ch1:

四个基本概念:

- 数据:是数据库中存储的基本对象,是描述事物的符号记录,可以是文字、图形、图像、声音,数据与其语义是不可分的
- 数据库:是长期储存在计算机内、有组织的、可共享的大量数据的集合,基本特征:
  - 按一定的数据模型组织、描述和储存
  - 可为各种用户共享
  - 冗余度较小
  - 数据独立性较高
  - 易扩展
- 数据库管理系统(DBMS):一种操作和管理数据库的大型软件,用于建立,使用和维护数据库,保证数据库的安全性和完整性,并发控制,数据库恢复,主要功能:
  - 数据定义:提供数据定义语言(DDL)
  - 数据操作:提供数据操作语言(DML)
  - 数据库的事务和运行管理
  - 数据库的建立和维护功能
- 数据库系统:是指在计算机系统中引入数据库后的系统构成,在不引起混淆的情况下常常把数据库系统简称为数据库,构成:
  - 数据库
  - 数据库管理系统(及其开发工具)
  - 应用系统
  - 数据库管理员和用户

数据管理:

- 对数据进行分类、组织、编码、存储、检索和维护,是数据处理的中心问题

		人工管理阶段	文件系统阶段	数据库系统阶段
背景	应用背景	科学计算	科学计算、管理	大规模管理
	硬件背景	无直接存取存储设备	磁盘、磁鼓	大容量磁备盘
	软件背景	没有操作系统	有文件系统	有数据库管理系统
	处理方式	批处理	联机实时处理、批处理	联机实时处理、分布处理、批处理
特点	数据的管理者	用户（程序员）	文件系统	数据库管理系统
	数据面向的对象	某一应用程序	某一应用	现实世界
	数据的共享程度	无共享，冗余度极大	共享性差，冗余度大	共享性高，冗余度小
	数据的独立性	不独立，完全依赖于程序	独立性差	具有高度的物理独立性和一定的逻辑独立性
	数据的结构化	无结构	记录内有结构，整体无结构	整体结构化，用数据模型描述
	数据控制能力	应用程序自己控制	应用程序自己控制	由数据库管理系统提供数据安全性、完整性、并发控制和恢复能力

数据库系统的特点:

- 数据结构化:数据结构化是数据库主要特征;整体结构化;数据库中实现的是数据的真正结构化
- 数据的共享性高,冗余度低,易扩充
- 数据独立性高:物理独立性,逻辑独立性,由 DBMS 的二级映像功能来保证
- 数据由 DBMS 统一管理和控制

数据库中用数据模型来抽象,表示和处理数据和信息,分为概念层数据模型和组织层数据模型(包括逻辑模型和物理模型)
数据模型的组成三要素:数据结构,数据操作和完整性约束
实体:客观存在并可相互区别的事物
属性:实体所具有的某一特性称为属性，一个实体可以由若干个属性来刻画
码:码是唯一标识实体的属性
域:属性的取值范围称为域
实体型:对实体所有固有特性的描述
实体值:实体型的一个实例
实体集:同型实体的集合
E-R 图:

- 实体:矩形框
- 属性:椭圆框
- 联系:菱形框

常用的数据模型:

- 非关系模型
  - 层次模型:有且只有一个结点没有双亲结点,这个结点称为根结点,根以外的其它结点有且只有一个双亲结点
  - 网状模型:网状数据库系统采用网状模型作为数据的组织方式
- 关系模型:
  - 在用户观点下，关系模型中数据的逻辑结构是一张二维表，它由行和列组成
  - 关系必须是规范化的，满足一定的规范条件;最基本的规范条件:关系的每一个分量必须是一个不可分的数据项
  - 实体完整性(具有唯一性标识);引用完整性(建立外键约束);域完整性(对属性的取值做强制性约束)
- 面向对象模型
- 对象关系模型

三级模式包含外模式, 模式, 内模式

- 外模式:也叫做子模式或者用户模式, 是数据库用户(应用程序员和终端用户)看见的结构. 一个数据库可以有多个外模式
- 模式:也叫逻辑模式, 是全体数据的逻辑结构和特征描述(其实就是数据库结构, 不包含数据; 数据库某时刻的状态是模式的实例), 是中间层, 不涉及数据的存储细节, 也和具体的应用程序等无关.
- 内模式:也称存储模式,和数据的物理结构,存储方式有关.一个数据库只有一个内模式

二级映像:在三层结构之间做 map.外模式和模式之间的称为外模式/模式映像,模式和内模式之间的称为模式/内模式映像
模式改变的时候, 相应地改变外模式/模式映像, 于是外模式保持不变, 应用程序不需要修改, 保证了数据的逻辑独立性
存储结构改变时, 相应地改变模式/内模式映像, 于是模式不变, 应用程序不需要修改, 保证了数据的物理独立性

ch2:

笛卡尔积:两个集合的所有可能的有序对的集合
关系:笛卡尔积子集合,记作$R(D_1, D_2, ...D_m)$
关系模式:关系的描述,记作R(属性 U, 域 D, 映射关系 dom, 依赖关系 F)
三类关系表:基本表(实际存在的表),查询表,视图表

关系基本操作:选择、投影、并、差、笛卡尔积是查询操作的基本操作,其他操作可以由这 5 种基本操作推出
关系数据库语言的分类:关系代数语言,关系演算语言,结构化查询语言
关系的三类完整性约束:

- 实体完整性(主码,行),实体完整性规则:若 A 是 R 的主属性,则 A 不取空
- 引用完整性(外码,表间):设 F 是关系 R 的一组属性,但不是关系 R 的码.如果 F 与基本关系 S 的主码K_s相对应,则称 F 是基本关系 R 的外码,R 称为参照关系,S 称为被参照关系或目标关系
- 域完整性(属性值的合法性约束,列)

关系代数:
集合运算:

- 并:$\cup$
- 差:—
- 交$\cap$
- 笛卡儿积$\times$
- 选择σ:$\sigma_{F(R)}$表示从 R 中选择满足条件(使逻辑表达式 F 为真)的元组,是对行的运算
- 投影π:$\pi_{A_1,A_2,...,A_n}(R)$表示从 R 中选择属性 A1,A2,...,An 的值,是对列的运算,投影的结果中要去掉相同的行
- 改名ρ:$\rho_{(A_1,A_2,...,A_n)(R)}$表示将关系 R 的属性改名为 A1,A2,...,An,可得到具有不同名字的同一关系
- 连串:若 r = $(r_1,...,r_n), s = (s_1,...,s_m)$,则定义 r 与 s 的连串为$\hat{rs} = (r_1,...,r_n,s_1,...,s_m)$
- θ连接:$R \bowtie_{A\theta B} S = \sigma_{(r[A]\theta r[B])}(R \times S)$
- 自然连接(等值连接分为 R.A 和 S.A 的两列合并为 A):$R \bowtie S = \sigma_{R[A]=S[B]}(R \times S)$,此时舍去的元组叫悬浮元组

内连接只能获得两个表有联系的记录,外连接保留一个表的所有数据,另一个连接无关联的记录为 null,外连接的形式:左外连接、右外连接、全外连接

- 赋值运算:临时关系变量$\leftarrow$关系代数表达式
- 除$\div$:给定关系$R(X,Y) \quad S(Y,Z)$,其中 XYZ 为属性组.R 与 S 的除运算得到一个新的关系 P(X),记作$R \div S = \left\{ \mathrm{tr}[X] \mid \mathrm{tr} \in R \wedge \pi_{Y(S) \subseteq Y_x} \right\}$,其中$Y_x$表示 x 在 R 中的象集,x=tr[X]
- 聚集函数:
  - sum 求和
    - avg 平均值
    - count 计数
    - max 最大值
    - min 最小值
- 创建视图:create view view_name as 查询表达式
- 分组:$_{\text{属性下标}}G$ 聚集函数$_{\text{属性下标}}(\text{关系})$
  - 如求每位学生的总成绩和平均成绩:$_{\mathrm{S\#}}G\ \mathrm{sum}_{\mathrm{SCORE}},\mathrm{avg}_{\mathrm{SCORE}}(\mathrm{SC})$

ch3:

SQL 特点:

- 综合统一(一体化)
- 面向集合的操作方式
- 高度非过程化两种使用方式
- 统一的语法结构
- 语言简洁,易学易用

默认数据库:
master 数据库:记录所有系统级别信息,用于控制用户数据库和数据操作
msdb 数据库:记录任务计划信息、事件处理信息、数据备份及恢复信息、警告及异常信息
model 数据库:为用户数据库提供的样板,每次创建一个新数据库时先制作一个 model 数据库的拷贝再扩展
tempdb 数据库:为临时表和其他临时工作提供了一个存储区
数据类型:

- 整数类型:int,smallint,tinyint,bigint
- 浮点类型:double,real,numeric(p,d)[定点数,小数点前 p 位后 q 位]
- 字符串类型:char(定长),varchar(变长)
- 日期时间类型:date,time

语句:

- 定义模式:`schema schema_name authorization user_name` 若没有指定 schema_name 则该项信息默认值为 user_name
- 删除模式:

  ```
  drop schema schema_name [cascade | restrict]
  ```

  若指定了 cascade 则删除模式下的所有对象,若指定了 restrict 则删除模式下的所有对象前必须先删除这些对象
- 创建表:`table table_name (column_name data_type [constraint],...)`
- 修改表:

  ```
  alter table table_name [add column_name data_type [constraint],... | drop
  column_name [cascade | restrict] | modify column_name data_type [constraint]]
  ```

