**Chapter 01：引言**

**-----------------------------------------------------------**

* 用文件系统处理数据信息的主要弊端：
  + 数据的冗余和不一致性
  + 完整性问题（一致性约束）
  + 原子性问题（事务要么处理完全要么完全不处理）
  + 并发访问异常
* 数据模型：
  + **关系模型（表）**
  + **实体-联系模型（E-R图）**
  + 半结构化数据模型
  + 基于对象的数据模型
* 数据抽象（从底层向上）：
  + 物理层：描述数据物理上是怎样存储的
  + 逻辑层：描述数据库中存储什么数据及其联系
  + 视图层：描述整个数据库的某一部分，使人机交互简单（有多个视图）
* 实例：特定时刻存储在数据库中的信息集合
* 数据库模式：数据库的总体设计（物理模式、逻辑模式、子模式）
* DDL（数据定义语言）：
  + 作用：说明数据库系统所采用的存储结构和访问方式
  + 内容：
    - 域约束：约束每个属性的取值范围
    - 引用完整性：确保一个关系中给定属性集上的取值也在另一关系中的某一属性集中出现
    - 授权：对不同用户在数据库中的不同数据值上允许不同的访问类型
* DML（数据操纵语言）：
  + 作用：使得用户可以访问或操纵数据
  + 类型：过程化DML（指明用户如何获取数据）、声明式DML（不用指明用户如何获取数据）
  + 内容：对数据库信息进行增、删、改、查
* SQL（结构化查询语言）：
  + DDL语句：如create table
  + DML语句**（声明式）**：如select...from...
* 数据库系统部件：
  + 存储管理器：负责底层数据与应用程序和查询之间提供接口
  + 查询处理器：DDL解释器、DML编译器、查询执行引擎
  + 事务管理部件：并发控制管理器、恢复管理器
* 事务管理：
  + 事务：数据库应用中完成单一逻辑功能的操作集合
  + 要求：原子性、一致性、持久性
* 数据库应用体系结构：

|  |  |
| --- | --- |
| **两层体系结构** | **三层体系结构** |
| **用户**  **应用程序**  **数据库系统** | **用户**  **应用客户**  **应用服务器**  **数据库系统** |
| 通过安装有特定应用程序的端与数据库交互  **例：邮局、服装店的管理应用**  **分散、不易升级** | 用户自己可以用客户端（前端）操作，通过应用服务器处理  **例：各种手机APP**  **易于对应用程序升级改造** |

**Chapter 02：关系模型介绍**

**-----------------------------------------------------------**

* 表（关系型数据库组成单位）：
  + 元组：表的一行
  + 属性：表的一列
    - 域：属性的取值范围
* 码（区分给定关系中的不同元组）：
  + 超码：一个或多个属性集合，可以唯一确定一个元组
  + 候选码：所有真子集都不是超码的超码（最小超码）
  + 主码：设计者选中的来区分元组的**候选码**
  + 外码：关系r1的属性A取值范围必须是关系r2的属性B的取值范围，**则A是从r1引用r2的外码**（r1是**引用关系**，r2是**被引用关系**，这样的约束要求叫**引用完整性约束**）
* 模式图：
  + 框：表示一个关系
    - 关系名：用灰色显示在顶部
    - 属性：逐个列出。主码属性用下划线表示
  + 箭头：
    - 单箭头：表示外码约束。从**引用关系**的**外码**属性指向**被引用关系**的**主码**属性
    - 双箭头：表示非外码约束的引用完整性约束
* 关系代数（输入为一个或多个关系，输出为一个关系）：
  + 选择
    - 作用：选择满足给定谓词的元组
    - 用法：
    - 说明：多个谓词可用连接词交、并、非连接成多个谓词
  + 投影
    - 作用：返回关系，但滤掉了特定属性
    - 用法：
    - 说明：下标依次列出需要选出的属性（泛化版本中允许是包含属性的表达式，如γ/12等）
  + 笛卡儿积
    - 作用：结合来自任意两个关系的信息
    - 用法：
    - 说明：运算仅将两个关系中的信息两两拼接成一个元组
  + 连接
    - 作用：将两个关系中的信息关联在一起
    - 用法：（等价于）
    - 说明：注意重名属性。predicate为空时做自然连接
  + 集合运算
    - 并：
    - 交：
    - 差：
  + 赋值：
    - 作用：将关系表达式的一部分赋给关系变量
    - 用法：
    - 说明：该运算不会向用户展示任何关系
  + 更名：
    - 作用：给关系及其属性更名
    - 用法：、
    - 说明：name是给关系更名，An是给第n个属性更名
  + 聚集运算
    - 作用：对一定范围内的值进行运算并返回一个结果
    - 用法：
    - 说明：前面的是分组的属性，所有属性相同则在一组。γ后funcN是各个聚集函数，每个as后面是聚集结果的名称
* 等价查询：关系代数表示不同但结果相同的查询。对于等价查询，数据库系统中的**查询优化器**会查看结果，**并找到高效方式求解**，而非严格按照关系代数的计算步骤进行

