哪个是错误的呢？

如果一个事务在运行过程中没有其他事务同时运行，也就是说它没有受到其他事务的干扰，那么就可以认为该事务的运行结果是正常的或可预想的。因此**将所有事务串行起来的调度策略一定是正确的调度策略**。虽然以不同的顺序串行执行事务可能会产生不同的结果，但由于不会将数据库置于不一致状态，所以就运行本身和数据的一致性而言都是正确的。

可串行化调度

DEF： 多个事务的并发执行是正确的，当且仅当其结果与按照某一次序串行地执行它们时的结果相同，那么我们就称这种调度策略是**可串行化地调度**。

可串行性是并发事务正确性的准则。

一个给定的并发调度当且仅当它是可串行化的，才认为是正确调度。

可串行化的例子

关于可串行化的并发事务调度的实例，请同学们参考教材P_{272}—P_{273}。（留给同学们自学）

结论：判断某个并发事务的调度策略是否是正确的，可以将该并发调度的结果与某一个串行化执行的结果相比较，如果相同，则该并发调度策略是正确的。

DBMS保证调度正确的策略

为了保证并发操作的正确性，DBMS的并发控制机制必须提供一定的手段来保证调度是可串行化的。

从理论上讲，在某一事务执行时禁止其他事务执行的调度策略一定是可串行化的调度，这是*最简单的调度策略*。但实际上是不可取的。因为用户不能充分的共享数据库资源。所以，DBMS必须能够提供某种策略来实现并发操作调度的可串行性，从而保证调度的正确性。

两段锁协议就是保证并发调度可串行化的封锁协议。当然，还有其他一些方法，如时标方法等。

8.6 两段锁协议

两段锁协议指的是：**所有事务必须分两个阶段对数据项加锁和解锁**。这两个阶段分别是：

（1）在对任何数据进行读、写操作之前，首先要申请并获得对该数据的封锁；

（2）在释放一个封锁之后，事务不再申请和获得任何其他封锁。

对两段锁协议的理解

“两段”锁是指：事务在锁的处理上分为两个阶段：

第一阶段是**获得封锁**，也称为扩展阶段。在这阶段，事务可以申请获得任何数据项上的任何类型的锁，但是不能释放任何锁。

第二阶段是**释放封锁**，也称为收缩阶段。在这阶段，事务可以释放任何数据项上的任何类型的锁，但是不能再申请任何锁。

若并发执行的所有事务均遵守两段锁协议，则对这些事务的任何并发调度策略都是可串行化的。（证明部分，留给同学们思考。）

可串行调度与两段锁协议的联系

事务遵守两段锁协议是可串行化调度的充分条件，而不是充要条件。也就是说，若并发事务都遵守两段锁协以，则对这些事务的任何并发调度策略都是可串行化的；若对并发事务的一个调度是可串行化的，不一定所有事务都符合两段锁协议。

关于两段锁协议与可串行化调度的关系的实例，同学们可以参考教材P_{275}的图8.6。（留做自学）

两段锁协议与一次封锁法的异同

一次封锁法要求每个事务必须一次将所有要使用的数据全部加锁，否则就不能继续执行，因此，**一次封锁法是遵守两段锁协议的**；

但是两段锁协议并不要求事务必须一次将所有要使用的数据全部加锁，因此**遵守两段锁协议的事务仍然可能发生死锁**。（请参考教材P275的图8.7。）

8.7 封锁的粒度

封锁对象的大小称为**封锁粒度**。***封锁对象可以是逻辑单元，也可以是物理单元。***以关系数据库为例，封锁对象可以是这样一些逻辑单元：属性值、属性值的集合、元组、关系、索引项、整个索引甚至整个数据库，也可以是这样一些物理单元：页（数据页或索引页）、块等。

