数据库原理

- 宋安平
- 上海大学计算机学院
- Apsong@shu.edu.cn
- 计算机学院1015室

- 第5-8周

教学内容：

- 事务的定义，事务的**ACID**性质，事务的状态变迁图。
- 恢复的定义、基本原则和实现方法,故障的类型,检查点技术。
- 并发操作带来的三个问题，**X**锁、**PX**协议…,活锁、死锁，并发调度、串行调度、并发调度的可串行化，两段封锁法。
- 完整性的定义，完整性子系统的功能，完整性规则的组成；**SQL**中的三大类完整性约束，**SQL3**中的触发器技术。
- 安全性的定义、级别，权限，**SQL**中的安全性机制。

第8章 数据库的管理

- 事务的概念
- 数据库的恢复
- 数据库的并发控制
- 数据库的完整性
- 数据库的安全性

第一节 事务的概念

- 事务的定义
- 事务的性质

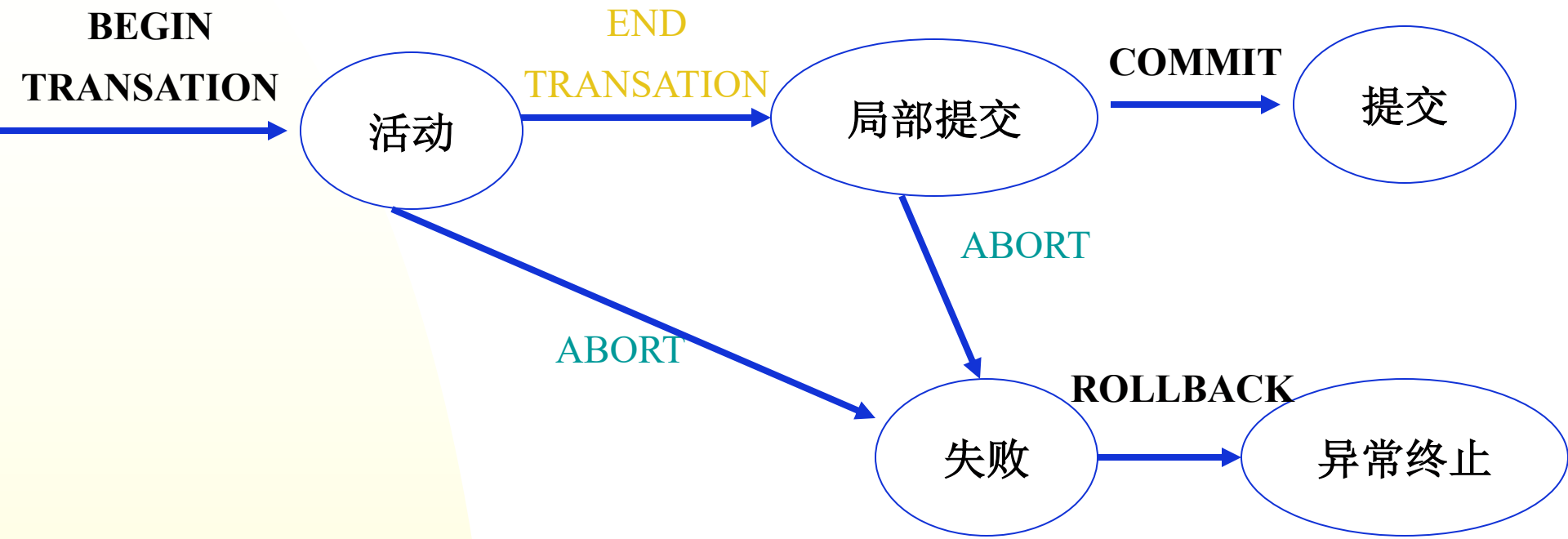
一、事务的定义

- 事务的概念
  - 事务是由若干数据库操作组成的一个逻辑工作单位，是一个不可分割的工作单位。
  - 一个应用程序可以包括多个事务。
  - 事务以BEGIN TRANSACTION语句的成功执行开始，以COMMIT或ROLLBACK语句的成功执行结束。
  - COMMIT（提交）语句表示一事务的全部操作都已成功，它对DB的所有更新可真正写到DB中。
  - ROLLBACK（回退）语句表示事务没有成功地完成全部操作，系统将撤消该事务对DB已作的更新。

二、事务的性质

- 事务有四个重要性质：原子性、一致性、隔离性和持久性。通常称为“ACID性质”。
  - 原子性（atomicity）：事务对数据库的更新要么全部起作用，要么完全不起作用。 DBMS的事务管理子系统
  - 一致性（consistency）：事务将数据库从一个一致性状态转变为另一个一致性状态。 程序员（正确地编写事务）DBMS的完整性子系统
  - 隔离性（isolation ）：事务相互隔离，在多个事务并发执行时，任一事务的更新操作在该事务成功提交前对其他事务都是不可见的。 DBMS的并发控制子系统
  - 持久性（durability）：事务一旦成功提交，其对数据库的更新就永久有效，不会因系统崩溃等而丢失。 DBMS 的恢复管理子系统