**Chapter 03：SQL介绍**

**-----------------------------------------------------------**

* 基本类型：
  + char(n)：固定长度n的字符串
  + varchar(n)：最大长度n的可变长度字符串
  + int：整数
  + smallint：小整数
  + numeric(p, d)：定点小数，共p位，小数点后有d位
  + real：浮点数
  + double precision：双精度浮点数
  + float(n)：精度至少为n位的浮点数
* 关系定义：
  + 语法：

create table r

(A1 D1 ,

A2 D2 ,

......

An Dn ,

<完整性约束1>,

......

<完整性约束k>);

* 说明：
* 关系名：r
* 属性名：An
* 属性类型：Dn
* 完整性约束：
* 主码声明：primary key (A1, A2, ..., Aj)
* 外码声明：foreign key (A1, A2, ..., Ak) references s（取值范围为s主码属性的范围）
* 非空约束：not null（在属性类型之后）
* 关系删除：
  + 语法：
    - drop table r;
    - delete from r;
  + 说明：
    - drop不但删除所有的元组，还将r从数据库中删除
    - delete只删除所有的元组，保留r。意味着还可以向r中插入新的元组
* 关系属性更改：
  + 语法：
    - alter table r add A D;
    - alter table r drop A;
  + 说明：
    - add为增加属性，需指明类型为D。增加后所有元组在A上取值都为null
    - drop为删除属性
* 查询
  + 基本结构：
    - select：跟属性名或含属性名的表达式（运算符、函数），用逗号隔开
    - from：跟关系。直接逗号隔开为笛卡儿积
    - where：跟谓词，可以有<、<=、>、>=、=、<>，用and、or、not连接
    - 分号结束
  + 重复：
    - 去除：select distinct
    - 保留：select all
  + 更名：old-name as new-name（可以出现在select或from子句中）
  + 字符串匹配（%匹配任意字符串，\_匹配任意一个字符）：
    - 语法：A like (string)、A not like (string)
    - string用法示例：
      * ‘Intro%’：匹配以”Intro”打头的字符串
      * ‘%Comp%’：匹配含”Comp”子串的字符串
      * ‘\_ \_ \_’：匹配只含三个字符的字符串
      * ‘\_ \_ \_%’：匹配至少含三个字符的字符串
  + 调整显示次序：
    - 语法：
      * order by A
      * order by A desc、order by A asc
      * order by A desc, B asc
    - 说明：
      * 缺省时按升序排列
      * 用desc指定降序，asc指定升序
      * 多个属性按逗号连接，优先级从左到右
  + 集合运算：
    - 语法：(r1) union/intersect/except (r2)
    - 说明：自动去除了重复元组，若要保留，在运算关键字后加all
  + 空值运算：
    - 算术运算：只要含null，则结果为null
    - 布尔运算（涉及空值的谓词均为unknown，unknown与其它谓词运算结果如下表）：

|  |  |  |  |
| --- | --- | --- | --- |
|  | **true** | **false** | **unknown** |
| **and** | unknown | false | unknown |
| **or** | true | unknown | unknown |
| **not** | unknown | | |

