数据库系统简介

- 数据管理技术的驱动力
  - 数据管理的任务
    - 定义存储的对象
    - 实现加工流程的操作
  - 数据结构
    - 逻辑 / 物理
  - 数据结构 + 算法 = 程序
    - 问题：完成需求后继续改动，如果遗漏某种查询方式较好解决，如果需要更改对象的数据类型则较复杂
    - 良好的数据结构 + 糟糕的代码 >> 糟糕的数据结构 + 优秀的代码
  - **数据独立性**
    - 数据管理追寻的目标
    - 定义：当数据结构发生变化时，通过系统提供的映象（转换）功能，使应用程序不必改变
      - 数据的物理独立性：当数据存储结构发生变化时，使应用程序不必改变
      - 数据的逻辑独立性：当数据逻辑结构发生变化时，使应用程序不必改变
  - 开发信息系统需要的管理三大帮助
    - 数据定义：逻辑结构 / 物理结构
    - 数据操作：查询 / 更新
    - 数据约束：数据合理范围
  - **数据独立性的实现**：把方方面面的数据管理的功能放到一个统一的软件中，*将数据管理从应用程序中剥离出来*
  - 发展历史
    - 文件系统阶段
      - 提供了一定的物理独立性
      - 文件的结构是由应用程序所定义的，数据结构发生变化会引发较大问题
      - 不同部门创建自己的独立数据
        - 多副本，浪费空间
        - 产生数据的不一致，信息孤岛需要信息集成
          - 联邦数据库技术，编写不同部门文件转换程序，4个程序需要12个转换程序
- 数据模型
  - 数据库管理系统的基石
- 数据库模式
  - 数据库应用系统的基石

数据库系统简介

- （缺到第二章中间）
- 语言翻译处理层：（集合级操作）
- 存取层：事务，日志，封锁，存取路径，提供单元组接口（记录级操作）
- 存储层：把操作系统的信息格式化成数据页面
- 操作系统
- 数据字典：表的结构...

实体-联系模型

- 三阶段数据库设计过程
  - 需求分析（不属于此）
    - 确定存储哪些数据，建立哪些应用，常用的操作及对象有哪些等
  - 概念数据库设计
    - 对需求分析所得到数据的更高层的抽象描述：最核心、最耗时的阶段
    - ER模型 UML
  - 逻辑数据库设计
    - 将概念模型所描述的数据映射为某个特定的DBMS模式数据
    - 四阶段此处有实现阶段：
      - 选择数据类型、定义表、约束、触发器
  - 物理数据库设计
- ER模型：信息模型
  - 把er设计作为数据库设计的第一步，切忌跳过er
  - 实体的描述方式+联系的描述方式
    - 实体
      - **客观存在**并可以**相互区分**的事物
      - 描述：实体具有的属性
      - 域：属性的取值范围
      - 实体型：实体名+属性名集合
      - 实体集：同型实体的集合
    - 联系：
      - 实体之间的相互联系
      - 联系也可以有属性
      - 联系的元：参与联系的实体集个数
        - 学生与学生的班长的联系是一元联系
    - 图表示
      - 实体集矩形，属性椭圆，联系菱形

        - 实体集不能多次出现
        - ER图的连通性意味着什么？
            - 联通的节点存在某种关联
            - 数据库设计中，若er图不联通则可以直接放到不同的数据库中
        - 实体集，而非实体
- 实体的码
    - 超码
        - 能唯一标识实体的属性或属性组
    - 候选码
        - 其任意真子集都不能成为超码的最小超码
    - 主码
        - 从所有候选码中选定一个用来区别同一实体集中的不同实体
- 联系的码
    - 一对多，“多”可以当做码（学生-老师）
- er图中下划线作码
- 多种码
    - 替代码：除去主码之外的候选码
    - 代理码：人工码，只起唯一标识作用的序列号
    - 自然码：一个与行中属性有逻辑联系的候选码，它是实体的“真正的”属性（院系名称）
    - 智能码：经过编码的标识符（对某些属性编码，便于流程处理）
- 简单属性 复合属性
    - 简单属性：不可再分的属性
    - 复合属性：可以划分为更小的属性（电话号码=区号+本地号码）
    - 把相关属性聚集起来以反映更高层次的概念，可以使模型更清晰
- 单值/多值：某个特定属性在改属性上的取值是否唯一
    - 多值属性需要单独存放
- 派生属性：可以从其他相关的属性或实体派生出来的属性值（gpa）
    - 数据库中，一般只存基属性值，而派生属性只存其定义或依赖关系，用时再从基属性中计算出来
- 多值属性多椭圆表示，派生属性虚椭圆表示
- 空值（NULL属性）：表达当前“值未知”或“无意义、没有”
- NULL属性对数据库设计的影响
    - 如果不支持NULL，数据库就无法录入缺失部分信息的实体
    - 如果不支持NULL，数据库就无法将稍具差异的实体放入同一模式中
    - 判断逻辑变为3值逻辑，true，false，unknown
    - 实体完整性：主码不能为空
- 联系的种类
    - 一对一