三、事务的状态变迁


BEGIN
TRANSATION
活动
END
TRANSATION
局部提交
COMMIT
提交
ABORT
ABORT
失败
ROLLBACK
异常终止

第二节　数据库的恢复

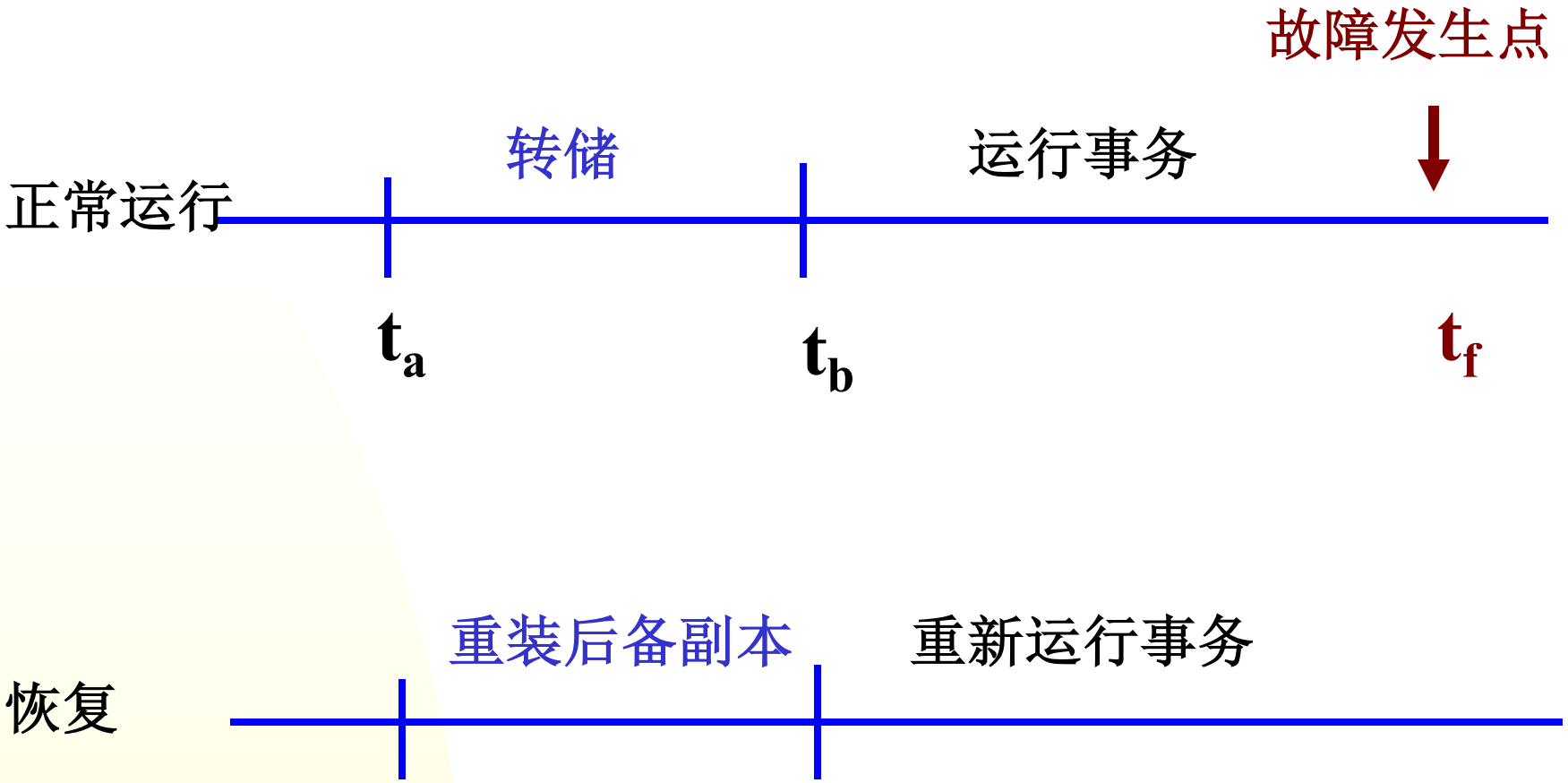
- 数据库恢复的定义原则和方法
- 故障类型
- 恢复方法
- 检查点机制
- 运行记录优先原则

一、数据库恢复的定义原则和方法

- 数据库恢复的定义
  - 系统能把数据库从被破坏、不正确的状态、恢复到最近一个正确的状态，DBMS的这种能力称为数据库的可恢复性。

一、数据库恢复的定义原则和方法

- 数据库恢复的基本策略和实现方法
  - 恢复的基本策略：冗余（数据重复存储）
  - 实现方法：
    - 备份　定期对数据库复制或转储（静态转储和动态转储，海量转储和增量转储）。
    - 日志　执行事务时，记录其开始、结束和对DB的每次更新操作。


故障发生点
转储
运行事务
正常运行
t_a
t_b
t_f
重装后备副本
重新运行事务
恢复

二、故障类型

- 常见的故障有：
  - 事务故障
    - 非预期事务故障：运算错误、数据错误、死锁等，故障事务夭折
    - 可预期事务故障：在事务中可预期出错的地方所加的ROLLBACK语句
  - 系统故障：硬件、系统软件出错，停电等，事务执行被打断，内存中数据被破坏
  - 介质故障：磁头、磁盘控制器或磁盘损坏，盘上数据丢失，病毒破坏等，DB遭破坏

三、恢复方法

当系统运行过程中发生故障，利用数据库后备副本和日志文件将数据库恢复到故障前的某个一致性状态。不同故障其恢复技术不一样：

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中止，此时恢复子系统应撤销（UNDO）此事务已对数据库进行的修改。

事务故障恢复的具体做法如下：

① 反向扫描日志文件（即从最后向前扫描日志文件），

查找该事务的更新操作。

② 对该事务的更新操作执行逆操作。

即将日志记录中“更新前的值”写入数据库：

若记录中是插入操作，则相当于做删除操作

若记录中是删除操作,则做插入操作；

若是修改操作，则用修改前值代替修改后值。

③ 继续反向扫描日志文件，查找该事务的其他更新操作，并做同样处理。

④ 如此处理下去，直至读到此事务的开始标记，事务故障恢复就完成了。

事务故障的恢复是由系统自动完成的，不需要用户干预。

2. 系统故障的恢复

系统故障造成数据库不一致状态的原因有两个：

- 未完成事务对数据库的更新已写数据库；
- 已提交事务对数据库的更新还留在缓冲区没来得及真正写入数据库。

恢复操作：撤销故障发生时未完成的事务，

重做已完成的事务。

具体做法如下：

① 正向扫描日志文件(即从头开始扫描日志文件)，找出在故障发生前：

已提交事务(既有<Ti,START>记录，也有<Ti，COMMIT>记录),将其事务标识记入重做队列。

尚未完成的事务(有<Ti,START>记录，无<Ti,COMMIT>记录),将其事务标识记入撤销队列。

② 对撤销队列中的各个事务进行撤销（UNDO）处理

进行撤销（UNDO）处理的方法是：

反向扫描日志文件，

对每个UNDO事务的更新操作执行逆操作.