* + - 测试空值：A is (not) null
    - 测试谓词是否为未知：predicate is (not) unknown
  + 聚集函数：输入值集返回单个值
    - 类型：
      * 平均值：avg，输入必须为数字集
      * 最小值：min
      * 最大值：max
      * 总和：sum，输入必须为数字集
      * 计数：count
    - 语法：
      * function(A)
      * function(distinct A)（去重后再进行聚集）
      * count(\*)（统计元组个数）
  + 分组：
    - 语法：group by A, B, ..., N
    - 说明：在指定的属性集上相等的元组被分在一个组。**没有出现在group by子句，却出现在select子句中的属性，只能作为聚集函数的参数**
  + 分组后筛选：having predicate（有having必有group by，因为having作用于分组后）
  + 判断是否为集合成员：
    - 语法：
      * A (not) in (r)（r中只有1个属性，且与A同名）
      * A (not) in (value1, value2, ..., valueN)
      * (A, B, ..., N) (not) in (r)（r中有N个属性，且与A到N一一对应）
  + 集合比较：
    - 语法：A (compare symbol) some/all (r)
    - 说明：
      * compare symbol可以是所有比较运算符
      * some只要存在即为true，all只有都满足才为true
      * r中只有1个属性，与A可比
  + 判断关系是否为空：(not) exists (r)
  + 判断关系是否有重复元组：(not) unique (r)
  + 定义临时关系：
    - 语法：
      * with name as (r)
      * with name(A, B, ..., N) as (r)
      * with name1(A1, ..., An) as (r1), name2(B1, ..., Bn) as (r2)
    - 说明：
      * 定义一个临时关系为name，其属性名为原始属性名
      * 定义一个临时关系为name，其属性名依次变为A到N
      * 定义两个临时关系name1、name2
      * **与同级select子句无标点分隔**
  + 标量子查询：结果为1行1列的查询，出现在期望出现一个值的地方，将返回的唯一一个值发挥作用
* 删除：
  + 语法：delete from r where P;
  + 说明：
    - 只能作用在一个关系上
    - 若where子句为空，则删除所有元组，保留关系r
    - **删除会影响聚集函数的结果，若以聚集函数为谓词进行删除，先用with记录删除前的情况！！！**
* 插入：
  + 语法：
    - insert into r values (v1, v2, ..., vN);
    - insert into r (A, B, ..., N) values (v1, v2, ..., vN);
    - insert into r (查询结果)
  + 说明：
    - 向r中插入一个属性值为v1到vN的元组
    - 向r中插入元组，其属性A值为v1...属性N值为vN
    - 向r中插入一个查询结果的所有元组
* 更新：
  + 语法：update r set A = (新值) where predicate
  + 说明：
    - 在r中满足predicate的元组上的A属性执行属性A的set操作
    - 若where子句为空，则在r上所有元组执行属性A的set操作
    - update的次序非常重要，可以使用case结构分情况执行更新
  + case结构：
    - case
      * when pred1 then result1
      * when pred2 then result2
      * ...
      * when predn then resultn
      * else result0
    - end

