# 数据库系统原理

教程：数据库系统概论（第5版）

结合：CMU 15-445/645 INTRO TO DATABASE SYSTEMS

华中科技大学 计算机学院

左琼

# 第十章　数据库恢复技术

*Principles of Database Systems*

# 第十章 数据库恢复技术

# 10.1 事务的基本概念

- 1. 事务（Transaction）
- 定义：
  - 事务是用户定义的一组数据库操作，这组操作要么都做，要么都不做，不可分割。
  - 恢复和并发控制的基本单位

- 事务和程序比较：
  - 关系数据库事务：一个/一组SQL语句，或一段程序。
  - 一个程序通常包含多个事务

事务——transaction——交易、买卖

# 10.1 事务的基本概念

- 定义事务
- 显式定义方式：

```
BEGIN TRANSACTION
    SQL 语句1
    SQL 语句1
    ……
COMMIT WORK / ROLLBACK WORK
```

- 隐式方式：

  当用户没有显式地定义事务时，DBMS按缺省规定自动划分事务。

  某种环境或者程序中的一条SQL语句，应用程序或操作窗口退出。

# 10.1 事务的基本概念

例：银行转帐事务：从A帐户过户￥50到B帐户

```
begin transaction
        read(A);
        A := A – 50;
        write(A);
        if (A<0) then rollback;
        read(B);
        B := B + 50;
        write(B);
commit;
```

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

# 10.1 事务的基本概念

事务的特性（ACID特性）：

- 原子性（Atomicity）

  事务中包含的所有操作要么全做，要么全不做。

- 一致性（Consistency）

  事务的执行必须是将数据库从一个一致性状态转换到另一个一致性状态。

  一致性状态指数据库中只包含成功事务提交的结果，没有夭折事务残留的修改。

- 隔离性（Isolation）

  事务的执行不受其它并发执行事务的影响。

  对任何一对事务T1，T2，在T1看来，T2要么在T1开始之前已经结束，要么在T1完成之后再开始执行。

- 持久性（Durability）

  一个事务一旦提交之后，它对数据库的影响必须是永久的。

# 10.1 事务的基本概念

- 隔离性例：

Begin transaction

R(A)

U(A):A=A-1

Commit;

Begin transaction

R(A)

U(A):A=A-3

Commit

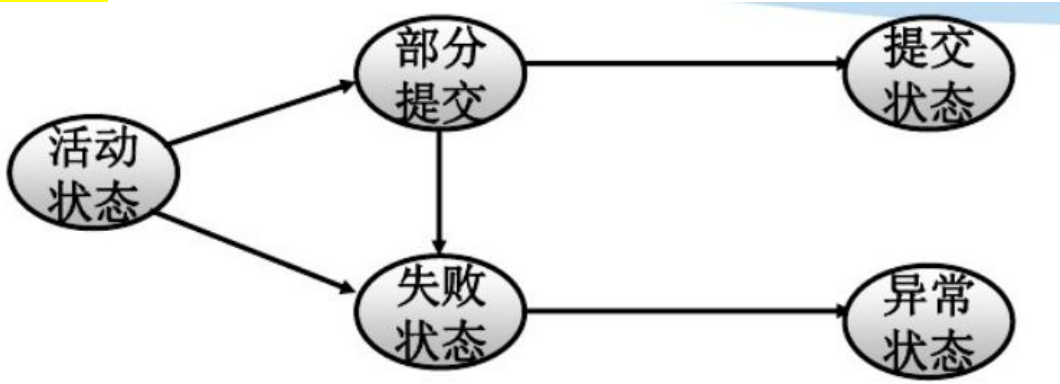
| T1 | T2 |
| --- | --- |
| (1) R: A=16 | |
| (2) | R: A=16 |
| (3) A=A-1<br>写回 A＝15 | |
| (4) | A=A-3<br>写回A＝13 |

事务1:

```
begin tran
   declare @sl  int
   select @sl=a from sales where id='A0001'
   waitfor delay '00:00:30.000'
   update sales set a=@sl-1 where id ='A0001'
comit tran
```

# 10.1 事务的基本概念

- 事务的两段提交：




- 对活动事务：事务执行对数据库的读写操作，写操作并非立即写磁盘，而是放在系统缓冲区中；直到事务正常提交，永久存入DB；否则，若事务失败，DBMS撤销它对DB和其他事务的影响，恢复到事务执行前的状态。
- 对异常状态事务，依据失败原因处理：
  - 外因（硬件故障、软件错误等）：重启该事务；
  - 内因（事务内部错误），废除该事务，报错。