即将日志记录中“更新前的值”写入数据库。

③ 对重做队列中的各个事务进行重做（REDO）处理

进行重做REDO处理的方法是：

正向扫描日志文件，

对每个REDO事务重新执行登记操作。

即将日志记录中“更新后的值”写入数据库。

系统故障的恢复也由系统自动完成的,不需要用户干预。

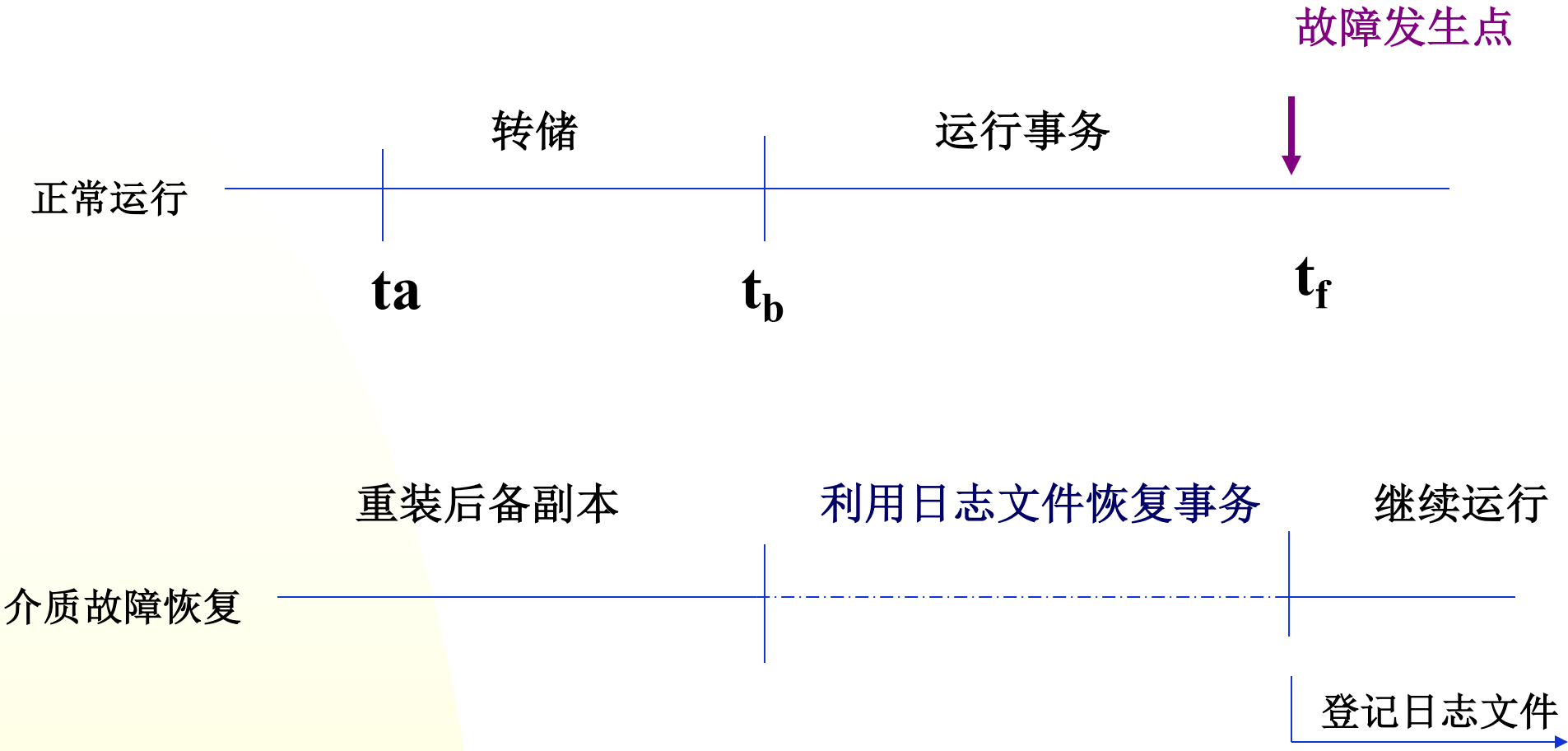
3．介质故障的恢复

在发生介质故障和遭受病毒破坏时，磁盘上的物理数据库遭到毁灭性破坏。此时恢复的过程如下：

① 装入最新的后备副本到新的磁盘，使数据库恢复到最近一次转储时的一致状态。

② 装入有关的日志文件副本，重做已提交的所有事务。

这样就可以将数据库恢复到故障前某一时刻的一致状态。


故障发生点
转储
运行事务
正常运行
ta
t_b
t_f
重装后备副本
利用日志文件恢复事务
继续运行
介质故障恢复
登记日志文件

四、检查点机制

- 检查点（Checkpoint）方法
  - **DBMS定时设置检查点，在检查点时刻才真正做到把对DB的修改写到磁盘，并在日志文件写入一条检查点记录。**
- 检查点方法的恢复方法（二步）
  - **根据日志文件建立重做队列和事务撤消队列。**
  - **对重做队列中的事务进行REDO处理，对撤消队列中的事务进行UNDO处理。**

检查点方法

DBMS定时设置检查点，在检查点时，做下列事情：

第一步：　将日志缓冲区中的日志记录写入磁盘。

第二步：　将数据库缓冲区中修改过的缓冲块内容写入磁盘。

第三步：　写一个检查点记录到磁盘，内容包括：

① 检查点时刻，所有活动事务；

② 每个事务最近日志记录地址。

第四步：把磁盘中日志检测点记录的地址写入“重新启动文件中”。

2. 检查点恢复步骤

① 正向扫描日志文件，建立事务重做队列和事务撤消队列。

将已完成的事务加入重做队列；

未完成的事务加入撤销队列。

② 对撤销队列做UNDO处理的方法是：

反向扫描日志文件，根据撤销队列的记录对每一个撤销事务的更新操作执行逆操作，使其恢复到原状态。

③ 对重做队列做REDO处理的方法是：

正向扫描日志文件，根据重做队列的记录对每一个重做事务实施对数据库的更新操作。

五、运行记录优先原则

- 运行记录优先原则包括两点：
  - 将一个更新结果写到DB中前，必须确保先在日志中成功登记了这个更新。
  - 直至对一事务的日志登记全部完成，方能够允许该事务完成**COMMIT**处理。
- 这个原则确保了发生故障后能够根据日志对事务进行REDO或UNDO。

第三节 数据库的并发控制

- 并发操作带来三个问题
- 封锁机制
- 活锁和死锁
- 并发事务的可串行化调度
- SQL中事务的存取模式和隔离级别

一、并发操作带来三个问题

- 为了充分利用数据库这个共享资源，DBMS允许多个事务并发地存取数据库。
- 如果不对并发操作进行恰当的控制，可能导致如下的数据不一致性问题：
  - 丢失更新问题：一事务所作的更新操作因另一事务的操作而丢失。
  - 读“脏数据”问题：又称为未提交依赖，指一事务取用了别的事务未提交随后又被撤消的数据。
  - 不一致分析问题：指事务从数据库中读取了处于不一致状态的数据，并因此进行了不一致的分析。

1. 丢失更新（Lost update）

指事务Ti与事务Tj从数据库中读入同一数据并修改,事务Tj的提交结果破坏了事务Ti 提交的结果,导致事务1的修改被丢失。

时间	事务Ti	数据库中A的值	事务Tj
t_0		18	
t_1	检索A： A＝18		
t_2			检索A： A＝18
t_3	修改A： A←A-1		
t_4	写回A： A=17		
t_5		17	
t_6			修改A：A←A-2
t_7			写回A： A=16
t_8		16	

2.不一致分析（不可重复读nonrepeatable read ）

指事务Ti读取数据后，事务Tj执行更新操作，使事务Ti无法再读取前一次结果。

时间	事务Ti	数据库中A、B的值	事务Tj
t_0		50、100	
t_1	检索A、B：A＝50，B=100		
t_2	求和		
t_3			检索B：B=100
t_4			修改B：B←B*2
t_5			
t_6		50、200	
t_7	检索(验算):A＝50,B=200		
t_8	求和：A+B=250		