- 删除表:

  ```
  drop table table_name [cascade | restrict]
  ```

- 建立索引:

  ```
  create [unique][cluster] index index_name on table_name (column_name [asc |
  desc],...)
  ```

建立聚簇索引时,索引顺序会与表中记录的物理顺序一致,在常查询而不更新的情况下建立聚簇索引,每个表只能有一个

- 删除索引:

  ```
  drop index index_name
  ```

- 数据查询:

  ```
  select [distinct(去重)|all] column_name [as alias,或者直接空格alias],... from
  table_name [where condition] [group by column_name,...] [having condition] [order
  by column_name [asc | desc],...]
  ```

  (顺序不能变)
  - where 子句常用查询条件:
    - 比较运算符

  - 逻辑运算符:and,or,not
  - 模糊查询:(not)like,如 `where name like '张%'`
  - 空值判断:is null,is not null(is 不能用=代替)
  - 范围查询:(not)between and,in(in('a1','a2','a3')), exists
  - 转义符:like 中使用转义符时,如 `where name like '张\%' escape '\'`
- order by 子句:对查询结果按属性进行排序,默认升序,可指定降序,null 默认最大,默认升序
- group by 子句:对查询结果按属性进行分组,常与聚集函数一起使用

- 连接查询:
  - 等值连接:

    ```
    select table_name1.column_name,table_name2.column_name,... from
    table_name1,table_name2 where condition
    ```

  - 外连接:
    - 左外连接:`left out join table_name2 on condition`
    - 右外连接:`right out join table_name2 on condition`
  - 内连接:

    ```
    select column_name,... from table_name1 [inner] join table_name2 on condition
    ```

- 嵌套查询:
  - 一个 select-from-where 语句称为一个查询块,子查询不能使用 orderby
  - 可以用比较运算符+any 或者 all
  - exists 子查询:()中填一个子查询,如果子查询结果不为空则返回 true
- 集合查询:
  - union:合并两个查询的结果集,去重
  - union all:合并两个查询的结果集,不去重
  - intersect:返回两个查询的交集,去重
  - except:返回第一个查询中不在第二个查询中的记录,去重
- 基于派生表的查询:派生表是一个临时表,在查询中使用,可以在 from 子句中使用子查询来实现
- 数据操作:
  - 插入元组:

    ```
    insert into table_name (column_name,...) values (value,...)
    ```

  - 更新元组:

    ```
    update table_name set column_name=value,... [where condition]
    ```

    - 删除元组:

      ```
      delete from table_name [where condition]
      ```

- 视图:
  - 创建视图:

    ```
    create view view_name as select column_name,... from table_name [where condition]
    [with check option]
    ```

with check option 确保通过视图插入或更新的数据必须满足创建该视图时定义的 WHERE 子句

子查询中不允许含有 orderby 语句和 distinct 短语

- 删除视图:

```
drop view view_name cascade
```

ch4:

是否有系统安全保护措施是数据库系统主要的指标之一

数据库的不安全因素:

- 非授权用户对数据库的恶意存取和破坏
- 数据库中重要或敏感的数据被泄露
- 安全环境的脆弱性

安全标准简介:

- TCSEC 安全标准
- CC 标准
- TCSEC/TDI 标准的基本内容
  - 从四个方面来描述安全性级别划分的指标:安全策略,责任,保证,文档
  - C1 级,比萨斜塔:实现对用户和数据的分离,进行自主存取控制(DAC),保护或限制用户权限的传播,现有的商业系统稍作改进即可满足
  - C2 级,人口普查:安全产品的最低档次,提供受控的存取保护,将 C1 级的 DAC 进一步细化,以个人身份注册负责,并实施审计和资源隔离
  - B1 级,人口普查强制落实到户:标记安全保护,对标记的主体和客体实施强制存取控制(MAC),审计等安全机制
  - B2 级,形式主义:结构化保护,建立形式化的安全策略模型并对系统内的所有主体和客体实施 DAC 和 MAC
  - B3 级,形式主义但提供民众监控且可以重开:安全域该级的 TCB 必须满足访问监控器的要求,提供系统恢复过程
  - A1 级,走进基层:验证设计,提供 B3 级保护的同时给出系统的形式化设计说明和验证以确信各安全保护真正实现

DAC:

- C2 级
- 用户对不同的数据对象有不同的存取权限,不同的用户对同一对象也有不同的权限
- 用户可将其拥有的存取权限转授给其他用户

MAC:

- B1 级
- 每一个数据对象被标以一定的密级
- 每一个用户被授予某一个级别的许可证
- 主体是系统中的活动实体,客体是系统中的被动实体，受主体操纵
- 仅当主体许可证级别≥客体密级时，该主体才能读取相应的客体;仅当主体许可证级=于客体密级时，该主体才能写相应的客体

存取控制流程:

- 系统对访问请求用户进行身份鉴别
- 在 SQL 处理层进行自主存取控制和强制存取控制
- 对用户访问行为和系统关键操作进行审计,对异常用户行为进行简单入侵检测