# ACID特性带来的DBMS技术需求

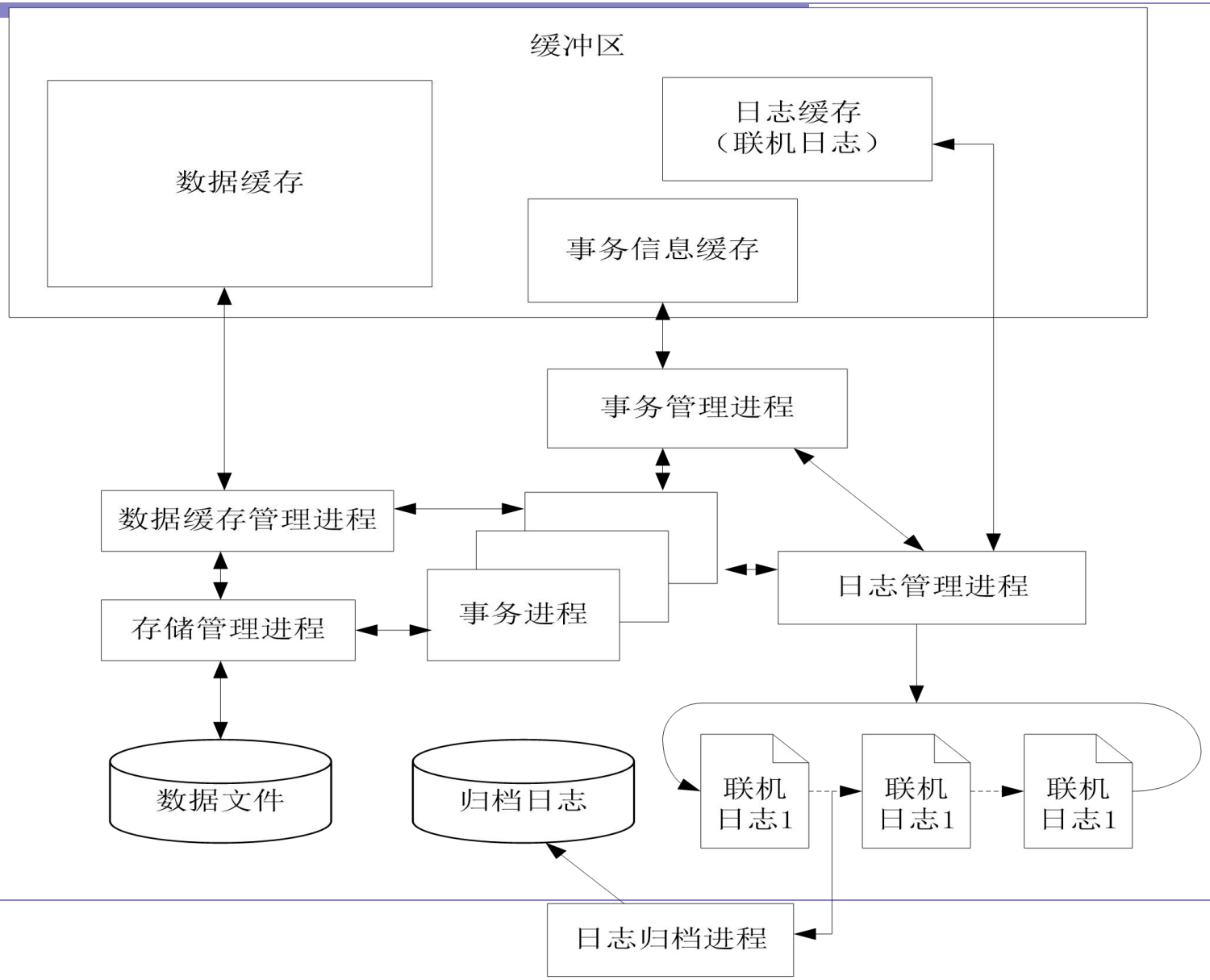
- 恢复:

  复杂系统如何保存数据（记录过程）、恢复策略（算法）、高可用性、其他技术手段

- 并发:

  锁、协议、标准

# DBMS的缓存与核心进程



# 10.2 数据库恢复概述

- 故障是不可避免的
  - 系统故障：计算机软、硬件故障
  - 人为故障：操作员失误、恶意破坏等，事务在运行过程中被强行停止。
- 数据库的恢复：

  把数据库从错误状态恢复到某一已知的正确状态(亦称为一致状态或完整状态)。保证事务*ACID*特性。
- DBMS恢复子系统占整个系统代码的10%+，是衡量系统优劣的重要指标。

- 问题复杂的原因：

  1）DBMS内部程序结构及程序执行的复杂性（多任务）；

  2）应用逻辑的语义需求

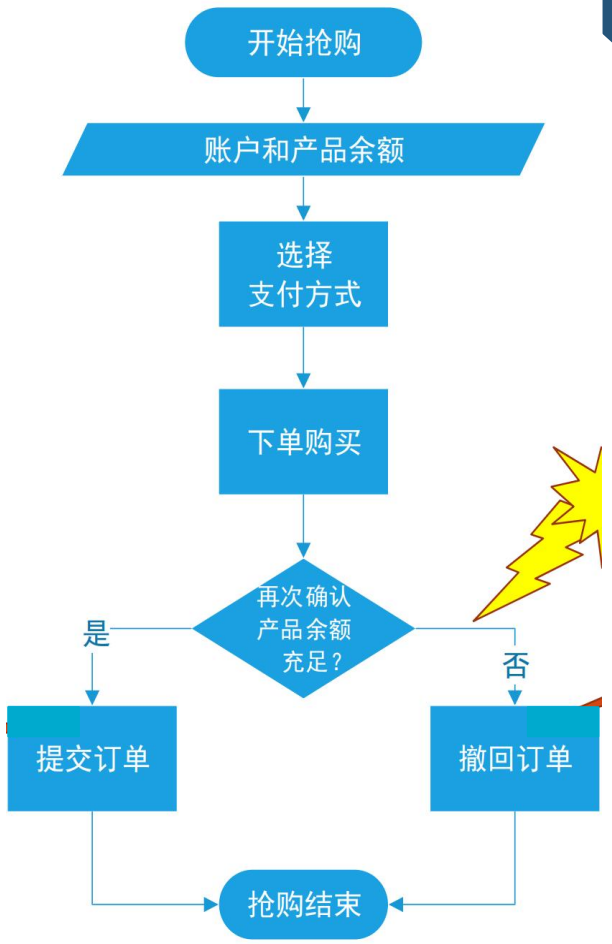
事务

# 10.3 故障的种类

故障种类：事务内部的故障、系统故障、介质故障、计算机病毒。

1. 事务内部故障

- 应用处理异常：有的可以通过事务程序本身发现的。如：转账事务、非法输入、找不到数据等。
- 有的是非预期的，不能由应用程序处理。如：运算溢出、死锁、事务超时、违反完整性限制等。
- 特点：
  - 非预期事务故障：事务没有达到预期的终点（COMMIT或ROLLBACK），事务夭折。夭折事务对数据库的部分修改可能已写入数据文件（数据库可能处于不正确、不一致状态）。
  - DBMS仍在正常运行
- 事务故障恢复：撤销事务（UNDO），DBMS自动完成




在线抢购示意图

# 10.3 故障的种类

例：事务内部故障

```
Create table t1 (
   a int,
   b  int check (b>2)
)
begin tran
   insert into t1 values(1,5)
   insert into t1 values(2,0)
commit
```

SET XACT_ABORT { ON | OFF }

- 当为OFF（默认）时：
  只回滚产生错误的SQL语句，而事务将继续进行处理；

- 当为ON时，
  如果SQL语句运行产生错误，整个事务将终止并回滚。

# 10.3 故障的种类

2. 系统故障：软故障（Soft Crash）

造成系统停止运转的任何事件，使得系统要重新启动。

- 软件故障（DBMS，OS，APS死机、蓝屏、意外重启、意外退出）；
- 操作失误；
- 特定类型的硬件错误（CPU、内存、主板等非外存储设备故障等）；
- 停掉电。

□ 特点：

- 整个系统的正常运行突然被破坏，DBMS不能正常运行；
- 所有正在运行的事务都非正常终止；
- 内存数据丢失或不再可靠；外存数据不受影响；
- DB处于不正确或不一致状态：一些尚未完成事务的结果可能已送入DB；已完成事务的结果可能部分还未送入DB；已完成事务的结果全部未送入DB（未及提交）。