3.读“脏”数据（diriy read）

指:事务Ti修改某一数据，并将其写回磁盘，事务Tj读取同一数据后，事务Ti由于某种原因被撤销，这时事务Ti已修改过的数据恢复原值，事务Tj读到的数据就与数据库中的数据不一致，是不正确的数据，称为“脏”数据。

时间	事务Ti	数据库中C的值	事务Tj
t_0		100	
t_1	检索C：C=100		
t_2	修改C ：C←C*2		
t_3	写回C:: C=200		
T_4		200	
t_5			检索C： C=200
t_6	回滚：ROLLBACK		
t_7	C恢复为：100		
t_8		100	

二、封锁机制

- 封锁是实现并发控制的重要技术。所谓封锁，是指事务向系统发出对某数据对象加锁的请求，以取得对该对象一定的控制权。基本的封锁有两类：
  - 排它型封锁（写锁，X封锁）：一个事务对数据对象加了X锁后，在它释放X锁之前不允许其他事务再对该数据对象加任何锁。
  - 共享型锁（读锁，S封锁）：一个事务对数据对象加了S锁后，允许其他事务再对该数据对象加S锁，但在它释放S锁之前不允许其他事务加X锁。

封锁协议

- 排它型封锁
  - **PX**协议主要内容是：事务若要更新数据，则必须先提出对此数据对象的**X**封锁请求；事务如果未获准**X**封锁，那么进入等待状态，直至获准在此数据对象上的**X**封锁，才能继续执行。**PX**协议可以解决因多事务同时更新同一数据对象而引起的更新丢失问题。
  - PXC协议　是在“PX协议”上再加一规定：解除X锁的操作合并到事务的结束（COMMIT或ROLLBACK）操作中。PXC协议可以解决因事务ROLLBACK而引起的更新丢失问题。

等事务T1更新完成后再执行事务T2：（可解决丢失更新）

时间	事务T1	数据库中A的值	事务T2
t_0		18	
t_1	加锁：XFIND A	T1 ♀加锁	
t_2			XFIND A（失败）
t_3	更新： A:=A-1		等待
T_4	写回A=17：UPD A		等待
t_5		17	等待
t_6	COMMIT(包括解锁)	T1 ♂解锁	等待
t_7		T2 ♀加锁	XFIND A（重做）
t_8			更新：A:=A-2
t_9			写回A=15：UPD A
T_1		15	COMMIT(包括解锁)

封锁协议

- 共享型封锁
  - **PS协议**　其主要内容是：事务若要存取数据，则必须先提出对此数据对象的S封锁请求；事务如果未获准S封锁，那么进入等待状态，直至获准在此数据对象上S封锁，才能继续执行；事务在更新数据前必须先将它在该数据对象上的S封锁升级（**UPGRADE**）为X封锁。
  - PSC协议　是在“PS协议”上再加一规定：解锁操作合并到事务的结束（COMMIT或ROLLBACK）操作中。PSC协议可以解决丢失更新、读“脏数据”和不一致分析问题。

解决不一致分析和读“脏”数据问题：

时间	事务T1	数据库中A、B的值	事务T2
t_0		50、100	
t_1	加锁 SFIND A, B	T1:♀A、B加S锁	
t_2	求和：A+B=150		
t_3		T2:♀A、B加S锁	加锁： SFIND B
T_4			更新： B:=B*2
t_5			写回B=200 UPDX B 失败
t_6			等待
t_7	检索(验算)A＝50,B=100		等待
t_8	求和：A+B=150		等待
t_9	COMMIT(包括解锁A,B)	T1:♁解锁A,B	等待
t_{10}		T2:♀B加X锁	写回B=200:UPDX B重做
t_{11}		50,200	COMMIT(包括解锁B)

二、封锁机制

- 封锁相容矩阵

T2 \ T1	X	S	None
X	N	N	Y
S	N	Y	Y

三、活锁和死锁

- 恰当地运用封锁技术，可以保证并发调度的正确性、有效地避免数据不一致，但有可能引起活锁和死锁问题。
  - ① 活锁：出现某个事务永远处于等待状态得不到执行的现象。
  - 避免活锁的一个简便方法是采用“先来先服务”排队的策略。封锁子系统按请求在时间上的先后次序对事务排序,数据对象上原有的锁一释放,即执行队列中第一个事务的封锁请求。

活　锁：

事务T1	数据A	事务T2	事务T3	事务T4
XFIND A	T1 ♀加X锁		◦	
◦		XFIND A (失败)		
◦		等待	XFIND A (失败)	
解锁 A	T1 ♂解锁	等待	等待	XFIND A (失败)
◦	T3 ♀加X锁	等待	XFIND A (重做)	等待
		等待	◦	◦
		等待	◦	◦
		等待	◦	◦
	T3 ♂解锁	等待	解锁 A	等待
		等待	◦	等待
	T4 ♀加X锁	等待	◦	XFIND A (重做)
		等待	◦	◦

避免活锁的 简单方法是：采用先来先服务的策略。

三、活锁和死锁

- ② 死锁：出现若干事务因循环等待而无法继续执行的现象。
- 例： 两个事务T1和T2已分别封锁了数据D1和D2。 T1和T2由于需要各自分别申请封锁D2和D1,但是由于D2和D1已被对方封锁,因而T1和T2只能等待。 而T1和T2由于等待封锁而不能结束,从而使对方的封锁申请也永远不能被选中,这就形成了死锁。
- 死锁的诊断 DBMS周期地测试系统中是否出现了死锁。用事务依赖图的形式可以测试系统中是否存在死锁。如果在事务依赖图中，沿着箭头方向存在一个循环，那么表示已出现死锁现象。

“死 锁”：系统中有两个或两个以上的事务都处于等待状态，并且每个事务都在等待其中另一个事务解除封锁，它才能继续执行下去，结果造成任何一个事务都无法继续执行。

事务T1	数据A、B	事务T2
加锁:XFIND A	T1 令 A 加 X锁	
。	T2 令 B 加 X锁	加锁:XFIND B
。		
加锁:XFIND B（失败）		
等待		加锁:XFIND A（失败）
等待		等待
等待		等待
。		。
。		。
。		。

四、并发事务的可串行化调度

- 事务的调度、串行调度和并发调度
  - 事务的执行次序称为“调度”。
  - 若多个事务依次执行，则称为事务的串行调度。
  - 若设法（例如利用分时的方法）多个事务同时被处理（即交错执行），则称为事务的并发调度。
  - 在应用环境中，事务的任一串行调度都是有效的（正确的），但是事务的并发调度不一定是正确的，有可能产生前面提到的不一致性问题。