封锁粒度与系统的并发度和并发控制的开销密切相关。直观地看，封锁的粒度越大，数据库所能够封锁的数据单元就越少，并发度就越小，系统开销也越小；反之，封锁的粒度越小，并发度较高，但系统开销也就越大。（请参考教材P_{276}中关于封锁粒度与系统的并发度和并发控制的关系的实例）

多粒度封锁

由上面的讨论可知，如果在一个系统中同时支持多种封锁粒度供不同的事务选择是比较理想的，这种封锁方法称为**多粒度封锁**。选择封锁粒度时应该同时考虑封锁开销和并发度这两个因素，适当选择封锁粒度以求得最优的效果。

一般的选择原则如下：需要处理大量元组的事务可选择关系为封锁粒度；需要处理多个关系的大量元组的事务可选择数据库为封锁粒度；而对于一个处理少量元组的用户事务，应该选择以元组为封锁粒度比较合适。

多粒度树、多粒度封锁协议、意向锁

关于多粒度树、多粒度封锁协议及意向锁等内容，留给同学们自学，不作为必修内容。

8.8 本章小结

并发控制主要涉及保证各个事务本身正确的问题。

所谓事务就是对数据库等操作或运算的一组命令。

数据库系统是多用户共享的系统，对数据库的存取可能是并行的，所以，即使单个事务执行时所有的事务都是正确的，在并发的情况下，因为事务之间的互相干扰，仍然可能使最后的总的结果不正确。

并发控制就是以正确的方式调度并发事务。使一个事务的执行不受其他事务的干扰。在数据库环境下，并发控制的主要方式是封锁机制。即加锁。基本的锁类型有两类：排它锁和共享锁。