用户身份鉴别:
- 用户标识号:由用户名和用户标识号构成,在整个生命周期内唯一静态
- 静态口令:一般由用户自己设定
- 动态口令:口令是动态变化的,采用一次一密的方法
- 生物特征鉴别
- 智能卡鉴别

存取控制子系统:定义用户权限+合法权限检查
- 授权:
  - 授予:`grant privilege_name on table table_name to user_name [with grant option]`,with grant option 表示可以将这些权限授予他人,不允许循环授权(user_name 填 public 是全部用户,对特定属性要指名,如 UPDATE(Sno))
  - 撤销:`privilege_name on table_name from user_name`,撤销时不能使用 with grant option

审计:
- 启用一个专用的审计日志,将用户对数据库的所有操作(服务器发生的事件,系统权限,语句事件,模式对象事件)记录在上面,C2 及以上
- 审计分类:用户级审计和系统级审计

AUDIT 语句和 NOAUDIT 语句:
- AUDIT 语句:启用审计功能,记录用户对数据库的操作
- [例]对修改 SC 表结构或修改 SC 表数据的操作进行审计:AUDIT ALTER,UPDATE ON SC

加密:
- 存储加密:
  - 透明存储加密:写道磁盘时加密,读取时解密,内核级
  - 非透明存储加密:多层
- 传输加密:
  - 链路加密:报文报头均加密
  - 端到端加密:加密报文不加密报头

其他安全性保护:
- 推理控制:防止用户通过查询结果推断出更高机密的信息
- 隐蔽信道
- 数据隐私保护

ch5:

数据库的完整性:数据的正确性+数据的相容性
完整性维护:提供定义完整性约束机制+提供完整性检查机制+提供违约处理机制

实体完整性定义:
- 在 CREATE TABLE 中用 PRIMARY KEY 定义
- 单属性做主键有两种定义:
  - 列级约束条件:

```
CREATE TABLE Student
(Sno  CHAR(9)  PRIMARY KEY,
Sname  CHAR(20) NOT NULL,
Ssex  CHAR(2),
SBirthdate  DATETIME,
SchoolID  CHAR(20))
```

  - 表级约束条件:

```
CREATE TABLE Student
        (Sno  CHAR(9),
         Sname  CHAR(20) NOT NULL,
         Ssex  CHAR(2),
      SBirthdate  DATETIME,
      SchoolID CHAR(10),
          PRIMARY KEY (Sno)
       )
```

- 多属性做主键:定义为表级约束条件,显然应该单独一行定义

实体完整性检查:插入行或对主码列进行更新时检查,检查主码唯一性与非空性
检查方法:全表扫描,十分浪费时间,自动建立索引可缓解

参照(引用)完整性定义:
- 在 CREATE TABLE 中用 FOREIGN KEY 定义
- 用 REFERENCES 短语指明参照哪些表的主码
- 在行级定义参照完整性:

```
CREATE TABLE SC
        ( Sno  CHAR(9)  NOT NULL REFERENCES Student(Sno),
         Cno  CHAR(4)  NOT NULL REFERENCES Course(Cno),
         Grade    SMALLINT,
          PRIMARY KEY (Sno, Cno))
```

- 在表级定义参照完整性:

```
CREATE TABLE SC
        ( Sno  CHAR(9)  NOT NULL,
         Cno  CHAR(4)  NOT NULL,
         Grade    SMALLINT,
          PRIMARY KEY (Sno, Cno),
          FOREIGN KEY (Sno) REFERENCES Student(Sno),
          FOREIGN KEY (Cno) REFERENCES Course(Cno)
     ) 参照完整性违约处理:拒绝执行,级联操作,设置为空值
CREATE TABLE SC
       (Sno   CHAR(9)  NOT NULL,
        Cno   CHAR(4)  NOT NULL,
        PRIMARY KEY(Sno,Cno),
        FOREIGN KEY (Sno) REFERENCES Student(Sno)
         ON DELETE CASCADE     /*级联删除 SC 表中相应的元组*/
         ON UPDATE CASCADE, /*级联更新 SC 表中相应的元组*/
        FOREIGN KEY (Cno) REFERENCES Course(Cno)
              ON DELETE NO ACTION
              /*拒绝删除*/
       )
```

用户自定义的完整性是
- 针对某一具体应用的数据必须满足的语义要求
- 属性上的约束条件,check 语句:sex CHAR(2) CHECK (sex IN('男','女'))
- 元组上的约束条件的定义:如 check(Birthdate < Enrolldate))

完整性约束命名子句:
CONSTRAINT constraint_name< 完整性约束条件名 >< 完整性约束条件 >:
包括 NOT NULL,UNIQUE,PRIMARY KEY,FOREIGN KEY,CHECK等
修改表中的完整性限制:ALTER TABLE Student DROP CONSTRAINT constraint_name

ch6:

数据依赖:

- 一个关系内部属性与属性之间的一种约束关系
- 函数依赖 FD:如$F=\{\text{Sno}\to\text{Sdept},(\text{Sno,Cno})\to\text{Grade}\}$,对于 R(U)任意一个可能的关系 r,r 中不可能存在在 X 属性相等而 Y 上属性不相等,记作 Y 依赖于 X,X→Y
  - $X\to Y$,但$Y\not\subseteq X$,则称 $X\to Y$ 是非平凡函数依赖
  - $X\to Y,Y\subseteq X$,则称 $X\to Y$ 是平凡函数依赖
  - $X\to Y$,对 X 的任意一个真子集 X'有$X'\not\to Y$,称$X\xrightarrow{F}Y$是完全函数依赖,否则为部分函数依赖,记作$X\xrightarrow{P}Y$
  - $X\to Y$,若$X\to Y(Y\not\subseteq X),Y\not\to X,Y\to Z,Z\not\subseteq Y$,则称$X\xrightarrow{\text{传递}}Z$,Z 对 X 传递函数依赖;若 X $\longleftrightarrow$Y,则 Z 直接依赖于 X
  - 存在问题:数据冗余,更新异常,插入异常,删除异常
  - 解决方案:规范化数据依赖
- 多值依赖 MVD

码:

- 设 K 为$R<U,F>$中的属性或属性组合。若$K\xrightarrow{F}U$,则 K 称为 R 的一个候选码,又称候选关键字(一生万物);若关系模式 R 有多个候选码,则选定其中的一个作为主码,又称主键
- 关系模式 R 中属性或者属性组 X 并非 R 的码却是另一个关系模式的码则称 X 是 R 的外码,主属性是候选码中任意一个属性
- 超键:超键是候选键的超集,允许冗余

范式:

- 关系模型 R 为第 n 范式可简记为$R\in$nNF
- 第一范式 1NF:关系的每一个分量必须是一个不可分的数据项
- 第二范式 2NF:关系模式 R 满足 1NF,且 R 中每一个非主属性都完全函数依赖与所有候选码,若不属于 2NF 可能会产生以下问题(插入异常,删除异常,修改复杂)
  - 1NF 到 2NF 的转换:找部分函数依赖的属性,将其提取出来形成一个新关系,原关系中删除该属性
- 第三范式 3NF:关系模式 R 满足 1NF,若 R 中不存在码 X,属性组 Y 和非主属性 Z,$Z\not\subseteq Y,s.t.\ X\longleftrightarrow Y,Y\not\to Z$成立,则$R\in$3NF(将 2NF 分解为多个 3NF 的关系后,不能完全消除各种异常)
- BCNF:每个属性都不部分依赖于候选码也不传递依赖于候选码;第三范式只是要求 R 为非主码属性不传递依赖于 R 的候选码,而 BCNF 则是对 R 的每个属性都做要求
  - 此即:如果 R∈3NF 且 R 只有一个候选码,则 R 必属于 BCNF

  规范化的本质是提高数据独立性，解决插入异常、删除异常、修改复杂、数据冗余等问题的方法。规范化的基本思想是逐步消除数据依赖中不合适的部分。
  - 第一范式的目标:确保每列的原子性
  - 第二范式的目标:确保表中的每列，都和候选键相关,要求解决掉所有非主属性的部分完全函数依赖问题
  - 第三范式的目标:确保每列都和候选键列直接相关,要求解决掉所有非主属性的传递依赖问题
  - BCNF 的目标:BCNF 的目标是消除 3NF 允许主属性对候选键的部分或传递依赖,解决不同的候选键属性组之间的两两的依赖问题

ch7:

数据库的设计过程:

- 需求分析:了解用户需求,该阶段是否充分准确决定了构建的速度和质量
- 概念结构设计:形成独立于具体数据库管理系统的概念模型(E-R 图)
- 逻辑结构设计:转化为 DBMS 支持的数据模型并优化
- 物理结构设计:选取存储结构于存取方法
- 数据库实施
- 数据库运行维护

E-R 图的集成:

- 合并 E-R 图可能出现属性冲突,命名冲突,结构冲突
  - 属性域冲突:即属性的类型,取值范围;属性单位不同
  - 命名冲突:同名异义,异名同义
  - 结构冲突:同一对象在不同应用中具有不同的抽象(把属性变换为实体或者把实体变换为属性);同一实体在不同子系统中 E-R 图所包含的属性排列方式不同(取并集后换序)
- 冗余消除:冗余指的是可以被基本数据导出的数据

ch8:

查询处理阶段及主要内容:

- 步骤:
  - 查询分析:词法分析,语法分析,语义分析,符号名转换
  - 查询检查:安全性检查,完整性检查
  - 查询优化:代数优化,物理优化
  - 查询执行
- 选择操作的实现方法:
  - 全表扫描
  - 索引扫描
- 连接操作的实现方法:
  - 嵌套循环方法
  - 排序-合并方法
  - Hash Join 方法
  - 索引连接方法
- 查询代数优化的一般准则:
  - 选择运算尽可能先做
  - 在执行连接操作前对关系进行预处理,排序、索引等
  - 投影运算和选择运算同时做
  - 把某些选择运算与笛卡尔积改为连接运算
  - 将投影运算与其前面或后面的双目运算结合
- 代数优化等价变换规则:
  - 规则 1-2　连接、笛卡尔积具有交换律和结合律
  - 规则 3　投影的串接定律:若A_i集合构成B_j集合的属性子集, 则$\pi_{A_1,\dots,A_n}\left(\pi_{B_1,,\dots,B_n}(E)\right) \equiv \pi_{A_1,\dots,A_n}(E)$
  - 规则 4　选择的串接定律: $\sigma_{F_1}\left(\sigma_{F_2}(E)\right) \equiv \sigma_{F_1 \wedge F_2}(E)$
  - 规则 5　选择与投影的交换律:
    - 假设:选择条件 F 只涉及属性$A_1, \dots, A_n$,$\sigma_F\left(\pi_{A_1,\dots,A_n}(E)\right) \equiv \pi_{A_1,\dots,A_n}(\sigma_F(E))$
    - 假设:F 中有不属$A_1, \dots, A_n$的属性$B_1, \dots, B_m$,$\pi_{A_1,\dots,A_n}(\sigma_F(E)) \equiv \pi_{A_1,\dots,A_n}\left(\sigma_F\left(\pi_{A_1,\dots,A_n,B_1\dots,B_m}(E)\right)\right)$
  - 规则 6　选择与笛卡尔积的交换律:

  - 假设:F 中涉及的属性都是 E1 中的属性,$\sigma_{F(E_1\times E_2)}=\sigma_{F(E_1)}\times E_2$
  - 假设:且F_1只涉及E_1中的属性,F_2只涉及E_2中的属性,$\sigma_{F(E_1\times E_2)}\equiv\sigma_{F_1}(E_1)\times\sigma_{F_2}(E_2)$
  - 假设:F=$F_1\wedge F_2$,F_1只涉及E_1中的属性,F_2涉及E_1和E_2两者的属性,$\sigma_{F(E_1\times E_2)}\equiv\sigma_{F_2}\left(\sigma_{F_1}(E_1)\times E_2\right)$
- 规则 7　选择与并的交换:
  - 假设:E_1,E_2有相同的属性名,$\sigma_F(E_1\cup E_2)\equiv\sigma_{F(E_1)}\cup\sigma_{F(E_2)}$
- 规则 8　选择与差运算的交换:
  - 假设:E_1与E_2有相同的属性名,$\sigma_F(E_1-E_2)\equiv\sigma_{F(E_1)}-\sigma_{F(E_2)}$
- 规则 9　投影与笛卡尔积的交换:
  - 假设:E_1和E_2是两个关系表达式,A1 An 是E_1的属性,B1 Bm 是E_2的属性,$\pi_{A_1,\dots,A_n,B_1,\dots,B_m}(E_1\times E_2)\equiv\pi_{A_1,\dots,A_n}(E_1)\times\pi_{B_1,\dots,B_m}(E_2)$
- 规则 10　投影与并的交换:
  - 假设:E_1和E_2有相同的属性名,$\pi_{A_1,\dots,A_n}(E_1\cup E_2)\equiv\pi_{A_1,\dots,A_n}(E_1)\cup\pi_{A_1,\dots,A_n}(E_2)$

- 关系代数表达式的优化算法:
  - 分解选择运算:利用规则 4
  - 通过交换选择运算,利用规则 4 8 将其尽可能移到叶端
  - 通过交换投影运算,利用规则 3,5,9,l0 将其尽可能移到叶端
  - 利用规则 3 5 合并串接的选择和投影,以便能同时执行或在一次扫描中完成
  - 每一双目运算($\bowtie,\times,\cup,-$)和它的所有直接祖先(σ,π)一组
- 物理优化:选择高效合理的操作算法或存取路径
  - 选择操作的启发式规则:
    - 小关系，使用全表顺序扫描
    - 大关系，启发式规则有:
      - 对于选择条件是主码＝值的查询，选择主码索引
      - 对于选择条件是非主属性＝值,非等值查询,范围查询，并且选择列上有索引,如果查询结果比例< 10%使用索引扫描方法否则还是使用全表顺序扫描
      - 有索引优先用索引,若用 or 析取或无索引全表扫
  - 在作连接运算时,若两个表(R1,R2)均无序,连接属性上也没有索引，则可以有下面几种查询计划:
    - 对两个表作排序预处理
    - 对 R1 在连接属性上建索引
    - 对 R2 在连接属性上建索引
    - 在 R1,R2 的连接属性上均建索引
  - 对不同的查询计划计算代价,选择代价最小的一个

ch9:

事务和程序是两个概念,一个应用程序通常包含多个事务
事务是恢复和并发控制的基本单位
事务的定义:

```
begin transaction;
sql statements
commit | rollback
```

事务的 ACID 特性:

- 原子性:事务是一个不可分割的工作单位,要么全部完成,要么全部不做
- 一致性:数据库的状态满足所有的完整性约束
- 隔离性:并发不影响事务的执行
- 持久性:事务一旦提交,其对数据库的修改是永久性的

数据库四类故障:

- 事务故障:未运行至正常终止点就夭折了
  - 需要强行回滚撤消事务清除该事务对数据库的所有修改
- 系统故障:整个系统的正常运行突然被破坏,内存中缓冲区的信息丢失;外部存储设备上的数据未受影响
  - 清除尚未完成的事务的所有修改,系统重新启动时,恢复程序要强行撤消所有未完成事务将缓冲区中已完成事务提交的结果写入数据库,重做所有已提交的事务
- 介质故障:存储在外存中的数据丢失
  - 装入数据库发生故障前某个时刻的数据副本;重做自此时始的所有成功事务
- 计算机病毒

恢复的基本原理:利用存储在系统其它地方的冗余数据来重建
恢复的实现技术:

- 建立冗余数据
  - 数据备份:
    - 静态备份:冷备份,备份期间不允许对数据库的活动
    - 动态备份:热备份,但不能保证副本中的数据正确有效
    - 完全备份:海量备份，每次备份全部数据库
    - 差异备份:增量备份
- 日志文件:以流水方式记录每个事务对数据库操作,必须先写日志再写数据库
  - 记录内容:各个事务的开始标记,结束标记,所有更新操作,更新前后数据,与事务有关的内部更新操作
- 事务故障的恢复:系统自动完成
  - 反向扫描文件日志查找该事务的更新操作
  - 对更新操作执行逆操作
  - 直至读到开始标记
- 系统故障的恢复:
  - 正向扫描日志文件:确定 Undo(故障发生时未提交的事务)与 Redo 队列(故障发生时已提交的事务)
  - 先对 Undo 队列事务进行 UNDO 处理:反向扫描日志文件,对每个 UNDO 事务的更新操作执行逆操作
  - 再对 Redo 队列事务进行 REDO 处理:正向扫描日志文件,对每个 REDO 事务重新执行登记的操作
- 检查点技术:
  - 检查点记录的内容:
    - 建立检查点时刻所有正在执行的事务清单
    - 这些事务最近一个日志记录的地址
    - 重新开始文件的内容
    - 记录各个检查点记录在日志文件中的地址

检查点（checkpoint）主要工作内容

① 将当前日志缓冲区中的所有日志记录写入磁盘的日志文件上。

② 在日志文件中写入一个检查点记录（定时或者事件触发）。

③ 将当前数据缓冲区的所有数据记录写入磁盘的数据库中。

④ 把检查点记录在日志文件中的地址写入一个重新开始文件。

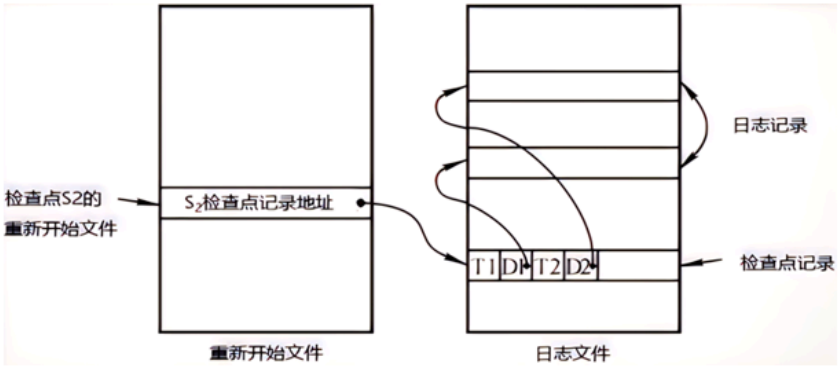
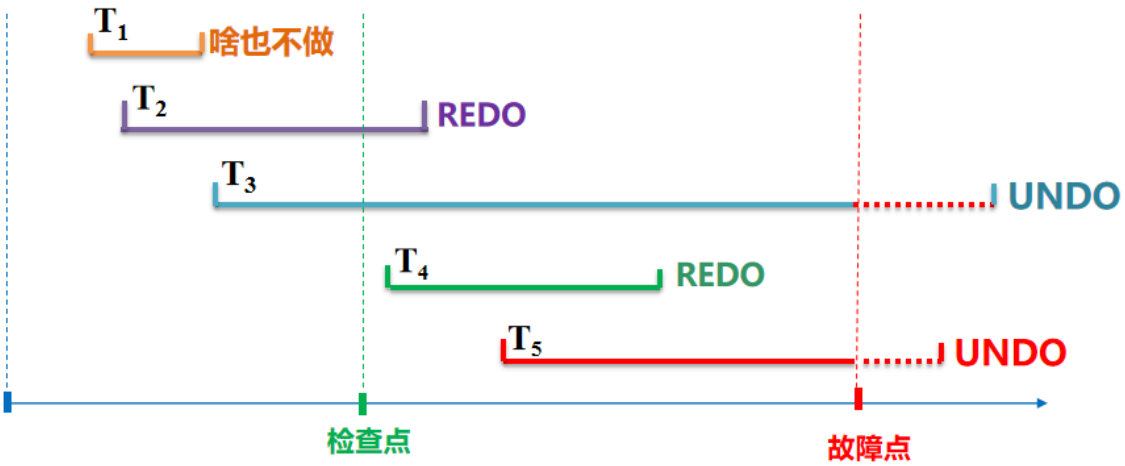