现在有两个事务，分别包含下列操作：

事务T1：读B；A=B十1；写回A；

事务T2：读A；B=A十1；写回B

假设A的初值为10，B的初值为2。

下图给出了对这两个事务的三种不同的调度策略。

（a）和（b）为两种不同的串行调度策略，虽然执行结果不同，但它们都是正确的调度。

（c）中两个事务是交错执行的，由于其执行结果与（a）、（b）的结果都不同，所以是错误的调度。

（d）中两个事务也是交错执行的，由于其执行结果与串行调度1（图（a））的执行结果相同，所以是正确的调度。

（a）串行调度1(先T1后T2)

时间	事务T1	数据库中A、B的值	事务T2
t_0		10、2	
t_1	检索B：B=2		
t_2	修改：　A←B+1		
t_3	写回A：A=3	3、2	
t_5			检索A：A=3
t_6			修改：B←A+1
t_7			写回B：B=4
t_8		3、4	

（b）串行调度2（先T2后T1）

时间	事务T1	数据库中A、B的值	事务T2
t_0		10、2	
t_1			检索A：A=10
t_2			修改B:B←A+1
t_3			写回B：B=11
t_5		10、11	
t_6	检索B：B=11		
t_7	修改A：A←B+1		
t_8	写回A：A=12	12、11	

不可串行化调度(交错执行)

时间	事务T1	数据库中A、B的值	事务T2
t_0		10、2	
t_1	检索B：B=2		
t_2			检索A:A=10
t_3	修改A：A←B+1		
t_5	写回A：A=3		
t_6		3、2	修改B:B←A+1
t_7			写回B：B=11
t_8		3、11	

（d）可串行化调度（结果同串行调度1）

时间	事务T1	数据库中A、B的值	事务T2
t_0		10、2	
t_1	检索B：B=2		
t_2			检索A：等待
t_3	修改A：A←B+1		等待
t_5	写回A：A=3		等待
t_6		3、2	检索A：A=3
t_7			修改B:B←A+1
t_8			写回B：B=4
t_9		3、4	

为了保证并行操作的正确性:

DBMS的并行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，这也是最简单的调度策略，但这种方法实际上是不可行的，因为它使用户不能充分共享数据库资源。

四、并发事务的可串行化调度

- 可串行化概念
  - 如果一个并发调度与某一串行调度具有相同的执行结果，那么称这个并发调度是“可串行化的调度”（即正确的并发调度），否则是不可串行化的调度。
- 两段封锁协议
  - 事务在对数据对象存取之前必须先获得对此数据对象的封锁，事务在解除了一个锁之后不再获得任何锁。

四、并发事务的可串行化调度

- 两段封锁协议
  - 若所有事务都遵守两段封锁协议,则对这些事务的任何并发调度策略都是可串行化的;
  - 但若并发事务的一个调度是可串行化的,并不一定所有事务都符合两段封锁协议。
  - 两段封锁协议与防止死锁的一次封锁法是不同的。 一次封锁法符合两段封锁协议,但两段封锁协议并不要求一次封锁法。 因此,遵守两段封锁协议的事务也有可能发生死锁。
- 封锁的粒度
  - 封锁对象的规模称为封锁粒度(Granularity)。

五、SQL中事务的存取模式和隔离级别

- 事务的存取模式
  - **READ ONLY（只读型）**
  - **READ WRITE（读写型）**
- 事务的隔离级别
  - **SERIALIZABLE（可串行化）**
  - **REPEATABLE READ （可重复读）**
  - **READ COMMITED（读提交数据）**
  - **READ UNCOMMITED（读未提交数据）**

第四节 数据库的完整性

- 完整性子系统和完整性规则
- SQL中的完整性约束
- SQL3的触发器

一、完整性子系统和完整性规则

- 完整性：是指数据的正确性、有效性和相容性。
- 完整性约束条件：为保证数据的完整性而规定的条件。
- 完整性检查：检查DB中数据是否满足完整性约束条件。
- 完整性子系统：DBMS中执行完整性检查的子系统。其功能：
  - ①监督事务的执行，测试其是否违反完整性约束条件；
  - ②若有违反，则按进行预定的处理。

一、完整性子系统和完整性规则

- 完整性规则的组成
- ① 每个规则包括三部分：
  - 触发条件 规定何时执行本规则进行检查；
  - 约束条件 又称谓词，定义应满足的条件；
  - 否则子句 规定不满足条件时该作的处理。
- ② 在关系DB中，完整性规则可分为三类：
  - 域完整性规则：定义属性取值范围。
  - 域联系的规则：定义属性间的联系、影响和约束。
  - 关系完整性规则：定义更新操作对值的影响和限制。
- ③ 完整性规则用DDL描述，由系统执行检查。

二、SQL中的完整性约束

- 完整性约束分为三大类：
  - 域约束
  - 基本表约束
  - 断言

1、域完整性规则：定义属性的取值范围——属性值约束。

包括：域约束子句、非空值约束、基于属性的检查子句。

①用“CREATE DOMAIN”语句定义新的域,并可出现CHECK子句。

例： 定义一个新的域DEPT，可用下列语句实现：

```
CREATE DOMAIN DEPT CHAR （20） DEFAULT ‘计算机软件’
CONSTRAINT  VALID_ DEPT                    /*域约束名字*/
CHECK（VALUE IN (‘计算机科学与技术’,‘计算机软件’)）;
```

允许域约束上的CHECK子句中可以有任意复杂的条件表达式。

②非空值约束（NOT NULL）

例： SNO char(4) NOT NULL

③基于属性的检查子句(CHECK)：

例：CHECK (GRADE IS NULL) OR

(GRADE BRTWEEN 0 AND 100)

- 2、基本表约束
- ①主键约束
  - 可用主键约束来描述实体完整性规则。主键约束可用两种形式表示：主键子句和主键短语。
  - 主码可在定义关系的**CREATE TABLE**语句中使用**PRIMARY KEY**关键字加以定义。有两种定义主键的方法，一种是在属性后增加关键字，另一种是在属性表中加入额外的定义主键的子句：
  - **PRIMARY KEY(**主键属性名表**)**
  - 使用关键字**UNIQUE**，说明该属性（或属性组）的值不能重复出现。

- ②外键约束

FOREIGN KEY（〈列名序列1〉）.

