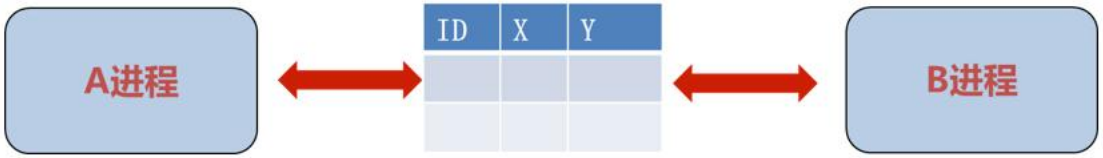
事务管理

事务

- 事务的引入
  - 计算环境的脆弱性—故障恢复问题
  - 计算环境的分布性—并发控制问题

并发操作：在多用户共享系统中，如果多个用户同时对同一个数据进行操作，称为并发操作。



- 例如
  - 银行转帐（故障恢复）
  - 订飞机票（并发执行）

事务概念

- 事务定义
  - **事务**是由一系列操作序列构成的程序执行单元，这些操作要么都做，要么都不做，是一个不可分割的工作单位(构成单一逻辑工作单元的操作集合)

    例如银行转帐、订飞机票等

事务概念

- SQL中事务的定义
  - 事务以Begin transaction开始
  - 事务以Commit或 Rollback结束
    - Commit表示提交（将事务中所有对DB的操作写入磁盘），事务正常结束
    - Rollback表示事务非正常结束，撤消事务已做的操作，回滚到事务开始时的状态

例：删除仓库“WH1”，并将职工T表当中所有在“WH1”仓库中职工记录删除。

```
BEGIN TRANSACTION MYDEL
DELETE FROM 仓库T
WHERE 仓库号='WH1'
DELETE FROM 职工T
WHERE 仓库号='WH1'
IF @@ERROR<>0
ROLLBACK TRANSACTION MYDEL
ELSE
COMMIT TRANSACTION MYDEL
```