**Chapter 04：中级SQL**

**-----------------------------------------------------------**

* 连接
  + 连接类型：
    - 内连接：inner join（仅保留两边有匹配的项）
    - 左外连接：left outer join（保留左侧全部的项）
    - 右外连接：right outer join（保留右侧全部的项）
    - 全外连接：full outer join（保留两侧全部的项）
  + 连接条件
    - 自然连接：natural（按两个关系的同名属性匹配，结果中**同名属性只有一个**）
    - 谓词连接：on predicate（按满足谓词的条件匹配，**不消去同名属性**）
    - 指定属性连接：using (A, B, ..., N)（按指定的同名属性连接，结果中**同名属性只有一个**）
  + 语法：
    - 连接类型写在两个关系名中间。先指定内连接或外连接，缺省为内连接，若为外连接则需指明外连接类型。然后加join关键字
    - 连接条件上，如果是自然连接，natrual写在连接类型前，如果是其它类型的条件，则写在第二个关系名的后面
  + 说明：不能用外连接时，可以用内连接+缺省项用null补全，select出来后合并
* 视图：
  + 定义：一种临时关系。相比with子句，视图可以在不同的查询中通用，在被显式删除前一直可用
  + 语法：
    - create view v as <查询结果>
    - create view v(A, B, ..., N) as <查询结果>
  + 说明：
    - v是视图名，查询结果最好用sql查询结果，即表来定义
    - 可以通过第二种方式指定视图的属性名
  + 物化视图
    - 定义：随着所依赖的实际关系变化的视图
    - 语法：create materialized view as ...
    - 物化视图维护（视图维护）：保持物化视图在最新状态
* 事务：
  + 组成：查询及更新语句的序列
  + 语法：
    - 提交事务：commit work
    - 回滚事务：rollback work
    - 关闭自动提交：set autocommit off（缺省情况下**每条SQL语句就是一个事务**，执行完就提交。若一个事务是**由多条SQL语句构成**，为了保持**原子性**，需要将自动提交关闭）
* 完整性约束
  + 单个关系上的约束（在DDL中）
    - 属性非空：not null
    - 指定属性无重复：unique(A, B, ..., N)
    - 每个元组都满足：check(predicate)（predicate可以含SQL语句，例如检查一个属性是否在一个SQL查询结果的表中）
  + 引用完整性：同Chapter 03。在此基础上可以设置级联修改：
    - 级联删除：on delete cascade
    - 级联更新：on update cascade
  + 给约束赋名：
    - 语法：constraint name <constraint>
    - 说明：name是约束名，<constraint>是约束，如check()、not null等。可以跟在属性定义后
    - 删除方法：alter table table\_name drop constraint con\_name
  + 断言：
    - 定义：相比于一般的predicate，断言可以在不同的查询中通用
    - 语法：create assertion assert\_name check predicate
* 数据类型：
  + 日期和时间：
    - 日期：date（有年、月、日）
    - 时间：time（有时、分、秒，time(p)可指定秒有几位小数）
    - 时间戳：timestamp（date和time的结合，timestamp(p)用法同上）
    - 提取单独的项：extract (field from d)。从类型为date或time的值d中提取单独的域field（year、month、day、hour、minute、second等）
  + 用户自定义类型：
    - 无约束类型：create type type\_name as <type> (final)（关键字final可省）
    - 有约束类型（域）：create domain dom\_name as <type> <constraint>
* 索引：
  + 语法：create (unique) index index\_name on r (A, B, ..., N)
  + 说明：默认在主码上创建索引。在声明后面的属性集合为候选码时加上unique（若实际上不是候选码则会报错）
  + 删除：drop index index\_name
* 权限：
  + 语法：grant/revoke <privileges> on r to <users>/public
  + 说明：
    - 授权grant，收权revoke
    - 权限有select、insert、update、delete
    - public用户名指系统所有当前用户和将来的用户（revoke时用public无法收回grant的权限）
    - 对同一权限，grant了n次，revoke了m次。若n>m，则权限仍拥有
    - u1将权限给了u2，收回u1权限也会收回u2的权限

**Chapter 05：高级SQL**

**-----------------------------------------------------------**

* 函数：
  + 语法：
    - create function func\_name(parameters)
      * returns <return type>
      * begin
      * <function process>
      * return <value>**;**
      * end
  + 说明：
    - parameters中各参数用逗号隔开，形式为参数名+类型
    - 分号出现的位置：①<function process>中的declare声明临时变量后；②end前的return语句后
    - <function process>若为空，则begin、end可省
    - 赋值：
* 过程
  + 定义：和函数类似，无返回值
  + 语法：
    - create procedure proc\_name(parameters)
    - begin
    - <procedure>
    - end
  + 说明：
    - 可以用in、out修饰参数，out在过程结束前必须给其赋值
    - 调用方法：call proc\_name(arguments)
* 循环结构
  + 语法：
    - while (bool-expression) do
    - <loop block>
    - end while
    - repeat
    - <loop block>
    - until (bool-expression)
    - end repeat
    - for r as
    - <relation>
    - do
    - <operation per iteration> //可以用set改变临时变量值
    - end for
* 分支结构：
  + 语法：
    - if (bool-expression-1)
    - then <select-block-1>
    - elseif (bool-expression-2)
    - then <select-block-2>
    - else
    - <select-block-3>
    - end if