        - 双箭头
        - 不是一一对应，某些实体可以不参与
    - 一对多
        - 用箭头或线段来表示联系的种类，箭头指向单方实体集
    - 多对多
        - 线段
- 多元联系
    - 菱形多出口
    - 多元联系中最多允许出现一个箭头，表明多方决定唯一目的实体
- 联系的势
    - 势表达了一个实体出现在联系中的次数（最少联系数，最多联系数）
    - 区分强制性和可选性联系
- 参与
    - 部分参与
    - 全部参与
        - 实体集E中的每个实体都参与到联系集R中的至少一个联系
        - 双线链接
- 角色
    - 一元联系中，当需要显式区分角色时，在连接菱形和矩形的线上加上说明性标注以区别不同的角色
- 存在依赖
    - x存在依赖于y：实体x的存在依赖于实体y的存在

- 扩展的er模型
    - 弱实体
        - 还款如何刻画?
            - 不能由还款号唯一标识，不能成为实体?
            - 不能重复贷款号属性
            - 当做属性，不能刻画支付了哪一笔还款
            - 解决方案：
                - 定义**弱实体**
                - 双层矩形，联系也是双层
                    - 标识性联系(identifying relationship)：弱实体集与其拥有者之间的联系弱实体集与强实体集之间是一对多的联系
                - 弱实体集的主码=强实体集的主码 + 弱实体集的分辨符
        - 何时使用弱实体?
            - 如果弱实体集不但参与和强实体集之间的标识性联系，而且参与和其它实体集的联系，或者弱实体集本身含有很多属性，则将其表述为弱实体集
    - 特化

        - 实体集中某些子集具有区别于该实体集内其它实体的特性，可以根据这些差异特性对实体集进行分组，这一分组的过程称作特化
        - 子类＝特例＝更小的实体集＝更多的属性
        - 父类 ➔ 子类
        - 倒三角表示
    - 概化
        - 各个实体集根据共有的性质，合成一个较高层的实体集。概化是一个高层实体集与若干个低层实体集之间的包含关系
        - 子类->父类
        - 概化 / 特化的问题
            - 高层实体集的属性被低层实体集自动继承
            - 格结构?
                - 双重父类->格结构
            - 概化中的成员身份
                - 两种成员身份
                    - 不相交的
                    - 有重叠的
            - 概化中的全部性约束
                - 全部的：每个高层实体必须属于一个低层实体集
                - 部分的：允许一些高层实体不属于任何低层实体集
            - 不想交+全部构成一种划分
    - 聚集
        - 三元表示问题
            - 职工-项目-机器
            - 直接写三元联系：有些职工并不和机器发生联系
            - 分为两个而元元素：损失哪个项目使用了什么机器的信息
            - 职工参加项目后才可能产生使用机器，“使用”应该指向“参加”
        - 把联系抽象成实体
    - 如何设计恰当的er图?
        - 抽象成实体还是属性
            - 实体有多方面性质，属性没有
            - 某个概念有唯一的码，应该抽象成实体
        - 属性分配到实体上还是联系上?
            - 不随联系改变的属性不放到联系上
        - 能否把多元联系转换为若干二元联系
            - 瓠瓜式通用转换方式
                - 可以，但没有意义
            - 能否用实体之间的二元联系替换三元联系