本章小结（续）

排它锁：

如果事务T对某个目标建立了排它锁，则不再允许任何事务对这个目标再加任何类型的锁，直至T释放其锁为止。

共享锁：

如果事务T对某个目标建立了共享锁，则另一个不同的事务T也可以对这个目标取得共享锁，但不能取得排它锁，直到对它的所有共事锁部释放为止。

本章小结（续）

死锁：

死锁发生时，其中两个或多个事务同时处于等待状态，其中的每一个在它能够进行之前都等待另一个释放锁。

在数据库中解决死锁的常用方法有：

（1）要求每个事务一次就将所有要使用的数据全部加锁，否则就不能执行。

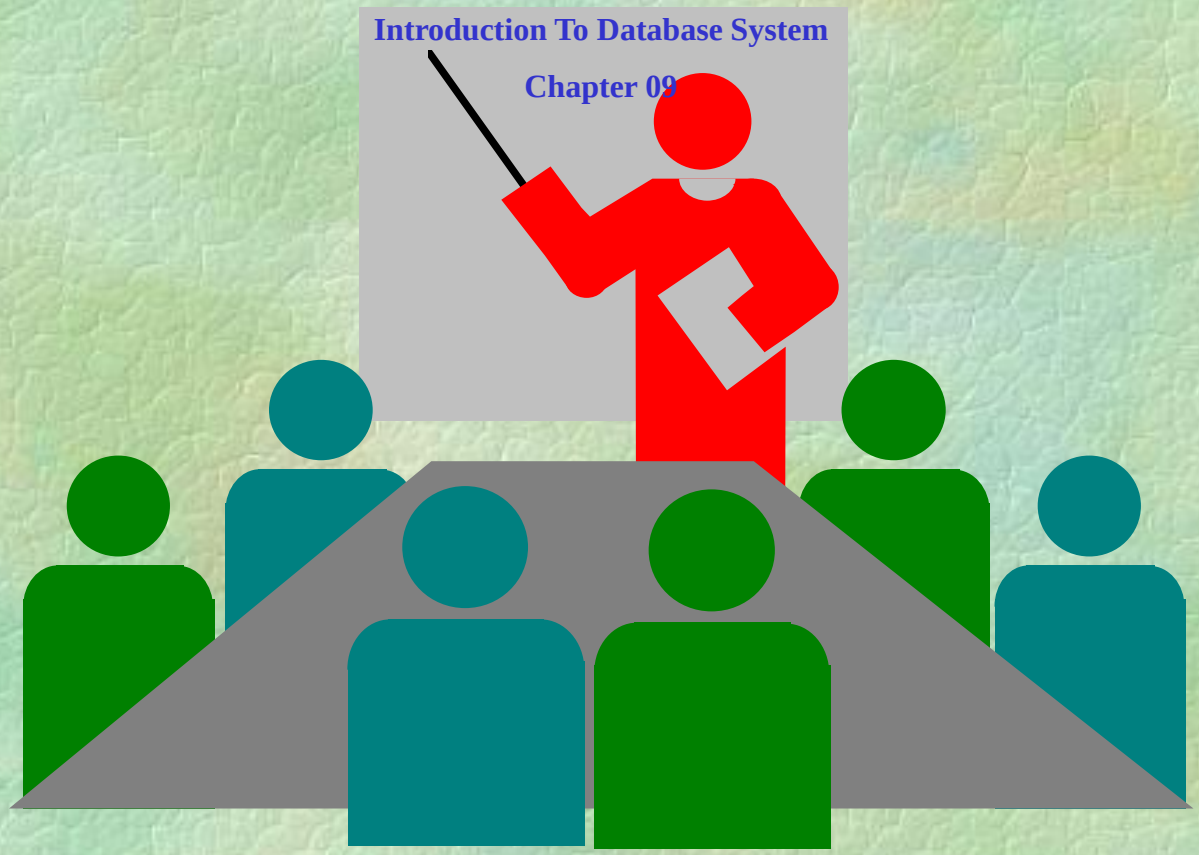
（2）预先规定一个封锁顺序，所有的事务部必须按这个顺序对数据执行封锁。

（3）不采取任何措施来预防死锁的发生，而是采用某种方法诊断系统中是否有死锁。

Chapter 08 is over

Thanks for all!!!

第九章 数据库安全性


Introduction To Database System
Chapter 09


Introduction to Database System

9.1 计算机安全性概论

数据库的安全性：

指保护数据库以防止不合法的使用所造成的数据泄漏、更改或破坏。

由于数据库系统中大量数据集中存放，而且许多最终用户可以直接共享这些数据，因此，在数据库系统中安全性问题是最为突出的。系统安全性保护措施是否有效是衡量数据库系统的主要指标之一。

9.1.1 计算机系统的三类安全性问题

计算机系统安全性：

是指为计算机系统建立和采取的各种安全保护措施，以保护计算机系统中的硬件、软件及数据，防止其因为偶然或恶意的原因使系统遭到破坏，数据遭到更改或删除等。

概括起来，计算机系统的安全性问题可以分为三大类：

技术安全类；

管理安全类；

政策法律类。

三类计算机安全问题

技术安全：

计算机系统中采用具有一定安全性的硬件、软件来实现对计算机系统及其所存数据的安全保护，当计算机系统受到无疑或恶意的攻击时仍能保证系统的正常运行，保证系统内的数据不增加、不丢失、不泄漏。

管理安全：

指软硬件故障、场地的意外事故、管理不善导致的计算机设备和数据介质的物理破坏、丢失等安全问题。

政策法律类安全：

指政府部门建立的有关计算机犯罪、数据安全保密的法律道德准则和政策法规、法令等。

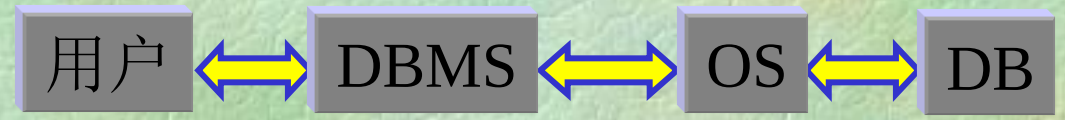
9.1.2 可信计算机系统评测标准

随着计算机资源共享和网络技术的应用日益广泛和深入，特别是Internet技术的发展，计算机安全性问题越来越得到人们的重视。对各种计算机及其相关产品、信息系统的安全性要求越来越高。为此，在计算机安全技术方面逐步发展并建立了一套可信计算机系统的概念和标准，以此规范和指导安全计算机系统部件的生产，测定产品的安全性能指标等。