- 事务可以是：

一条SQL；
一组SQL；
整个程序

- 一个程序可包含多个事务

问题

- 示例

银行转帐：事务T从A帐户过户50￥到B帐户

T：
```
read(A);
A := A – 50;
write(A);
read(B);
B := B + 50;
write(B);
```

read(X)：从数据库传送数据项X到事务的工作区中

write(X)：从事务的工作区中将数据项X写回数据库

原子性问题

- 示例

银行转帐：事务T从A帐户过户50￥到B帐户

T：
```
read(A);
A := A – 50;
write(A);
read(B);
B := B + 50;
write(B);
```

发生故障

read(X)：从数据库传送数据项X到事务的工作区中

write(X)：从事务的工作区中将数据项X写回数据库

事务的特性

事务具有4个特性，简称ACID特性

① 原子性（atomicity）:事务是一个不可分隔的整体，事务中的语句要么都执行，要么都不执行；

② 一致性(consistency)：事务必须使数据库中的从一种状态变换到另一种状态时要保证数据一致；

③ 隔离性(isolation)：指并发事务之间的干扰性，有具体的隔离级别决定；

④ 持续性(Durability)：事务提交后，对数据库的更改是永久的，不能再撤消 。

一致性问题

- 示例

银行转帐：事务T从A帐户过户50￥到B帐户

T：
```
read(A);
A := A – 50;
write(A);
read(B);
B := B + 50;
write(B);
```

转帐前后两个帐户金额之和应保持不变

read(X)：从数据库传送数据项X到事务的工作区中

write(X)：从事务的工作区中将数据项X写回数据库

事务的特性

事务具有4个特性，简称ACID特性

① 原子性（atomicity）:事务是一个不可分隔的整体，事务中的语句要么都执行，要么都不执行；

② 一致性(consistency)：事务必须使数据库中的从一种状态变换到另一种状态时要保证数据一致；

- 当DB中只包含成功事务提交的结果时，数据库处于一致性状态；
- 事务开始前，数据库处于一致性的状态；事务结束后，数据库必须仍处于一致性状态

③ 隔离性(isolation)：指并发事务之间的干扰性，有具体的隔离级别决定；

④ 持续性(Durability)：事务提交后，对数据库的更改是永久的，不能再撤消 。

隔离性问题

- 示例

银行转帐：事务T从A帐户过户50￥到B帐户

T：
```
read(A);
A := A – 50;
write(A);
read(B);
B := B + 50;
write(B);
```

另外一个事务：读取并修改A，B的值

read(X)：从数据库传送数据项X到事务的工作区中

write(X)：从事务的工作区中将数据项X写回数据库

事务的特性

事务具有4个特性，简称ACID特性

① 原子性（atomicity）:事务是一个不可分隔的整体，事务中的语句要么都执行，要么都不执行；

② 一致性(consistency)：事务必须使数据库中的从一种状态变换到另一种状态时要保证数据一致；

③ 隔离性(isolation)：指并发事务之间的干扰性，有具体的隔离级别决定；

系统必须保证事务不受其它并发执行事务的影响

对任何一对事务T_1，T_2，在T_1看来，T_2要么在T_1开始之前已经结束，要么在T_1完成之后再开始执行

④ 持续性(Durability)：事务提交后，对数据库的更改是永久的，不能再撤消 。

永久性问题

- 示例

银行转帐：事务T从A帐户过户50￥到B帐户

T：
```
read(A);
A := A – 50;
write(A);
read(B);
B := B + 50;
write(B);
```

转账成功后，任何其他操作或故障都不会引起此次转账相关数据丢失

read(X)：从数据库传送数据项X到事务的工作区中

write(X)：从事务的工作区中将数据项X写回数据库

事务的特性

事务具有4个特性，简称ACID特性

① 原子性（atomicity）:事务是一个不可分隔的整体，事务中的语句要么都执行，要么都不执行；

② 一致性(consistency)：事务必须使数据库中的从一种状态变换到另一种状态时要保证数据一致；

③ 隔离性(isolation)：指并发事务之间的干扰性，有具体的隔离级别决定；

④ 持续性(Durability)：事务提交后，对数据库的更改是永久的，不能再撤消 。

一个事务一旦提交之后，它对数据库的影响必须是永久的；
系统发生故障不能改变事务的持久性

事务ACID面临的问题

- 多个事务并发运行时，不同事务操作交叉执行；

并发控制

- 事务在运行过程中被强行停止。

恢复机制

- 恢复机制与并发控制机制的提出
  - 事务在运行过程中因某种故障被强行终止，数据库一致性被破坏，需进行恢复。
  - 多个事务并行运行时，不同事务的各种操作交叉进行，为保证各事务的执行互不干扰，需进行并发控制。
- 事务是恢复和并发控制的基本单位

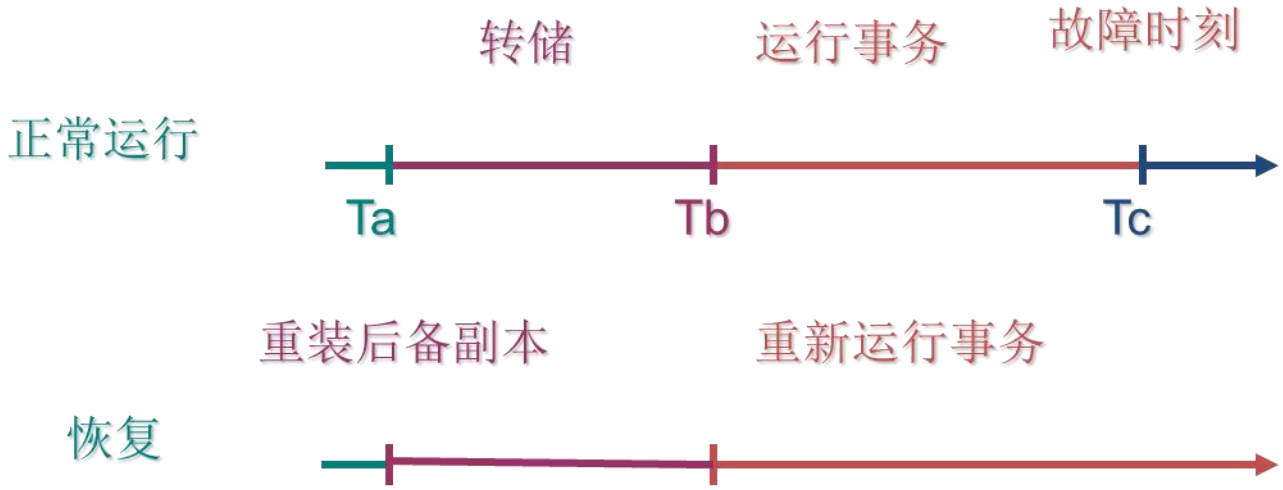
数据库故障

- 事务故障：事务故障意味着事务没有达到预期的终点，因此数据库可能处于不正确的状态。
- 系统故障：系统故障又称为软故障。系统故障是指造成系统停止运转的任何事情，使得系统要重新启动。
- 介质故障：使存储在外存上的数据部分损失或全部损失，称为介质故障。介质故障又称为硬故障。
- 恶意破坏：主要指计算机病毒、非法入侵数据库系统破坏数据。如“黑客”。