**Chapter 06：使用E-R模型的数据库设计**

**-----------------------------------------------------------**

* 数据库模式设计中避免的主要缺陷
  + 冗余：不好的设计可能会重复信息
  + 不完整：不好的设计可能会使得建模难以进行
* E-R图：通过图形方式表示数据库的总体逻辑结构
  + 实体集：共享相同性质或属性的、具有相同类型的实体的集合
  + 联系集：
    - 定义：是相同类型联系的集合
    - 联系：多个实体间的相互关联
    - 描述性属性：联系集独有的属性
    - 度：参与联系集的实体集数目
  + 角色：实体在联系中扮演的功能（递归联系集：同一实体集以不同角色多次参与一个联系集）
  + 复杂属性：
    - 复合属性：可以被划分成子部分的属性（用缩进表示其各个子部分）
    - 多值属性：对一个实体可以有多个值的属性（用花括号括起多值属性）
    - 派生属性：可以从其他相关属性或值派生出值的属性（在属性名后加圆括号）
  + 映射基数：
    - 定义：表示一个实体通过一个联系集能关联到另外一些实体的数量
    - 二元情况：一对一、一对多、多对一、多对多
    - 说明：
      * “一”方用箭头表示，指向实体集
      * 若实体集全部参与联系，则用双线表示
      * 三元关系最多有一个“一”方，否则有歧义
  + 弱实体集：
    - 定义：依赖于其它实体集存在的实体集
    - 说明：
      * 用双框矩形表示
      * 标识性实体集是被依赖的一方
      * 标识性实体集拥有它所标记的弱实体集
      * 弱关系集将标识性实体集与弱实体集联系起来，用双框菱形表示
* 将E-R图转化为关系模式
  + 实体集：
    - 强实体集：名称、属性、主码与实体集的一致
      * 复合属性：将每个子部分单独作为一个属性
      * 多值属性：单独构建关系模式，属性为所在实体集的主码+对应的多值属性值，主码也是这两者
      * 派生属性：去掉
    - 弱实体集：名称与实体集一致，属性为标识性实体集主码+自身属性，主码为标识性实体集主码+自身分辨符
  + 联系集：名称都与联系集的一致
    - 多对多：属性为所有参与联系的实体集主码+联系集本身属性。主码为所有参与联系的实体集主码
    - 一对多/多对一：
      * 联系集属性少：不需转换，在多方增加一方的主码和联系集自身属性（主码不变）
      * 联系集属性多：自身属性+双方主码（主码是多方主码）
    - 一对一：任意一方当成多方并进行一对多/多对一的操作
    - 标识性：不转化
* E-R特性：
  + 特化：
    - 定义：在实体集内部进行分组
    - 类型：
      * 重叠特化：一个实体可以属于多个特化实体集
      * 不相交特化：一个实体只能属于一个特化实体集
    - 表示：空心箭头，从子类指向父类
  + 概化：
    - 定义：多个实体集根据共有特征综合成更高层的实体集
    - 说明：与特化类似，只是自底向上
  + 完全性约束
    - 全部特化/概化：每个高层实体必须属于一个低层实体集（在箭头旁标total）
    - 部分特化/概化：一些高层实体可以不属于任何低层实体集（默认）
  + 聚集：一种抽象，联系集被视为高层实体

