# 第8章 数据库的管理

数据库的管理/保护：DBMS（数据库管理系统）对DB（数据库）的监控。

**实现：数据库的恢复、并发控制、完整性控制、安全性控制。**

DBS的最小逻辑单位：事务。

## 8.1 事务的概念

### 8.1.1 事务的定义

定义8.1 事务是构成**单一逻辑工作单元的操作集合**。DBS必须保证事务**正确，完整的执行**。

# 银行转账事务  
T:BEGIN TRANSACTION; /\* 事务开始语句 \*/  
 read(A);  
 A := A - 50;  
 write(A);  
 if (A < 0): ROLLBACK; /\* 事务回退语句 \*/  
 else {  
 read(B);  
 B := B + 50;  
 write(B);  
 COMMIT;} /\* 事务提交语句 \*/

COMMIT语句：事务执行成功，“提交”，所有更新都写入磁盘。

ROLLBACK语句：事务执行不成功，“回退”，撤销所有更新，数据库恢复初始状态。

read(X)：把数据X从磁盘的数据库，读到内存的缓冲区。

write(X)：把数据X从内存的缓冲区，写入磁盘的数据库。

**注意：write操作不一定导致数据立即写回磁盘。很可能先暂存在内存缓冲区，稍后再写回磁盘。**

### 8.1.2 事务的ACID性质

1. **原子性** Atomicity：不可分割。要么全部执行，要么一个不做。 [ 事务管理子系统 ]
2. **一致性** Consistency：一个事务独立执行的结果，应保持数据库的一致性。 [ 完整性子系统 测试 ]
3. **隔离性** Isolation：并发时，事务串行，不受其他数据干扰。 [ 并发控制子系统 ]
4. **持久性** Durability：正确的事务对数据库的更新应该永久保留。 [ 恢复管理子系统 ]

## 8.2 数据库的恢复

### 8.2.1 恢复的定义原则和方法

1. 恢复的定义

* **恢复管理子系统**采取一些列的措施保证在任何情况下**保持事务的原子性和持久性**，确保**数据不丢失、不破坏。**
* 定义 8.2 **数据库的可恢复性**：系统能把数据库从破坏、不正确的状态，恢复到最近一个正确状态。

1. 恢复的基本原则和实现方法
   1. 转储和建立日志
      * 周期地对整个数据库复制。
      * 建立日志数据库。
   2. 发送故障时，分情况处理
      * 数据库破坏：复制最近一次数据到数据库，利用日志“重做”。
      * 数据库完好，数据损坏：利用日志“撤销”处理。