            - 不可以
            - 有损：重构的话会多出不存在的三元联系
            - 联系的属性：无处安放
        - 两个动作不太合适叠成三元联系
            - 两个动作的属性无法安放
            - 需要用聚集替换
- 概念数据库设计：
    - 局部设计-全局设计（消除局部er图的冲突（属性冲突 / 命名冲突））-全局优化（消除冗余）
- 逻辑数据库设计：
    - er图向关系模式转换
        - 实体->关系
        - 属性->关系的属性
    - 复合属性向关系模式的转换
        - 关系模型分量的取值具有原子性，符合属性只记录简单属性
    - 多值属性单独放到一个表里，避免冗余
    - 一对多联系向关系模式的转换
        - 将单方参与实体的码作为多方参与实体的属性
        - “多”的一方可以标识这个联系，因而可以在“多”的实体的表中放入
    - 多对多联系向关系模式的转换
        - 将联系定义为新的关系，属性为参与双方的码
    - 一对一联系向关系模式的转换
        - 都部分出现
            - 参与度低，新建一个表开销小
            - 参与度高，在原表中加一列
        - 联系一方全部参与
            - 将联系另一方的码作为全部参与一方的属性
    - 弱实体向关系模式的转换
        - 弱实体集所对应的关系的码由弱实体集本身的分辩符再加上所依赖的强实体集的码
    - 概括向关系模式的转换
        - 一般情况下，高层实体集和低层实体集分别转为表，低层实体集所对应的关系包括高层实体集的码
        - 如果概括是不相交并且是全部的，则可以不为高层实体集建立关系，低层实体集所对应的关系包括上层实体集的的所有属性
    - 聚集向关系模式的转换
        - 实体集A与B及其联系R被抽象成实体集C，C与另一实体集D构成联系S，则S的码由C和D的码构成
- 逆向工程：关系模式向ER的转换
    - 重合属性体现了实体之间的联系

- 如何对时态进行概念建模?
  - 概念模型反映该约束并不方便，需要借助数据库
- ER模型面向业务型应用，而非分析型应用(需要综合不同部门的数据)
- 底层业务处理 / 上层分析应用
  - 分析应用不契合ER模型

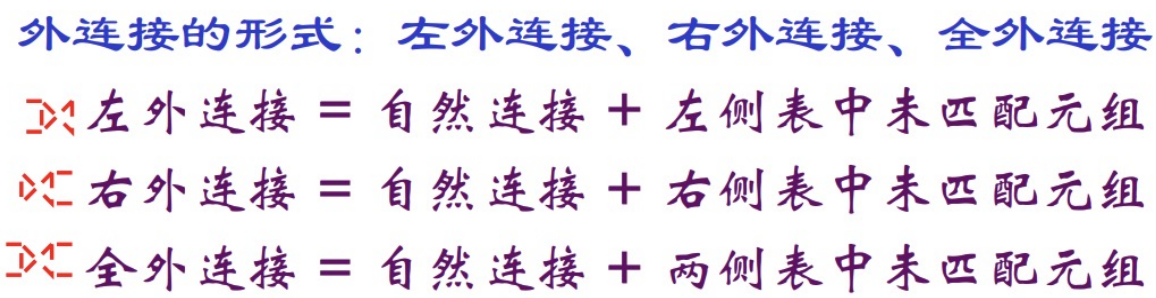
关系模型

- table
- 关系的基本概念
  - 笛卡尔积 -> 关系定义 -> 关系性质
- 笛卡尔积
  - 域
    - 具有相同数据类型的一组值的集合如整数集合、字符串集合、全体学生集合
  - 笛卡尔积定义在一组域上
  - 笛卡尔积的元素 $d1, d2, \dots, dn$称作n元组(tuple)
  - 若Di的基数为mi则笛卡尔积的基数为$\prod_{i=1}^{n} m_i$元组
  - 每一个值di称作分量（component）
  - 刻画可能的世界
- 关系：笛卡尔积的（有意义）子集
  - 什么是有意义?
    - 关系集合包含了真正存在的实体或发生的联系
    - 关系名体现了现实实体或联系
  - 关系的性质
    - 列是同质的，是同一类型的数据，即每一列中的分量来自同一域
    - 不同的列可以来自同一域，每列必须有不同的属性名
    - 行列的顺序无关紧要
    - 任意两个元组不能完全相同（集合内不能有相同的两个元素）
    - 每一分量必须是不可再分的数据，称其为作满足第一范式（1NF）的关系
      - 嵌套关系多元素方不能反查（声明性查询 vs 路径式查询）
- 关系模型三要素
  - 数据结构
    - 单一的数据结构——关系
      - 无需为不同数据结构提供不同的操作
    - 实体集、联系都表示成关系
    - 码
      - 候选码
        - 关系中的一个属性组，其值能唯一标识一个元组