REFERENCES <参照表> [（<列名序列2>）]

[ON DELETE <参照动作>]

[ON UPDATE <参照动作>]

  - 外部码的取值只有两种情况：①要么取空值；②要么取参照关系中的主码值。当用户的删除或修改操作违反了上述规则时，如何保持此种约束呢？SQL中提供了五种可选方案供数据库实现者使用。

参照动作可以有五种方式：

NO ACTION（无影响）

CASCADE（级联方式）

RESTRICT（受限方式）

SET　NULL（置空值）

SET　DEFAULT（置缺省值）

- ③ 检查约束
  - 对单个关系的元组值加以约束。
  - 对表内元组说明约束时，在CREATE TABLE语句中的属性表、主码、外部码的说明之后加上CHECK子句。每当对元组进行插入或修改操作时，都要对CHECK子句的条件表达式求值，如果条件为假，违背了约束，系统将拒绝该插入或修改操作。
  - CHECK子句的一般格式为： CHECK <条件>

- 3、断言
  - 如果完整性约束与多个关系有关，或者与聚合操作有关，SQL提供“断言”（Assertions）机制让用户书写完整性约束。
  - 定义：
  - CREATE ASSERTION 断言名 CHECK（条件）
  - 撤消：
  - **DROP** ASSERTION 断言名

例：设有三个关系模式：

EMP（ENO，ENAME，AGE，SEX，ECITY）

COMP（CNO，CNAME，CITY）

WORK（ENO，CNO，SALARY）

试用SQL的断言机制定义下列完整性约束：

①每个职工至多可在3个公司兼职工作:

```
CREATE ASSERTION  ASSE1  CHECK
        ( 3 >= ALL (SELECT COUNT(CNO)
                        FROM WORK
                        GROUP BY ENO ) );
```

② 每门公司男职工的平均年龄不超过40岁:

```
CREATE  ASSERTION  ASSE2  CHECK
  (40 >= ALL (SELECT  AVG(EMP.AGE)
                FROM  EMP,WORK
                WHERE EMP.ENO=WORK.ENO
                  AND SEX='男'
                GROUP  BY  CNO) );
```

③ 不允许女职工在建筑公司工作:

```
CREATE  ASSERTION  ASSE3  CHECK
 ( NOT  EXISTS  (SELECT  *
                  FROM  WORK
                  WHERE CNO IN (SELECT  CNO
                                 FROM  COMP
                                WHERE  CNAME ='建筑公司'))
               AND  ENO  IN (SELECT  ENO
                              FROM  EMP
                              WHERE  SEX='女')));
```

断言也可以在关系定义中用检查子句形式定义，但是检查子句不一定能保证完整性约束彻底实现，而断言能保证不出差错。

三、SQL3的触发器

- 定义：
  - 触发器（Trigger）不仅能实现完整性规则，而且能保证一些较复杂业务规则的实施。所谓触发器就是一类由事件驱动的特殊过程，一旦由某个用户定义，任何用户对该触发器指定的数据进行增、删或改操作时，系统将自动激活相应的触发器，在核心层进行集中的完整性控制。
- 触发器结构分三部分：
  - 事件：对数据库的插入、删除和修改等操作。
  - 条件：触发器将测试条件是否成立。
  - 动作：如果测试满足预定的条件，就由DBMS执行这些动作。

三、SQL3的触发器

- 触发器结构组成
  - 触发事件包括表中行的插入、删除和修改，即执行INSERT、DELETE、UPDATE语句。在修改操作（UPDATE）中，还可以指定，特定的属性或属性组的修改为触发条件。事件的触发还有两个相关的时间：Before和After。Before触发器是在事件发生之前触发，After触发器是在事件发生之后触发。

三、SQL3的触发器

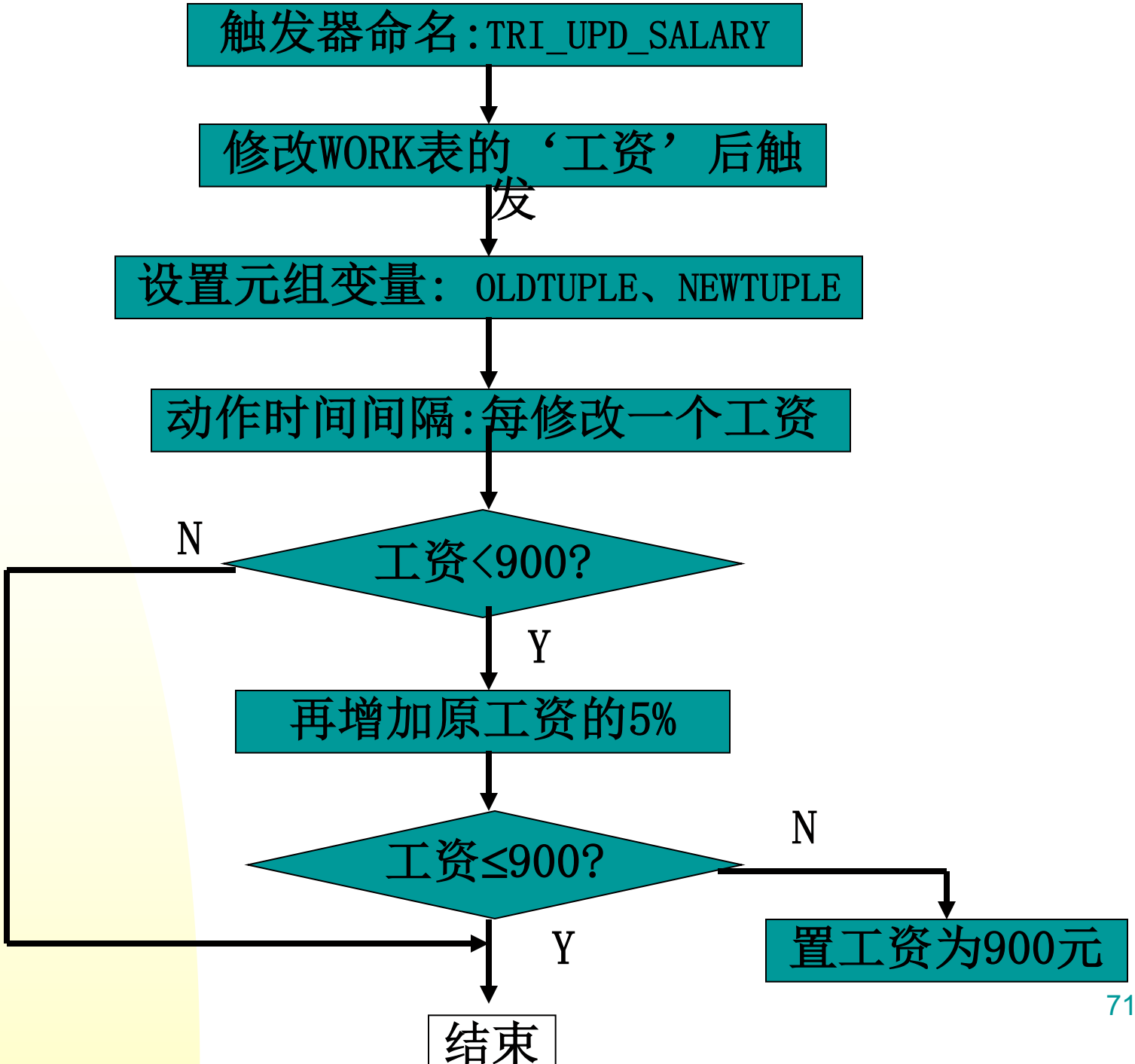
- 触发器结构组成
  - 触发动作实际上是一系列SQL语句，可以有两种方式：
  - (1)对被事件影响的每一行（FOR EACH ROW）——每一元组执行触发过程，称为行级触发器。
  - (2)对整个事件只执行一次触发过程（FOR EACH STATEMENT）称为语句级触发器。该方式是触发器的默认方式。所以综合触发时间和触发方式，触发器的基本类型如下表所示。