# 10.3 故障的种类

- 发生系统故障时，事务未提交：
  - 恢复策略：强行撤消（UNDO）所有未完成事务

- 发生系统故障时，事务已提交，但缓冲区中的信息尚未完全写回到磁盘上：
  - 恢复策略：重做（REDO）所有已提交的事务

# 10.3 故障的种类

3. 介质故障：硬故障，外存发生故障。
   - 分类：磁盘损坏、磁头碰撞、强磁场干扰等
   - 特点：
     - 数据库遭到破坏，存在外存的数据部分丢失或全部丢失，正在执行的事务中断。
     - 发生可能性小
     - 破坏性最大
   - 介质故障恢复：
     - 装入数据库发生介质故障前某个时刻的数据副本
     - 重做REDO自此时始的所有成功事务，将这些事务已提交的结果重新记入数据库

# 10.3 数据恢复技术

- 计算机病毒
  - 一种人为的故障或破坏，是一些恶作剧者研制的一种计算机程序
  - 可以繁殖和传播
- 危害
  - 消耗资源（内存、磁盘、网络端口），破坏系统正常运行
  - 破坏、盗窃系统中的数据
  - 破坏系统文件
  - 数据库本身被破坏，或者DB正常，但数据可能不正确，系统不再可靠、可信

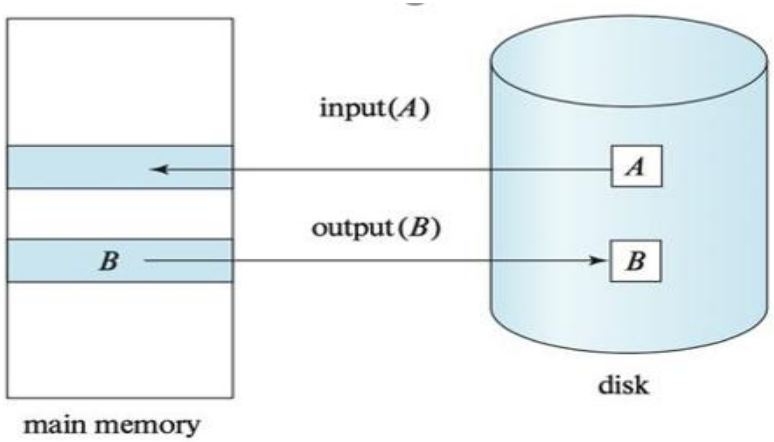
# 各类故障对数据库的影响

- 故障发生时数据可能不正确（事务的运行被恶意干扰或非正常终止）。
- 当涉及多个写磁盘（output）操作的事务出现故障时，如果只有数据本身的当前值状态，是无法判断哪些值是完成了相应的output操作的。需要借助系统的容错机制，找出不正确的数据，恢复正确的数据。
- 不同的故障影响的范围不同，采取的恢复策略也不尽相同。

  自动、人工启动、借助外部资源

- 数据文件本身可能被破坏

  需要去寻找可用的数据，重建系统状态。




- 恢复的基本原理：冗余（包括对于数据变化过程的记录）

# 10.4 恢复的实现技术

- 数据恢复技术的基本原理：冗余

  利用冗余，重建数据库，使其达到一致的状态。

- 恢复技术涉及的关键问题：
  - 如何建立数据的冗余数据？
    - 数据转储（backup）
    - 登记日志文件（logging）
  - 如何利用这些冗余数据实施数据库恢复？

# 10.4.1 数据转储（备份）

DBA定期地将整个数据库复制到磁带或另一个磁盘上保存起来的过程。备用的数据称为后备副本或后援副本。

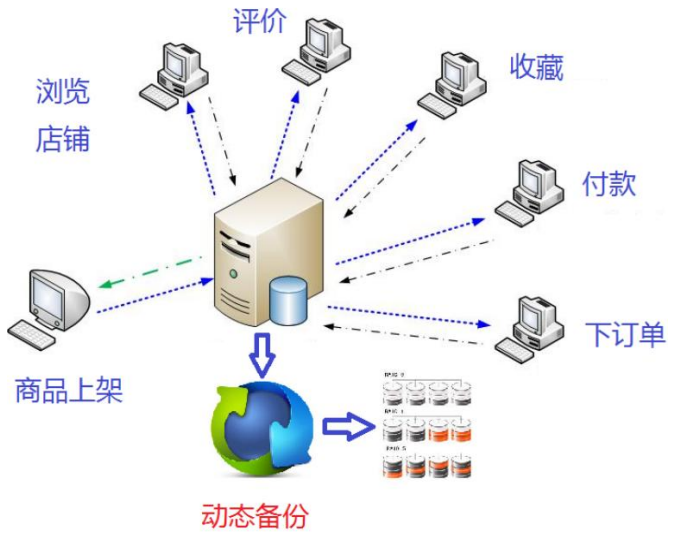
- 使用：
  - 数据库遭到破坏后可以将后备副本重新装入。
  - 重装后备副本只能将数据库恢复到转储时的状态。
- 转存方法：
  1. 静态转储与动态转储
  2. 海量转储与增量转储