        - 若从属性组中去掉任何一个属性，它就不具有这一性质了，这样的属性组称作候选码
    - 任何一个候选码中的属性称作主属性
    - 主码
        - 进行数据库设计时，从一个关系的多个候选码中选定一个作为主码
    - 外码
        - 关系R中的一个属性组，它不是R的（主）码，但它与另一个关系S的（主）码相对应（不一定属性名相同），称这个属性组为R的外码
        - 一个联系对应一个外码
            - 有联系一定会产生外码
        - 外码可以是自己的主码
- 总结
    - 关系模式包括
        - 关系名、关系中的属性名
        - 属性向域的映象，通常说明为属性的类型、长度等
        - 属性间的数据依赖关系，比如在特定的时间和教室只能安排一门课
    - 关系实例
        - 某一时刻对应某个关系模式的内容(元组的集合)
    - 关系数据库的构成
        - 关系数据库的型
            - 是关系模式的集合，即数据库描述。称作数据库的内涵(Intension)
        - 关系数据库的值
            - 是某一时刻关系的集合。称作数据库的外延(Extension)

- 数据完整性
    - 实体完整性
        - 关系的主码中的属性值不能为空值
    - 参照完整性
        - 外码对应的值必须是存在的或空值
    - 用户定义完整性
        - 用户针对具体应用环境定义的完整性约束条件
            - sno要求是8位整数，首位是0或1
            - 两个属性之间的约束关系
            - 某属性的约束
            - 跨表、跨数据库的约束
- 数据操作
    - 关系操作是集合操作操作的对象及结果都是集合，是一次一集合(Set-at-a-time)的方式
    - 非关系型（记录型）的数据操作方式是一次一记录(Record-at-a-time)
    - 关系数据语言的特点

        - 一体化：对象单一，都是关系，因此操作符也单一
        - 非过程化：用户只需提出“做什么”，无须说明“怎么做”存取路径的选择和操作过程由系统自动完成
        - 面向集合的存取方式：操作对象是一个或多个关系，结果是一个新的关系(一次一关系)非关系系统是一次一记录的方式
    - 抽象关系模型查询语言：
        - 关系代数：用预定义操作算子的执行序列来表达查询
        - 关系验算：用谓词来表达查询，只需描述所需信息的特性
            - 元组关系验算：谓词变元是元组
            - 域关系验算：谓词变元是属性列
- 关系代数运算
    - 基本
        - 一元
        - 多元
        - 具体运算
            - 选择运算σ
                - 在关系中选择满足给定条件的元组（行角度）
                - 选择条件
                    - F是选择的条件， $\forall t \in R$, $F(t)$要么为真，要么为假
                    - F由逻辑运算符连接算术表达式而成
                        - 逻辑表达式: $\wedge, \vee, \neg$
                        - 算数表达式：$X\theta Y$
                            - X, Y 是属性名、常量、或简单函数, θ是比较算符，$\theta \in \{>, \geq, <, \leq, =, \neq\}$
                        - 通常将过滤能力强或者佩戴索引的先执行
                            - 数据库中的查询优化器所做
            - 投影运算Π
                - 从关系中取若干列组成新的关系（从列的角度）
                - 需要去重
            - 更名ρ
                - 必要性
                    - 将更名运算施加到关系上，得到具有不同名字的同一关系
                    - 当同一关系多次参与同一运算时需要更名
            - 并
            - 差
            - （广义）笛卡尔积
                - 集合级运算
                - 把多表数据融合到一个表中

        - $R \times S = \{\hat{rs} | r \in R \wedge s \in S\}$
- 扩展
    - 交
        - 可以用差解决
    - θ连接
        - 先做笛卡尔积，然后再做theta运算
        - θ为算术比较符，为等号时称为等值连接
        - $R \overset{\bowtie}{_{A\theta B}} S$
    - 自然连接$\bowtie$
        - 和等值连接完全不同，θ连接不要求有重复的列(值相等即可)
        - 公共属性出现两表出现次数乘积次
        - 从两个关系的广义笛卡儿积中选取在相同属性列B上取值相等的元组，并去掉重复的列
        - 查询优化第一准则
            - 先做选择运算
        - 关于null的问题，需要有不同辩论
            - 数据库非外码关系时当distinct时会去重
            - 外码时数据库会抛弃，但按理取值范围一定，可能要匹配
        - 问题
            - 如果某表中没有某项会导致最终缺少行
            - 生成报表会出现问题
    - 外连接
        - 解决自然连接缺失的行
        - 外连接 = 自然连接 + 未匹配元组(悬挂元组)
        - 