### 8.2.2 故障类型和恢复方法

软故障：系统故障；硬故障：介质故障

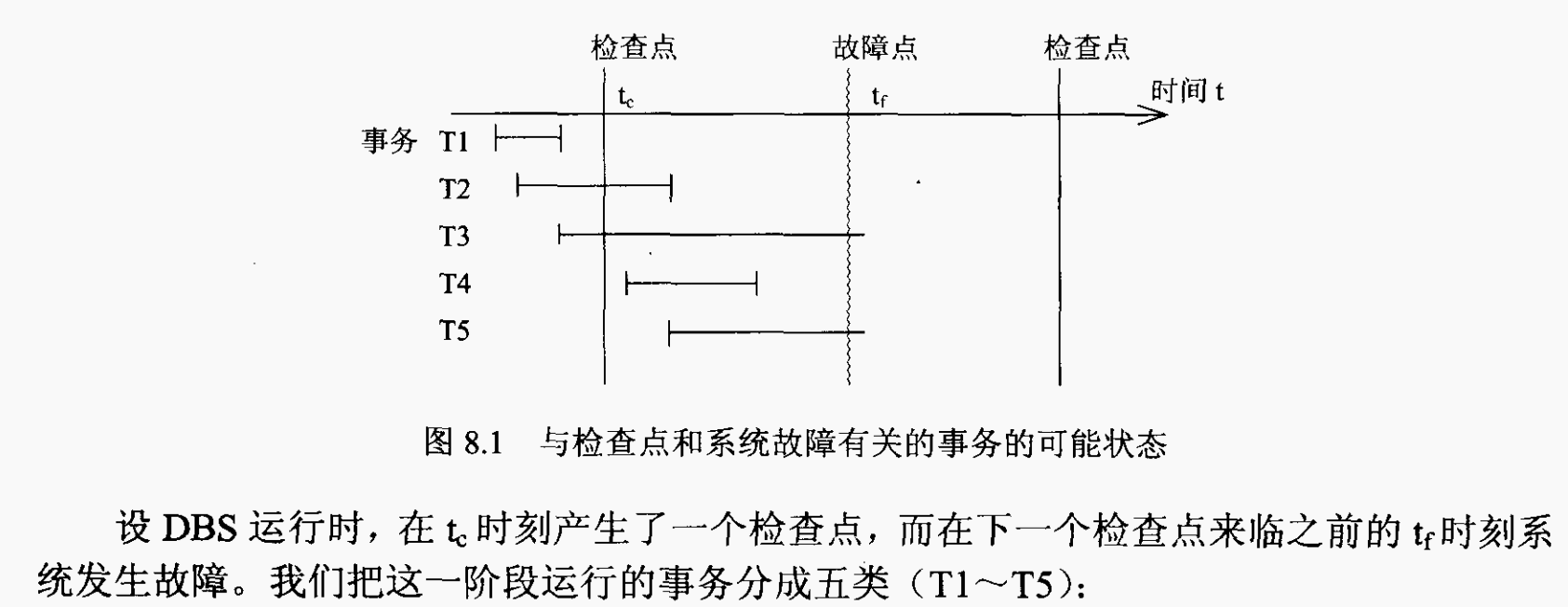
1. 事务故障：
   1. 可预期的：程序编写ROLLBACK 语句。
   2. 不可预期的：系统进行UNDO处理。
2. 系统故障：
   1. 定义：引起系统停止运转随之要求重新启动的事件。
   2. 特点：**正在运行的事务有影响，内存只能够数据丢失，不破坏数据库。**
   3. 重启时：1）对未完成事务进程UNDO处理；2）对已提交事务但更新还在缓冲区的事务进行REDO处理。
3. 介质故障：
   1. 特点：物理损坏，磁盘数据丢失。
   2. 恢复：
      * 重装转储的副本，恢复数据库到正确状态。
      * 在日志中找出转储以后所有已提交的事务。
      * 对已提交事务REDO处理。

### 8.2.3 检查点机制

1. 检查点方法

DBMS定时设置检查点。在检查点时刻才真正把数据写回磁盘，并在日志文件中写入检查点信息。

当DB需要恢复时，仅需恢复检查点之后还在执行的事务。



T1 : 已经完成，并且写入磁盘，无需操作。

T2 和 T4：已经完成，没有写回，需要REDO。

T3 和 T5：还未完成，需要UNDO。

1. 恢复算法
   1. 根据日志文件建立事务重做队列和事务撤销队列。
      * 重做队列：正向扫描日志，添加故障前**COMMIT的事务**。
      * 撤销队列：正向扫描日志，添加故障前**未COMMIT的事务**。
   2. 对重做队列进行REDO，对撤销队列进程UNDO。
      * REDO：正向扫描日志，根据重做队列进行REDO。
      * UNDO：**反向**扫描日志，根据撤销队列进行UNDO（逆操作）。

### 8.2.4 运行记录优先原则

1. 至少要等相应运行记录（日志记录）已经写入运行日志文件后，才能允许事务往数据库中写记录；**“先日志，后数据库”**
2. 直至事务的所有运行记录（日志记录）都已经写入运行日志文件后，才能允许事务完成COMMIT处理。**“写全日志，才COMMIT”**

### 8.2.5 SQL对事务的支持

## 8.3 数据库的并发控制

### 8.3.1 三个问题

1. 丢失更新问题
2. 读“脏”数据：”脏数据“，未提交并随后撤销的数据。
3. 不一致分析问题 / 幻行

### 8.3.2 封锁机制

1. 排它锁 = X锁 = 写锁
   1. **定义8.3 加X锁后，不允许再对该数据加其他任何锁。**
   2. PX协议：
      * 更新前，执行 "XFIND R"，获取X锁；
      * 未获取，一直等待，直到获取。
   3. PXC协议：
      * PX协议 + 在 COMMIT / ROLLBACK 时，解除X锁。
2. 共享锁 = S 锁 = 读锁
   1. **定义 8.4 加S锁后，仍允许再加S锁。但在S锁全解除之前，不允许添加X锁。**
   2. PS协议：
      * 更新前，执行 "SFIND R"，获取S锁；
      * 若更新，执行 "UPDX R", 升级X锁；
      * 无X锁，一直等待，直到获取。
   3. PSC协议：
      * PS协议 + 在 COMMIT / ROLLBACK 时，解除X锁。
3. 封锁的相容矩阵