# 1. 静态转储和动态转储

- 静态转储（dump）：
  - 在系统中无运行事务时进行的转储操作；
  - 转储开始时数据库处于一致性状态；
  - 转储期间不允许对数据库的任何存取、修改活动；
  - 得到的一定是一个数据一致性的副本。
- 优点：实现简单
- 缺点：停止了一切事务运行，降低了数据库的可用性。
  - 转储必须等待正运行的用户事务结束；
  - 新的事务必须等转储结束。

# 1. 静态转储和动态转储

- 动态转储：转储操作与用户事务并发进行，转储期间允许对数据库进行存取或修改。
- 优点：不用等待正在运行的用户事务结束，不会影响新事务的运行。
- 缺点：不能保证副本中的数据正确有效。
- 例：在转储期间的某个时刻*Tc*，系统把数据A=100转储到磁带上，而在下一时刻*Td*，某一事务将A改为200。转储结束后，后备副本上的A已是过时的数据了。
- 解决：
  - 需要把动态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；
  - 后备副本加上日志文件才能把数据库恢复到某一时刻的正确状态。



# 2. 海量转储与增量转储

- 海量转储：定期或不定期将DB全部数据转储。

  优点：简单；

  缺点：重复转储，转储量大；停止运行（多为静态转储）。

- 增量转储（incremental clumping）：只转储上次转储后更新过的数据

  优点：备份量小；

  缺点：恢复过程较复杂。

- 海量转储与增量转储比较：
  - 从恢复角度看，使用海量转储得到的后备副本进行恢复往往更方便；
  - 但如果数据库很大，事务处理又十分频繁，则增量转储方式更实用更有效。

# 日志模式

- 日志记录方式（Logging）
  - 事务对数据库更新操作的文件。
- 影子拷贝方式（Shadow Paging）：即数据每次修改都通过Copy-on-Write的方式进行。在更新数据时，复制一份原数据的副本并在副本上进行更新，最后通过用副本替换原始数据的方式完成操作。
  - 影子拷贝事务的原子性：依赖于对数据库指针（处于单个磁盘块/单个扇区）的写操作的原子性。
  - 影子页面：使用一个包含指向所有页面的指针的页表，页表的作用和数据库指针相同。影子拷贝只将页表和所有更新的页面拷贝到一个新位置，当提交事务时，原子性的更新指向页表的指针以指向新拷贝。
  - 影子页面的局限性：对并发事务支持较弱，在数据库中未广泛使用。

# 10.4.2 登记日志文件

1. 日志文件的格式和内容

日志log：记录事务对数据库的更新操作的文件。按事务操作执行时间顺序记日志（多个事务操作并发）。

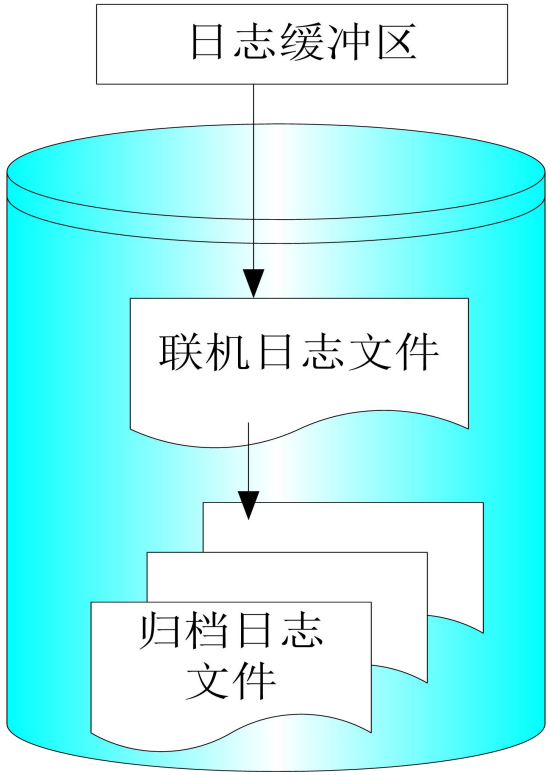
- 日志文件类型：
  1）记录为单位的日志文件；
  2）数据块为单位的日志文件。
- 记录为单位日志文件内容：
  1）事务开始 (BEGIN TRANSACTION) 标记（一个日志记录）；
  2）事务结束 (COMMIT或ROLLBACK) 标记（一个日志记录）；
  3）每个事务的所有更新操作（每个操作一个日志记录）。