        - 自然连接满足结合律，外连接+自然连接不满足
        - 外连接+外连接也不成立，两个全外连接也会不成立
            - 全外连接的反例：(A, B), (B, C), (A, C)
    - 半连接
        - 给出R里可以参与与S自然连接的行
        - 应用场景：减少分布式查询中的通讯量
            - 两个表在两个节点
                - 方案1:迁移左表到右节点

        - 方案2:右表对应的属性->左节点->半连接结果发到右节点
- 反半连接 anti join
    - 给出R里不能参与与S自然连接的行
    - 底层的物理操作符，用于not in / not exist
- 外部并 outer union
    - 两个表结构不一定完全相同，按公共属性相同连接起来(有些可以指明公共属性组的一部分)
    - 从不同数据源汇聚用户画像
        - 实体识别：识别是否是同一个用户
    - 和外连接结果相同，初衷不同
- 除法
    - 象集Z_x(Z，X是属性组，x是X的取值)
        - 选择+投影：从R中选出在X上取值为x的元组，只留Z属性
    - 求选修了全部课程的学生
        - $\{u|r \in SC \wedge u = r[name] \wedge C \subseteq COURSE_u\}$
        - $\{u|u \in \Pi_{name}(SC) \wedge \forall v \in C(\hat{uv} \in SC)\}$(先把姓名和目标笛卡尔积再看是不是都在原始表里)
    - 定义表达式
        - $R(X,Y) \div S(Y) = \{x|r \in R \wedge x = r[X] \wedge S \subseteq Y_x\} = \{u|u \in \Pi_X(R) \wedge \forall v \in S(\hat{uv} \in R)\}$
    - 计算表达式
        - $R(X,Y) = \Pi_X(R) - \Pi_X(\Pi_X(R) \times \Pi_Y(S) - R)$
        - 投影同学(所有可能的选课情形 - 实际发生的选课关系) = 所有没选完所有课程的同学，减去这些同学即得答案
            - 减法操作区别不同性质子集的作用
        - 下面的除法是错的

- SC(sno, cno, grade)

  求选修了所有课程的学生

  方案1：$\prod_{sno,cno}(SC) \div \prod_{cno}(C)$

  方案2：$\prod_{sno}(SC \div \prod_{cno}(C))$

  - 这个除法要求姓名+成绩相同才能被除出来

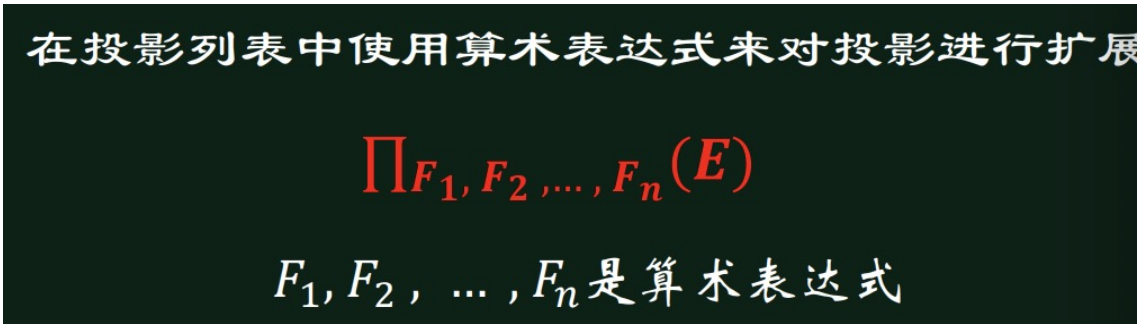
- 关系代数更新运算
  - 赋值运算
    - 临时关系变量 <- 关系代数表达式
    - $$R \div S = \prod_X(R) - \prod_X(\prod_X(R) \times \prod_Y(S) - R)$$

      $$temp1 \leftarrow \prod_X(R)$$

      $$temp2 \leftarrow \prod_X(temp1 \times \prod_Y(S) - R)$$

      $$result \leftarrow temp1 - temp2$$
  - 删除
    - 作差

- 插入
  - 并
- 更新
  - 广义投影(不明白为什么放在这里)
    - 

  - 利用广义同影改变元组的某些属性上的值

- 关系代数查询实例
  - 在树的结构中，查询所有祖父(递归查询)，是无法用关系代数操作完成，实际数据库会使用递归查询完成
  - 最低成绩查询：要基于现有性质完成查询