### 8.3.3 活锁、饿死和死锁

1. 活锁问题
   1. **定义 8.5 活锁：某个事务一直处于等待状态，得不到封锁的机会。**
   2. 解决：队列，FIFO
2. 饿死问题
   1. **定义 8.6 饿死：对某数据，S锁永远存在，没有机会上X锁。**
   2. 解决：授权加锁：T2对Q加锁
      * 不存在在数据项Q上持有X锁的其他事务。
      * 不存在等待对数据项Q加锁且先于T2申请加锁的事务。
3. 死锁问题
   1. **定义 8.7 死锁：相互等待。**
   2. 解决：事务依赖图（有向图）消除环
      * 死锁测试程序：每隔一段时间检查并发的事务之间是否发生死锁。

### 8.3.4 并发调度的可串行化

1. 概念 （ 定义8.8 ）
   1. **调度：事务的执行次序。**
   2. **串行调度：多个事务依次执行。**
   3. **并发调度：利用分时方法，同时处理多个事务。**
2. 可串行化概念
   1. **定义 8.9 每个事务中，语句的先后顺序在各种调度中始终保持一致。**
   2. 可串行化的调度：并发调度的结果 = 某一串行调度的结果
3. 两段锁
   1. 作用：确保可串行，并发度降低。
   2. 扩展阶段（ 第一阶段 ）：获得封锁
   3. 收缩阶段（ 第二阶段 ）：释放封锁，ROLLBACK / COMMIT 阶段。
   4. **满足 “ 两段锁 ” ==> 可串行**

### 8.3.5 SQL中事务的存取模式和隔离级别

## 8.4 数据库的完整性

### 8.4.1 完整性子系统和完整性规则

1. 概念
   1. **数据库的完整性**：数据的 **正确性、有效性、相容性**
      * 正确性：数据的合法性，e.g. 数值类型中只含数字，不含字母。
      * 有效性：数据是否属于定义的有效范围。
      * 相容性：同一事实的两个数据应相同。
   2. **完整性检查**：检查数据库中数据是否满足规定条件。
   3. **完整性约束条件**：数据库中数据应满足的条件。
   4. **完整性子系统**：DBMS中执行完整性检查的子系统。
2. 功能
   1. 监督事务执行，测试是否违法完整性规则。
   2. 有违反现象，进行处理。
3. 完整性规则的组成
   1. 触发条件：什么时候检查。
   2. 约束条件/谓词：要检查什么。
   3. ELSE子句：查出错误，怎么办。

### 8.4.2 SQL中的完整性约束

1. 域约束

* CREATE DOMAIN COLOR CHAR(6) DEFAULT '???' # 若插入COLOR不再范围内，则置为???  
   CONSTRAINT VALID\_COLORS # 域约束名  
   CHECK(VALUE IN ('RED', 'YELLOW', 'BLUE', 'GREENN', '???'))

1. 基本表约束

* \* 都可以加 CONSTRAINT 约束名 # 指定域约束名
  1. 候选键的定义
  + UNIQUE(<列名序列>) # 候选键，仅表示唯一，非空需要加 NOT NULL。  
    PRIMARY KEY(<列名序列>) # 主键，仅有一个，默认非空。
  1. 外键的定义
  + FOREIGN KEY(<列名序列>)  
     REFERENCES<参照表>[<列名序列>]  
     [ON DELETE<参照动作>]  
     [ON UPDATE<参照动作>]
  + 参照动作：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）
  1. "检查约束"的定义
  + CHECK (weight >= 80 and (color = 'red' and weight <= 200) or (color != 'red' and weight <= 400))
  + **Check子句只对定义它的关系起作用。**在哪张表里面定义，只对该表起作用。
  + 例：在SPJ表中用CHECK子句定义S.SNO外键，删除S表中的SNO，并不会检查SPJ表中的CHECK语句。
  + 检查子句中的条件，尽可能不要涉及其他关系。