数据恢复

- 数据库恢复机制，它包括一个数据库恢复子系统和一套特定的数据结构。
- 数据库的恢复基本原理很简单，就是“冗余”，即数据库数据重复存储。
- 最常用的技术是：数据转储和登录日志文件
- 事务故障的恢复是由系统自动完成的，对用户是透明的。

- 转储：即DBA定期地将整个数据库复制到磁带、光盘或另一个磁盘上保存起来的过程。这些备用的数据文本称为后备副本或后援副本。
- 日志文件是用来记录事务的每一次对数据库更新操作的文件，包括用户的更新操作以及由此引起的系统内部的更新操作。它记录最近一次后备副本后的所有数据库的变更以及所有事务的状态（已COMMIT或未COMMIT）。



系统恢复

系统故障发生时，造成数据库不一致状态的原因有两个：

- 一是由于一些未完成事务对数据库的更新已写入数据库。
  恢复策略：根据日志文件执行UNDO操作
- 二是由于一些已提交事务对数据库的更新还留在缓冲区没来得及写入数据库。

  恢复策略：要根据日志文件执行REDO操作。

并发操作的问题

并发操作引起数据的不一致。

(1) 丢失更新

(2) 不可重复读

(3) 读“脏”数据

三类问题

T1	T2
(1) 读A=10	
(2)	读A=10
(3) A←A-1	
(4)	A←A-1
(5) 写回A=9	
(6)	写回A=9

(a) 丢失更新

T1	T2
(1) 读A=10	
(2)	读A=10
A←A-2	
(3)	写回A=8
(4) 读A=8	

(b) 不可重复读

T1	T2
(1) 读A=100	
A←A*2	
写回A=200	
(2)	读A=200
(3) ROLLBACK	
A恢复为100	
(4)	

(c) 读“脏”数据

讨论

- 例：
  - 旅客A来到A售票处，要买一张15日北京到上海的13次直达快速列车的软卧车票，售票员A（下称用户A）在终端A查看剩余票信息；
  - 几乎在同时，旅客B来到B售票处，也要买一张15日北京到上海的13次直达快速列车的软卧车票，售票员B（下称用户B）从终端B查到了同样的剩余票信息；
  - 旅客A买了一张15日13次7车厢5号下铺的软卧票，用户A更新剩余票信息并将它存入数据库；
  - 这时用户B不知道用户A已经将15日13次7车厢5号下铺的软卧票卖出，使旅客B也买了一张15日13次7车厢5号下铺的软卧票，用户B更新剩余票信息并将它存入数据库（重复了用户A已经做过的更新）。

并发控制

- DBMS的并发控制：负责协调并发事务的执行，使得事务的执行不受其他事务的干扰，保证数据库的完整性、同时避免用户得到不正确的数据。
- DBMS的并发控制常用的方法：封锁法、时间印法和乐观控制法，商用的DBMS一般都采用封锁法。

封锁

- 封锁的定义
  - 封锁就是一个事务对某个数据对象加锁，取得对它一定的控制，限制其它事务对该数据对象的使用。
  - 并发控制的基本方法就是封锁。
- 封锁的类型
  - **排它锁**（X锁，eXclusive lock，写锁）：事务T对数据对象R加上X锁，则其它事务对R的任何封锁请求都不能成功，直至T释放R上的X锁。
  - **共享锁**（S锁，Share lock，读锁）：事务T对数据对象R加上S锁，则其它事务对R的X锁请求不能成功，而对R的S锁请求可以成功。

并发控制过程

- 事务**对数据项进行操作前**根据操作类型向并发控制器申请相应类型的锁，并发控制器根据该数据项上的加锁情况，决定授予或者拒绝授予所需锁。

相容矩阵

T_1 \ T_2	X	S
X	N	N
S	N	Y

不相容请求

相容请求

并发控制

- 封锁协议：事务加锁和解锁须遵从的规则
  - 何时申请X锁或S锁
  - 持锁时间、何时释放
- 常用的封锁协议：三级封锁协议

1级封锁协议

- 事务T在修改数据R之前必须先对其加X锁，直到事务结束才释放
  - 正常结束（COMMIT）
  - 非正常结束（ROLLBACK）
- 1级封锁协议可防止丢失修改
- 在1级封锁协议中，如果是读数据，不需要加锁的，所以它不能保证可重复读和不读“脏”数据。

T1	T2
Lock X(A)	
读A=16	
	Lock X(A)
A←A-1	Wait
写A=15	Wait
Commit	Wait
UnLock	Wait
X(A)	Lock X(A)
	读A=15
	A←A-1
	写A=14
	Commit
	UnLock
	X(A)