关于可信计算机系统评测标准，留给同学们自学。

9.2 数据库安全性控制

计算机系统的安全模型：



用户标签和鉴别	存取控制	操作系统安全保护	数据密码存储

对上述安全模型的解释：

用户要求进入计算机系统时，系统首先根据输入的用户表示进行用户身份鉴定，只有合法的用户才可进入计算机系统。对已进入系统的用户，DBMS还要进行存取控制，只允许用户执行合法的操作。操作系统也将提供自己的保护措施。数据最后还可以以密码形式存储到数据库中。

9.2.1 用户标签与识别

用户表识和鉴别是系统提供的最外层安全保护措施。即：由系统提供一定的方式让用户标识自己的名字或身份。每次用户要求进入系统时，由系统进行核对，通过鉴定后才能提供及其使用权。

对于获上机权的用户如果使用数据库，DBMS还要进行用户标识和鉴定。常用的方法是：

用户名 + 用户密码

一、用户名：用一个用户名或者用户标识号来表明用户身份。系统内部记录着所有合法用户的标识，系统鉴别此用户是否是合法用户，如果是，则可以进入下一步的核实，否则，则不能使用系统。

二、密码：为了进一步核实用户，系统常常需要输入密码，系统将核对密码以鉴别用户的身份。

9.2.2 存取控制

数据库安全最重要的一点就是**必须确保只授权给有资格的用户访问数据库的权限，同时令所有未被授权的人员无法获取数据**，这就是数据库系统的存取控制机制需要实现的内容。

存取控制机制主要包括两个部分：

1、**定义用户权限**（指不同的用户对不同的数据对象允许执行的操作权限），将用户权限登记到数据字典中。

2、**合法权限检查**。每当用户发出存取数据库的操作请求后，DBMS查找数据字典，根据安全规则进行合法权限检查。

9.2.3 、 9.2.4 （略）

关于自主存取控制和强制存取控制，留给同学们自学。

9.2.5 视图机制

进行存取权限控制时可以为不同的用户定义不同的视图，把数据对象限制在一定的范围内，也就是说，通过视图机制把要保密的数据对无权存取的用户隐藏起来，从而自动地对数据提供一定程度的安全保护。

有关视图在数据库安全性控制方面的作用，我们在讲解视图的相关概念时，已做介绍，在此不再赘述。

9.2.6 审核（Audit）

DBMS为了达到一定的安全级别，通常还需要在其他方面提供相应的支持。审核功能就是其中的一种。

审核：

把用户对数据库的所有操作自动记录下来并存储到审核日志中，DBA可以利用审核日志中的信息，重现导致数据库现有状况的一系列事件，从而可以找出非法存取数据的人、事件和内容等。

SQL Server 2000支持审核机制。限定审核日志的最大大小为200M。系统使用文件翻转可以避免审核跟踪由于审核日志已填满而失败。如果审核登记失败，则系统将在Windows事件日志和 SQL Server 错误日志中生成一项记录。

9.2.7 数据加密

对于高度敏感的数据，除了采用上述安全性措施外，还可以采用数据加密技术。

数据加密技术是防止数据库中数据在存储和传输中失密的有效手段。通常的思想是：根据一定的算法将原始数据变换为不可直接识别的格式，从而使得不知道解密算法的人无法获知数据的内容。