# 联机日志文件和归档日志文件

日志文件可分为2类：

1）联机日志文件

大小有限，直接和DBMS日志缓冲区关联，保存数据库当前一段时间内的事务执行的变化过程，主要用于事务故障和系统故障。




2）归档日志文件

保存历史上所有的不再用于联机处理的日志记录，由联机日志文件归档而成，主要用于介质故障的恢复。

# 10.4.2 日志（Logging）

- 每个日志记录内容
  1）事务标识
  2）操作类型（插入、删除或修改）
  3）操作对象标识
  4）前像（BI）：更新前数据旧值
  5）后像（AI）：更新后数据新值

2. 以数据块为单位日志文件内容：事务标识+数据块
  1）数据块（整块）更新前内容
  2）数据块更新后内容

3. 日志的作用
  1）事务故障恢复和系统故障恢复
  2）协助后备副本进行介质故障恢复

# 日志类型

- **逻辑日志（Logical logging）**：记录高级别的事务操作，比物理日志耗费的存储空间小（一条逻辑日志可能涉及多个页面上的多个元组的更新）。
- 存在的问题：基于逻辑日志的恢复技术实现复杂，尤其日志包含了并发事务时，涉及原子性、并发正确性等问题。
- 例：登记日志文件

```
begin tran
   insert into sales
      values('A0002',0,10);
   update sales set b=b-3
      where id = 'A0001';
commit tran;
```

以记录为单位的日志文件：

```
<T0 start>
<T0, I, sales, null,
        ('A0002',0,10)>
<T0, U, sales,
        b(id='A0001'),30,27 >
<T0 commit>
```

| 序号 | 日志 | |
|---|---|---|
| 1 | T1 : 开始 | A=0,B=0,C=0 |
| 2 | T1 : 写A , A=10 | |
| 3 | T2 : 开始 | |
| 4 | T2 : 写B , B=9 | |
| 5 | T1 : 写C , C=11 | |
| 6 | T1 : 提交 | A=10,C=11 |
| 7 | T2 : 写C , C=13 | |
| 8 | T3 : 开始 | |
| 9 | T3 : 写A , A=8 | |
| 10 | T2 : 回滚 | C=11,B=0 |
| 11 | T3 : 写B , B=7 | |
| 12 | T4 : 开始 | |
| 13 | T3 : 提交 | A=8,B=7,C=11 |

# 日志类型

- **物理日志（Physical logging）**：记录字节级的数据库变化，一般以页面为单位，例如记录一个页面内的某个地址的数据记录。
- **存在的问题**：存储开销大。

| 类型 | 表空间ID | 页号 | 日志内容 |
|---|---|---|---|

| 类型 | 表空间ID | 页号 | 页面偏移量 | 数据 |
|---|---|---|---|---|

| 类型 | 表空间ID | 页号 | 页面偏移量 | 数据长度 | 数据 |
|---|---|---|---|---|---|

- **物理逻辑日志（Physiological logging）**：物理和逻辑两种日志技术的混合策略。物理层面以页面为单位（日志记录仅针对单个数据页面，对应内容不跨页），逻辑层面限于页面内部（例如记录某个数据页内某些槽（slot）的字节级变化。
- 原子性可控，存储开销不大，在现有的DBMS中较多使用。

| 事务1 | 块1 | 事务1 | 块1 |
|---|---|---|---|
| SNO=3,B=数据库,RK=NULL（更新前） | | SNO=3,B=数据库系统原理,RK=C（更新后） | |

| 事务2 | 块2 | 事务2 | 块2 |
|---|---|---|---|
| D=001,E=3（更新前） | | D=001,E=3.5（更新后） | |

| 事务1 | 块1 | 事务1 | 块1 |
|---|---|---|---|
| SNO=3,B=数据库系统原理,RK=C（更新前） | | NULL（更新后） | |