(a) 没有丢失更新

2级封锁协议

- 1级封锁协议+事务T在读取数据R前必须先加S锁，读完后即可释放S锁
- 2级封锁协议可以防止丢失修改和读“脏”数据。
- 在2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。

(c) 不再读“脏”数据

3级封锁协议

- 1级封锁协议 + 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
- 3级封锁协议可防止丢失修改、读脏数据和不可重复读。

T_1	T_2
① Slock A	
读A=50	
Slock B	
读B=100	
求和=150	
②	Xlock B
	等待
	等待
③ 读A=50	等待
读B=100	等待
求和=150	等待
Commit	等待
Unlock A	等待
Unlock B	等待
④	获得Xlock B
	读B=100
	B←B*2
⑤	写回B=200
	Commit
	Unlock B

封锁协议小结

	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
1级封锁协议		√			√		
2级封锁协议		√	√		√	√	
3级封锁协议		√		√	√	√	√

封锁技术有效解决了并发操作的一致性问题，但是也带来新的问题——活锁、死锁

活锁

- 事务T1封锁了数据R
- 事务T2又请求封锁R，于是T2等待。
- T3也请求封锁R，当T1释放了R上的封锁之后系统首先批准了T3的请求，T2仍然等待。
- T4又请求封锁R，当T3释放了R上的封锁之后系统又批准了T4的请求……

T2有可能永远等待，即活锁的情况。

T_1	T_2	T_3	T_4
lock R	.	.	.
.	lock R	.	.
.	等待	Lock R	.
Unlock	等待	.	Lock R
.	等待	Lock R	等待
.	等待	.	等待
.	等待	Unlock	等待
.	等待	.	Lock R
.	等待	.	.

活锁

■ **避免活锁**

先来先服务策略：多个事务请求封锁同一数据对象，系统按照请求封锁的先后顺序对事务进行排队，当数据对象上的锁释放，批准队列中的第一个事务获得锁。

T_1	T_2	T_3	T_4
lock R	.	.	.
.	lock R	.	.
.	等待	Lock R	.
Unlock	等待	.	Lock R
.	等待	Lock R	等待
.	等待	.	等待
.	等待	Unlock	等待
.	等待	.	Lock R
.	等待	.	.

死锁

- 事务T1封锁了数据R1
- T2封锁了数据R2
- T1又请求封锁R2，因T2已封锁了R2，于是T1等待T2释放R2上的锁
- 接着T2又申请封锁R1，因T1已封锁了R1，T2也只能等待T1释放R1上的锁
- 这样T1在等待T2，而T2又在等待T1，T1和T2两个事务永远不能结束，形成死锁

T1	T2
Xlock R_1	.
.	.
.	Xlock R_2
.	.
Xlock R_2	.
等待	Xlock R_1
等待	等待
等待	等待
.	.

预防死锁
诊断与解除死锁

死锁

- **预防死锁**

✓ 一次封锁法：每个事务必须将所有要使用的数据全部加锁,否则就不能继续执行。

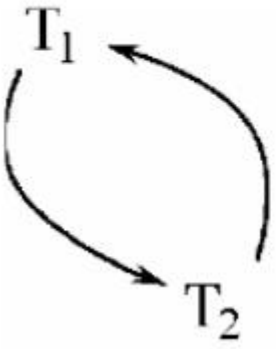
扩大了封锁的范围，从而降低了系统的并发度

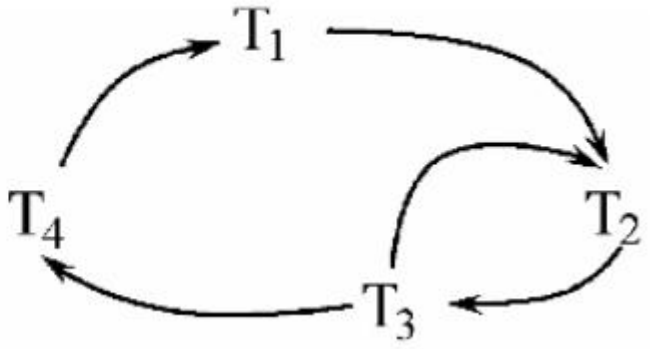
✓ 顺序封锁法：预先对数据对象规定一个封锁顺序，所有事务都按这个顺序实施封锁。

维护封锁顺序非常困难，成本很高

死锁

- **诊断死锁**
  - ✓ 等待图法：事务等待图是一个有向G=(T,U），T为结点的集合，每个结点表示正运行的事务；U为边的集合，每条边表示事务等待的情况。若T1等待T2，则在T1、T2之间画一条有向边，从T1指向T2。





图中存在回路→存在死锁

死锁

- **解除死锁**
  - ✓ 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。

    对撤销的事务所执行的数据修改操作必须加以恢复。