加密方法主要有两种：：

替换 和 **置换**

DES数据加密标准（美国）中同时综合了替换和置换两种方法。

9.3 、9.4 （略）

9.3 统计数据库安全性

与

9.4 Oracle的数据库安全性措施

不做要求，留给同学们自学。

SQL Server2000系统提供的安全性控制手段较多，由于学时原因，我们不再介绍。

Chapter 09 is over

Any

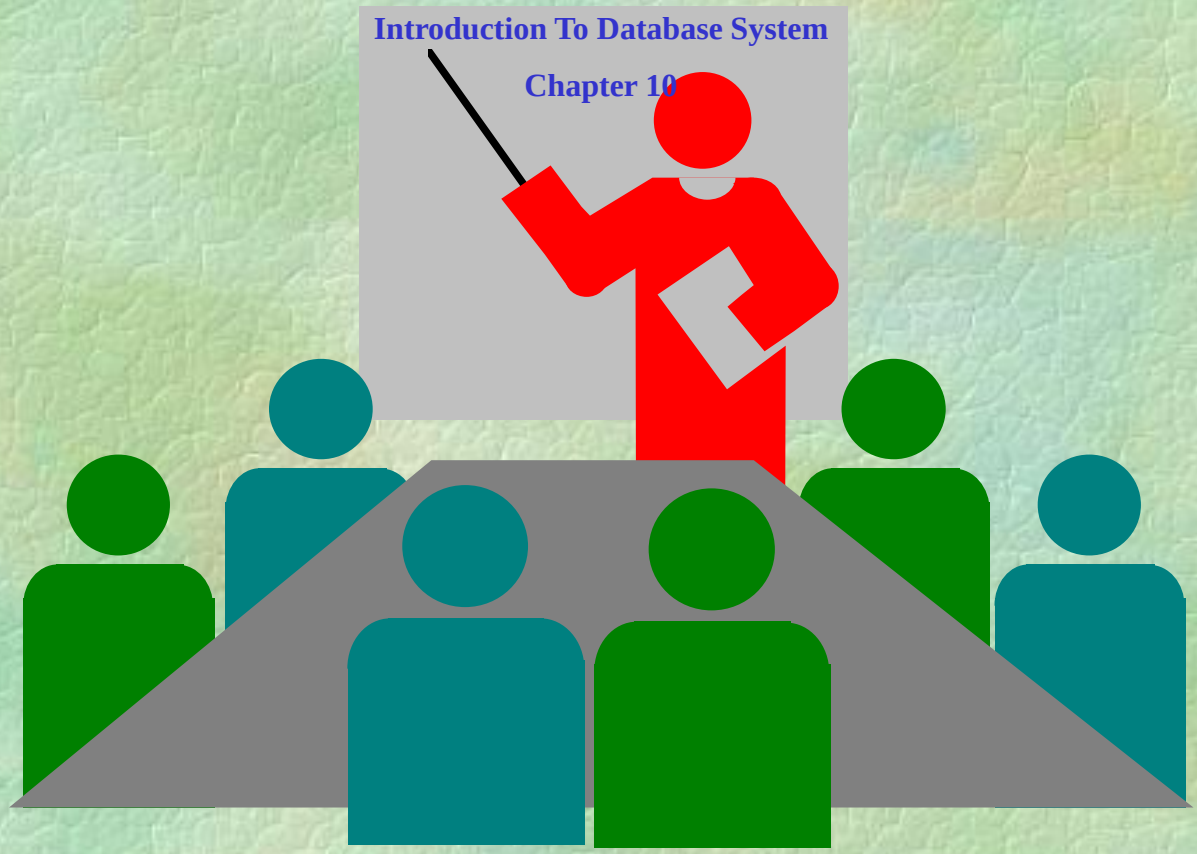
Question

？ ？ ？

Chapter 09 is over ！ ！

Thanks for my all friends ！ ！

第十章 数据库完整性


Introduction To Database System
Chapter 10


Introduction to Database System

前言

数据库的完整性是指数据的正确性和相容性。数据库是否具备完整性关系到数据库系统能否真实地反映现实世界，因此维护数据库的完整性非常重要。

完整性与安全性的区别：

完整性是为了防止数据库中存在不符合语义的数据，防止错误信息的输入和输出。针对不合语义的数据而言。

安全性是保护数据库，防止恶意的破坏和非法的存取。针对非法用户和非法操作而言的。

数据库完整性约束条件：

DBMS为了维护数据库的完整性而提供的用来检查数据库中的数据是否满足语义规定的条件的一种完整性约束机制。

10.1 完整性约束条件

完整性约束条件作用的对象可以是关系、元组、列三种。

列约束：

指列的类型、取值范围、精度、排序等约束条件。

元组约束：

指元组中各个字段间的联系的约束。

关系的约束：

指若干元组间、关系集合上以及关系之间的联系的约束。

静态约束和动态约束

完整性约束条件涉及的三种对象，其状态可以是静态的，也可以是动态的。

静态约束：

指数据库每一确定状态时的数据对象所应满足的约束条件，它是反映数据库状态合理性的约束。这是数据库中最重要的一种完整性约束。

动态约束：

指数据库从一种状态变为另一种状态时，新、旧值之间所应该满足的约束，它是反映数据库状态变迁的约束。

静态的列级、元组、关系约束

静态的列级约束是对一个列的取值域的说明，包括以下几个方面：

对数据类型的约束、对数据格式的约束、对取值范围或取值集合的约束、对空值的约束及其他约束等。

静态元组约束是规定元组的各个列之间的约束关系。