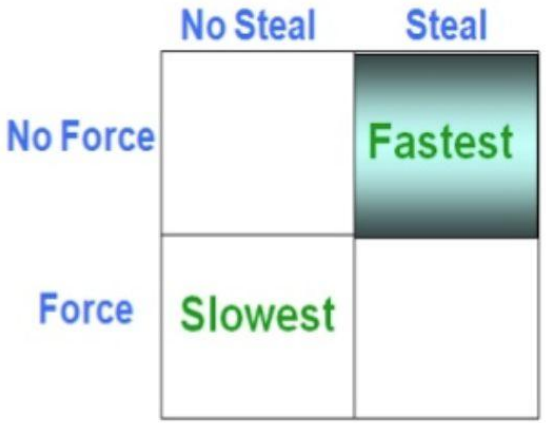
# 日志类型

- **abort日志**：当回滚事务Ti的所有undo操作都完成后，系统为该事务写一个<Ti abort>日志记录，表明撤销完成了，也对应事务的一种结束状态。
- 该机制可使得每个事务的undo过程至多完整执行一遍。
- 引入abort日志后，当发生系统崩溃后，扫描日志文件，当发现<Ti start>日志记录时：
  - 若未发现<Ti commit>，也没有<Ti abort>，则需要对该事务的所有日志记录执行撤销操作；
  - 若发现<Ti commit>或者<Ti abort>日志记录，都标识事务到达结束状态，都会对该事务的日志记录执行redo操作。

# 日志记录的使用

- REDO——将日志记录对应的操作执行一遍，运用数据的后像。
- UNDO——将日志记录对应的操作的逆操作执行一遍，运用数据的前像。
- 恢复 和 原子性：

- 强制写(Force): 事务完成之后，修改的 page 强制刷盘才能完成提交。
- 隐形(Steal): 事务结束前，随时允许脏页刷盘。





- 幂等性

每个日志记录的UNDO操作和REDO操作都具有幂等性，即无论重复执行多少次，效果等同于执行一次。

# 日志（Logging）

记录日志的方式：

- 在内存中开辟的临时保存日志记录的区域（日志缓冲区）；
- 根据需要一次将一个或多个缓冲块写入磁盘，从而减少写磁盘的次数。日志必须存储在稳定存储器上；
- 登记次序严格按并发事务执行的时间次序；写到磁盘中的日志记录顺序必须与写入日志缓冲区的次序完全一致；
- 先写日志后写数据库规则WAL （why？）

  - 在这两个操作之间可能发生故障
  - 如果先写了数据库修改，而在日志文件中没有登记下这个修改，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UNDO操作，并不会影响DB正确性

# 日志文件的单调递增性

- 日志文件是一个单调递增的文件

每个日志记录在日志中都有一个唯一的码，叫做日志序列号（Log Sequence Number，LSN）。

日志文件是按照LSN单调递增的顺序文件，如果操作A的日志记录在操作B的日志记录之后生成，则LSN(A)>LSN(B)。

- 对日志文件中的多条日志记录进行REDO，要按照日志序号(LSN)递增的顺序进行，即正向扫描日志文件进行REDO。
- 对日志文件中的多条日志记录进行UNDO，要按照日志序号(LSN)递减的顺序进行，即反向扫描日志文件进行UNDO。
- 当要对日志文件中的多条交错而无排列规律的日志记录进行REDO和UNDO操作时，只需各自按照上述规则进行，从基本原理上分析，REDO和UNDO之间原则上没有先后的要求，执行多次也没有影响。

# 写日志时机

- 先写日志协议实现技术：每个数据页面有一个字段记录最近版本对应的日志记录的LSN，写出该页面前，调用Log_flush内部函数将该LSN之前的所有日志记录写出。
- 提交时强制写日志协议force-log-at-commit——作为提交工作的一部分，必须强制写出该事务的所有日志记录。

  实现技术：调用Log_flush将该事务的commit日志记录及其之前的所有日志记录写出。
- 成组提交技术：当日志记录产生较为频繁时，凑齐一组（一个日志缓存页面）的日志记录才执行日志缓存写出操作，在此之前改组日志对应的事务提交均处于等待状态。