	FOR EACH STATEMENT(默认)	FOR EACH ROW
BEFORE 选项	语句前触发器： 在执行触发语句前激发触发器一次	行前触发器： 在修改由触发语句所影响的行前，激发触发器一次
AFTER 选项	语句后触发器： 在执行触发语句后激发触发器一次	行后触发器： 在修改由触发语句所影响的每一行后，激发触发器

SQL3的触发器实例

【实例1】 某单位修改工资原则:如果职工工资修改后仍低于900元，那么，在修改后的工资基础上再增加原工资的5%，但不得超过900元（元组级触发器 ）。

【实例2】 在学习关系SC表中修改课程号CNO,即学生的选课登记需作变化。在关系SC中的约束：要求保持每门课程选修人数不超过50。如果更改课程号后，违反这个约束，那么这个更改应该不做。（语句级触发器）


触发器命名:TRI_UPD_SALARY
修改WORK表的‘工资’后触发
设置元组变量: OLDTUPLE、NEWTUPLE
动作时间间隔:每修改一个工资
工资<900?
N
Y
再增加原工资的5%
工资≤900?
N
Y
置工资为900元
结束

```
CREATE TRIGGER TRI_UPD_SALARY      /*触发器命名*/
 AFTER UPDATE OF SALARY ON WORK /*触发时间,          ,目标表
*/
  REFERENCING                          /*设置必要的变量*/
     OLD AS OLDTUPLE                 /*为元组级触发器设置变量*/
     NEW AS NEWTUPLE
  FOR EACH ROW                        /*触发器的动作时间间隔*/
 WHEN ( 900 > NEWTUPLE.SALARY )    /*触发器的动作条件部分*/
```

```
BEGIN ATOMIC
  UPDATE WORK                                    /*触发器的动作部分1*/
    SET SALARY=NEWTUPLE.SALARY+OLDTUPLE.SALARY*0.05
   WHERE ENO=NEWTUPLE.ENO
   AND (900>NEWTUPLE.SALARY+OLDTUPLE.SALARY*0.05);
  UPDATE WORK                                    /*触发器的动作部分2*/
    SET SALARY=900
   WHERE ENO=NEWTUPLE.ENO
   AND (900≤NEWTUPLE.SALARY+OLDTUPLE.SALARY*0.05);
  END;
```

【实例2】在学习关系SC表中修改课程号CNO,即学生的选课登记需作变化。在关系SC中的约束：要求保持每门课程选修人数不超过50。如果更改课程号后，违反这个约束，那么这个更改应该不做。（语句级触发器）

```
CREATE  TRIGGER  TRI_UPD_SC                          /*触发器的命名*/
  INSTEAD  OF  UPDATE OF CNO  ON  SC                /*时间、事件、目标*/
  REFERENCING                                       /*设置变量*/
    OLD_TABLE  AS  OLDSTUFF                         /*为语句级触发器设置变量*/
    NEW_TABLE  AS  NEWSTUFF
 WHEN  (50 >= ALL (SELECT  COUNT(SNO)               /*动作时间条件*/
    FROM ((SC EXCEPT OLDSTUFF) UNION NEWSTUFF)
    GROUP  BY  CNO) ) )
  BEGIN ATOMIC                                      /*动作体*/
    DELETE FROM  SC                                 /*触发动作1*/
    WHERE (SNO,CNO,GRADE) IN OLDSTUFF;
    INSERT INTO SC                                  /*触发动作2*/
    SELECT * FROM NEWSTUFF
  END;
```

触发器的动作时间:INSTEAD OF（第2行），任何企图修改关系SC中CNO值都被这个触发器截获，并且触发事件的操作（即修改CNO）不再进行，由触发器的条件真假值来判断是否执行动作部分的操作。

INSTEAD OF 表示：**在触发事件发生时,只要满足WHEN条件，就执行动作部分的操作，而触发事件的操作不再执行。**

动作部分的操作由两个SQL语句组成，前一个语句是从关系SC中删除修改前的元组，后一个语句是在关系SC中插入修改后的元组。用这样的方式完成触发事件的操作。

因为是语句级触发器，所以没有FOR EACH ROW，在这里FOR EACH STATEMENT也省略了。

四、SQL Server的数据库完整性及实现方法

SQL Server具有较健全的数据库完整性控制机制。

SQL Server使用约束、缺省，规则和触发器4种方法定义和实施数据库完整性功能。

1、SQL Server的数据完整性的种类

SQL Server中的数据完整性包括 域完整性、实体完整性和参照完整性3种。

（1）域完整性为列级和元组级完整性----为列或列组指定一个有效的数据集，并确定该列是否允许为空。

（2）实体完整性为表级完整性---要求表中所有的元组都应该有一个唯一的标识符(主码)。

（3）参照完整性是表级完整性---维护参照表中的外码与被参照表中主码的相容关系。

2、SQL Server数据完整性的两种方式

SQL Server 使用声明数据完整性和过程数据完整性两种方式实现 数据完整性控制。

（1）声明数据完整性：通过在对象定义中定义、系统本身自动强制来实现。声明数据完整性包括各种约束、缺省和规则。

（2）过程数据完整性：通过使用脚本语言（主语言或TransactSOL）定义，系统在执行这些语言时强制完整性实现。过程数据完整性包括触发器和存储过程等。

3、SQL Server 实现数据完整性的具体方法有4种：

约束、缺省、规则和触发器。

（1）SQL约束类型——－－－效率高

可在定义、修改表语句中定义。

约束是通过限制列中的数据、行中的数据和表之间数据来保证数据完整性的方法：

（2） 缺省和规则————一功能较低开支大

缺省（DEFAULT）和规则（RULE）都是数据库对象。当它们被创建后，可以绑定到一列或几列上，并可以反复使用。

（3) 触发器－－－－－－高功能高开支的数据完整性方法

①Inserted和deleted 表

当触发器被执行时，SQL Server创建一个或两个临时表（Inserted或者deleted 表）。当一个记录插入到表中时，相应的插入触发器创建一个inserted表, 该表镜像该触发器相连接的表的列结构。