Figure 1: 检查点的构成



① 从重新开始文件中找到最后（最近）一个检查点记录

② 由该检查点得到REDO和UNDO事务队列

③ 对UNDO队列中的每个事务执行UNDO操作，对REDO中的每个事务执行REDO操作

Figure 2: 检查点策略

- 镜像技术:DBMS 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
- 备份方式:
  - 脱机物理备份

- 联机物理备份
- 逻辑备份

ch10:

并发操作带来的三种数据不一致性:

- 丢失修改:事务 A 与事务 B 从数据库中读入同一数据并修改,事务 B 的提交结果破坏了事务 A 提交的结果
- 脏读:A 事务读取 B 事务未提交的数据
- 不可重复读:A 事务读取 B 事务已提交的数据,但该数据在 B 事务中被修改了

锁的类型:

- 排他锁:X 锁,其他事务不可读不可写不可加锁吗,想写了
- 共享锁:S 锁,其它事务可读不可写,其他锁可以加 S 不能加 X,想读了
- 一级封锁协议(解决丢失修改):修改数据前需要对需加 X 锁直到事务结束释放
- 二级封锁协议(解决脏读):一级封锁协议+读取数据前须先加 S 锁,读后释放
- 三级封锁协议(解决不可重复读):二级封锁协议+读取数据前须先加 S 锁直到事务结束释放
- 活锁:相互谦让,循环消耗 CPU 但无进展
- 饥饿:一直被插队,不消耗 CPU
  - 解决方式:优先级调度;先来先服务
- 死锁:两个或多个事务互相等待对方释放锁获取数据
  - 死锁预防:
    - 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
    - 顺序封锁法:要求每个事务必须按照预先规定的顺序加锁
  - 死锁检测:
    - 超时法:可能误判
    - 等待图法:并发控制子系统周期性地检测事务等待图,若存在回路则出现死锁

并发调度的可串行性:

可串行化调度:几个事务的并行执行是正确的当且仅当其结果与按某一次序串行地执行它们时的结果相同

可串行性:正确并发调度应具备可串行性

冲突操作:不同事务对同一数据只有读读能交换

一个调度 Sc 在保证冲突操作的次序不变的情况下,通过交换两个事务不冲突操作的次序得到另一个调度 Sc',若 Sc'是串行的,称调度 Sc 与 Sc'等价,Sc 为冲突可串行化的调度

冲突可串行化的调度是可串行化调度的充分条件

两段锁:

- 第一阶段获得封锁,拓展阶段,给需要的数据加上锁,不能释放任何锁,与一次封锁法区别在于随着执行再逐步添加,直到开始收缩
- 第二阶段释放封锁,收缩阶段,只能释放锁,不能再加锁
- 遵守两段锁协议是可串行化调度的充分条件

封锁的粒度:并发控制时封锁对象的大小范围

对象:逻辑单元(属性,域,元组 etc.),物理单元(页,记录)

多粒度封锁:同时考虑开销与并发度(数据库→表→元组)←(多粒度树的分层)

高并发度需要细粒度锁

低锁开销需要粗粒度锁

多粒度封锁协议:

- 显式封锁:直接加锁

- 隐式封锁:上级被封锁因而连带

因此,对某个对象加锁检查冲突需要检查:
- 对数据对象本身
- 上下级

意向锁:
定义:若对一个结点加了意向锁则说明他的下级正在被加锁,提高对粗粒度对象加锁时的冲突检查效率
- 意向共享锁 IS 锁:表明后继结点加了 S 锁
- 意向排他锁 IX 锁:表明后继结点加了 X 锁
- SIX 锁:表示对对象加 S 锁再加上 IX 锁(要读整个表且改部分元组)

存在意向锁机制时,任何事务要对数据对象加锁申请的时候应该自上而下进行,释放反之