静态关系约束反映了在一个关系的各个元组之间或者若干关系之间存在的各种联系或约束等。常见的静态关系约束有：

实体完整性约束、参照完整性约束、函数依赖约束、统计约束。

动态的列级、元组、关系约束

动态列级约束是修改列定义或列值时应满足的约束条件。主要包括：

修改列定义时的约束、修改列值时的约束。

动态元组约束是指修改元组的值时元组中各个字段间需要满足的某种约束条件。

动态关系约束是加在关系变化前后状态上的限制条件，如事务一致性、原子性等约束条件。

10.2 完整性控制

DBMS的完整性控制机制应具有三个方面的功能：

- 定义功能：提供定义完整性约束条件的机制；
- 检查功能：检查用户发出的操作请求是否违背了完整性约束条件。
- 如果发现用户的操作请求使数据违背了完整性约束条件，则采取一定的动作来保证数据的完整性。

一个完善的完整性控制机制应该允许用户定义所有这六类完整性约束条件。

在关系系统中，最重要的完整性约束是实体完整性和参照完整性，其他完整性约束条件则可以归入用户定义的完整性。

下面，我们讨论实现参照完整性要考虑的几个问题。

实现参照完整性需要考虑的几个问题

1、外码能否接受空值？

2、在被参照关系中删除元组的问题；

一般地，当删除被参照关系的某个元组，而参照关系存在若干元组，其外码与被参照关系删除元组的主码值相同，这时有三种不同的策略：级联删除、受限删除、置空值删除。可以根据应用环境确定不同的策略。

3、在参照关系中插入元组时的问题；

一般地，当参照关系插入某个元组，而被参照关系不存在相应的元组，其主码值与参照关系插入元组的外码值相同，这时可以有以下策略：受限插入、递归插入。

4、修改关系中主码的问题；

这种情况，通常包括：不允许修改主码和允许修改主码的情况。

10.3 Oracle的完整性

该部分留给同学们自学。

10.4 小结

略

Chapter 10 is over

Thanks for all!!!

第十一章　数据库管理系统

这一章的内容，

留给同学们自学。

Course is over

《数据库系统概论》课程中的重点内容，我们已经在前续章节中讲过，至此，该门课程基本结束，我们将在课程的最后，利用***Visual C++***和***SQL Server2000***，开发一个简单的实例，并向同学们逐一演示，希望能够对同学们有所帮助。

请参考**数据库系统概论——实例篇**