②Update()函数

Update（）函数只在插入和更新触发器中可用，它确定用户传递给它的列是否已经被引起触发器激活的insert或update语句所作用。

（3) 触发器－－－高功能高开支的数据完整性方法

①Inserted和deleted 表

当触发器被执行时，SQL Server创建一个或两个临时表（Inserted或者deleted 表）。当一个记录插入到表中时，相应的插入触发器创建一个inserted表, 该表镜像该触发器相连接的表的列结构。

②Update()函数

Update（）函数只在插入和更新触发器中可用，它确定用户传递给它的列是否已经被引起触发器激活的insert或update语句所作用。

```
CREATE  TRIGGER  SC_UPDATA_GGRADE ON [dbo].[SC]
FOR   UPDATE                                        /*事件*/
AS
DECLARE @old_ggrade real,                           /*定义变量*/
                @new_ggrade real
BEGIN                                               /*动作体*/
   SELECT   @old_ggrade =ggrade
   FROM       deleted
   SELECT    @new_ggrade = ggrade
   FROM       inserted
   IF  update(ggrade)                               /*条件*/
     IF @old_ggrade >@new_ggrade
       ROLLBACK  TRANSACTION                        /*动作*/
END
```

过程数据完整性： 通过使用脚本语言（主语言或TransactSOL）定义，系统在执行这些语言时强制完整性实现。

过程数据完整性包括触发器和存储过程等。

存储过程的使用

(上机验证 P.292存储过程举例)

第五节 数据库的安全性

- 安全性级别
- 权限
- 安全性和授权
- 数据加密
- 自然环境的安全性

一、安全性级别

- 定义
  - 是指保护DB，防止不合法的使用，以免数据的泄漏、非法更改和破坏。
- 安全性级别
  - 设置安全措施所牵涉的层次。分成环境级、职员级、OS级、网络级和DBS级等五个级别。这里只讨论DBS级的安全性问题。

二、权限

- 权限的授予、转授与回收
  - 授权　可根据需要把在某对象上的某些权限授予特定的用户。
  - 转授　若拥有转授权，则允许把已获得的权限再转授给其他用户。
  - 回收　也可以撤消已授给某用户的某些权限。
- 权限的种类：
  - 读
  - 插入
  - 修改
  - 删除

三、安全性和授权

- 安全性由两个机制提供：视图和授权子系统。
  - 视图　视图是虚表，视图机制使DBS具有三个优点：数据安全性、逻辑独立性、用户操作简便性。
  - SQL中的用户权限及操作
    - 六类权限：**SELECT，INSERT，DELETE，UPDATE，REFERENCES，USAGE。**
    - 授权（GRANT）语句：将关系和视图操作权授予特定用户
    - 回收（REVOKE）语句：回收已授给某用户的权限。

三、安全性和授权

- 一般授权是指授予某用户对某数据对象进行某种操作的权利。 在SQL语言中,DBA及拥有权限的用户可用GRANT语句向用户授权。

  GRANT语句的格式：

  GRANT<权限表> ON <数据库元素>

  TO<用户名表>

  [WITH GRANT OPTION］；

- 其中,<数据库元素>规定了数据对象,如：TABLE student(基本表student)；

<权限表>规定了可以对<数据库元素> 所执行的操作,如： SELECT,UPDATE <用户名表>规定了得到权力的用户的用户标识符。

三、安全性和授权

- **WITH GRANT OPTION**的意义
- 如果在**GRANT**语句中选择了**WITH GRANT OPTION**的子句,则获得规定权限的用户不仅自己可以执行这些操作,还获得了用**GRANT**语句把这些权限授予其他用户的权限； 如果在**GRANT**语句中未选择此子句,则获得规定权限的用户仅仅只能自己执行这些操作,不能传播这些权限。

三、安全性和授权

- 用一般授权格式**GRANT**授出的权限者（及**DBA**）可用对应格式**REVOKE**语句收回之。 此**REVOKE**语句的格式为：

  REVOKE <权限表> ON <数据库元素>

  FROM<用户名表>

- 本语句将把**FROM**子句指定的所有用户，［对**ON**子句指定的数据对象］所具有的**<权限表>**全部收回。
- **(1)** 只有使用**GRANT**授出了权限的用户（及**DBA**）才能使用本语句收回自己授出去的权限。
- **(2)** 若**<用户名表>** 中,有些用户还把所授出的权限授予其他用户（因为当授权时，带有**WITH GRANT OPTION**子句）,则间接收到此权限的用户也自动被收回了这些权限。

例1：把对关系S的查询、修改权限授给用户WANG，并且WANG还可以把这些权限转授给其他用户：

```
GRANT  SELECT，UPDATE  ON  S  TO  WANG
WITH  GRANT  OPTION
```

例2:允许用户BAO建立新关系，并可以引用关系C的主键CNO作为新关系的外键，并有转让权限。

```
GRANT  REFERENCES  （CNO）  ON  C  TO  BAO
WITH  GRANT  OPTION
```

例3:从用户WANG连锁回收对关系S的查询、修改权限。

```
REVOKE  SELECT，UPDATE  ON  S  FROM  WANG  CASCADE
```

四、数据加密

- 数据加密　数据存储和传输时采用加密技术。
- 数据是存储在介质上的,数据还经常通过通信线路传输。　敌手既可在介质上窃取数据,也可在通信线路上窃听到数据,有时,跟踪审计的日志文件中也找不到敌手的踪影。　对敏感的数据进行加密储存是防止数据泄露的有效手段。　原始的数据（称为明文**Plain text**）在加密密钥的作用下,通过加密系统加密成密文（**Cipher text**）。　明文是大家都看得懂的数据,一旦失窃,后果严重。

四、数据加密

- 但密文是谁也看不懂的数据,只有掌握解密密钥的人，才能在解密密钥的帮助下,由解密系统解密成明文。 因此,单单窃得密文数据是没有用处的。
- 数据加、 解密的代价也不小。 因此,实际**DBMS**往往把加密特性作为一种可选功能,由用户决定是否选用。 如果选用了加密功能,用户必须要保管好自己的加密密钥和解密密钥,不能失去或泄露。

四、数据加密

- 失去密钥,则自己都无法知道密文的真实内容。 泄露密钥,则还不如不采用加密功能,采用一个不安全的加密系统,还不如采用一个不加密的系统来得安全。
- 数据的安全性和完整性是两个不同的概念。 数据的安全性是防止数据库被恶意破坏和非法存取,而数据完整性是为了防止错误信息的输入,保证数据库中的数据符合应用环境的语义要求。 安全性措施的防范对象是非法用户和非法操作,而完整性措施的防范对象是不合语义的数据。

五、自然环境的安全性

- **DBS的设备、硬件和环境的安全性。**