1. 断言
   1. 适用：约束与多个关系有关 / 与聚合操作有关。
   2. 触发时刻：所有修改都会触发，所有使断言为假的操作都被拒绝。
   * CREATE ASSERTION <断言名> CHECK (<条件>) # 定义  
     DROP ASSERTION <断言名> # 撤销，不提供RESTRICT和CASCADE

### 8.4.3 SQL-3 的触发器

1. 触发器结构
   1. **定义 8.10 触发器 / 主动规则 / 时间—条件—动作规则 是一个能有系统自动执行对数据库修改的语句。**
   2. 组成：ECA规则
      1. 事件：插入，删除，修改等操作。
      2. 条件：判断测试条件是否成立。
      3. 动作：满足条件，DBMS执行具体的动作。
2. SQL-3触发器实例

* CREATE TRIGGER TRIG1 # 触发器名：TRIG1  
  AFTER UPDATE OF PRICE ON SPJ # 触发事件：修改SPJ中的PRICE后激活  
  REFERENCING  
   OLD AS OLDTUPLE # 修改前元组  
   NEW AS NEWTUPLE # 修改后元组  
  WHEN (OLDTUPLE.PRICE > NEWTUPLE.PRICE) # 条件部分  
   UPDATE SPJ  
   SET PRICE = OLDTUPLE.PRICE  
   WHERE SNO = NEWTUPLE.SNO AND PNO = NEWTUPLE.PNO AND JNO = NEWTUPLE.JNO  
  FROM EACH ROW; # 对每一个修改的元组，都检查
* 撤销语句：DROP TRIGGER TRIG1

1. 触发器的结构组成
   1. 时间关键字：AFTER, BEFORE, INSTEAD OF
   2. 触发事件： UPDATE ( OF <属性表> ), DELETE, INSERT
   3. 动作条件： WHEN 定义
   4. 动作体： BEGIN ATOMIC ... END
   5. UPDATE： OLD AS ,NEW AS ; DELETE: OLD AS; INSERT: NEW AS
   6. 触发器种类：元祖级触发器 FROM EACH ROW，语句级触发器 FOR EACH STATEMENT(默认)

## 8.5 数据库的安全性

安全性 vs 完整性：前者，**非法，蓄意**；后者，**合法，无意**。

### 8.5.1 安全性级别

1. 环境级、职员级、OS级、网络级、DBS级

### 8.5.2 权限

1. 权限：用户使用数据库的方式。
2. 访问DB的权限：读 ，插入，修改，删除 权限
3. 修改DB的权限：索引，资源（创建新关系），修改，撤销 权限
4. 存储控制的类别
   1. 自主存储权限 DAC：创建者有控制权，可以通过授权传递权限。
   2. 强制存取控制 MAC：【安全性更高】
      1. 主体：活动实体（用户，进程等）； 客体（文件，表，索引等）。
      2. 敏感度标记：绝密，机密，可信，公开。
         1. 主体：许可证级别；客体：密级。
         2. 读：许可证级别 >= 密级
         3. 修改：许可证级别 = 密级

### 8.5.3 SQL中的安全性机制

1. 视图
   1. 用户只能使用视图中定义的数据，不能使用定义之外的数据。
   2. **优点：数据安全性，逻辑独立性，操作简便性**。
2. 用户权限及其操作
   1. 用户权限：SELECT INSERT DELETE UPDATE REFERENCES USAGE（允许使用已定义的域）
   2. 授权语句：
   * GRANT <权限表> ON <数据库元素> TO <用户名表> [WITH GRANT OPTION]  
       
     # 所有权限：ALL PRIVILEGES  
     # 数据库元素：关系，视图，域（域名前加 DOMAIN）  
     # WITH GRANT OPTION: 可以传递权限
   1. 回收语句
   * REVOKE <权限表> ON <数据库元素> FROM <用户名表> [RESTRICT|CASCADE] # 回收权限  
     REVOKE GRANT OPTION FOR ... # 回收转授出去的转让权限

### 8.5.4 数据加密

### 8.5.5 自然环境的安全性