**Chapter 07：关系数据库设计**

**-----------------------------------------------------------**

**设计目标：**

* 减少冗余
* 易于检索
* 没有信息损失

**有损/无损分解：**

* 定义：分解再连接后是否得到原始的关系
* 判断：分解后两个关系的交集是否为其中一个关系的超码，是则为无损，否则为有损（即或成立）

**规范化理论：**

判断关系模式是否是好的，不好则以无损分解将其分解成更小的关系模式

**合法实例：**

一个关系实例满足所有约束（数据库合法实例：所有关系实例都是合法实例）

**函数依赖：**

* 定义：给定r(R)的一个实例，对于其所有元组对t1和t2，若t1[α]=t2[α]，则t1[β]=t2[β]也成立
* 表示：
* 平凡情况：

**函数依赖集：**

* 定义：在某个关系上成立的函数依赖的集合
* 表示：F

**BC范式：**

所有F+中的函数依赖，要么为平凡的，要么α是R的超码

**保持依赖的分解：**

检查分解后的关系是否满足原有函数依赖时，不需要对分解后关系进行连接，这种分解就是保持依赖的（有时会与BCNF要求冲突，因此**工程实践上保持依赖的优先级更高**）

**第三范式：**

所有F+中的函数依赖，要么为平凡的，要么是R的超码，要么β-α的每个属性都在R的某个候选码中

**两种范式比较：**

* 3NF优点：总可以在无损分解以及保持依赖的条件下得到3NF
* 3NF缺点：不得不用空值表示数据项之间可能有意义的联系，并且会有信息重复

**函数依赖集的闭包：**

* 逻辑蕴涵：能从F推出的函数依赖f，称F逻辑蕴涵f
* 定义：F逻辑蕴涵的所有函数依赖的集合
* 表示：F+
* Armstrong公理：
  + 自反律：若，则
  + 增补律：若，则
  + 传递律：若且，则
* 推论：
  + 合并律：若且，则
  + 分解律：若，则且
  + 伪传递律：若且，则

**属性集的闭包**

* 定义：函数依赖集F下被α函数决定的所有属性的集合
* 表示：α+
* 计算：
  + result首先置为α
  + 对于F中的每个函数依赖，若β是result的子集，则将γ并到result中
  + 这一过程若引起了result改变，则继续下一轮直至result不改变为止
  + 返回result
* 用法：
  + 检查α是否为超码
  + 检查某个左侧为α的函数依赖是否在F上成立
  + 计算F+（对R的所有子集γ，计算γ+，并对于γ+的所有子集S，得到一个函数依赖。所有的函数依赖求出则得到F+）

**正则覆盖：**

* 无关属性：
  + 定义：对F以及其中的一个函数依赖：
    - ，**F逻辑蕴涵**，则属性A在α无关
    - ，**逻辑蕴涵F**，则属性A在β无关
  + 判定：
    - ，先求，再求F’下α+。如果α+有A，则A是无关的
    - ，先求，在F下求γ+。如果γ+包含了β，则A是无关的
* 表示：Fc
* 定义：F与Fc相互逻辑蕴涵所有的函数依赖。但Fc中任何函数依赖都没有无关属性，且每个函数依赖左侧都是唯一的
* 计算：
  + Fc初始化为F
  + 使用合并律将Fc中任何形如、的依赖合并成
  + 检查Fc所有的函数依赖中的所有属性，找到无关属性则从所在函数依赖的那一侧消去（一侧仅剩下一个属性时停止在这一侧的检查）
  + 这一过程若引起了Fc改变，则继续下一轮直至Fc不改变为止
  + 得到Fc
* 说明：当同一侧多个属性均为无关时，删去任意的属性均符合要求。因此对一个特定的F，Fc不唯一

**保持依赖的检测：**

* 背景：F为R上的函数依赖集，R1，R2，...，Rn是R的一个分解
* 计算：
  + 方法一：令Rn的限定Fn为只包含Rn属性的函数依赖集合，，若F’+=F+，则为保持依赖的分解（计算闭包，开销过大）
  + 方法二：对F中所有函数依赖执行以下过程：
    - result初始化为α
    - 对每个Ri：，
    - 这一过程若引起了result改变，则继续下一轮直至result不改变为止
    - 若result包含了β，则被保持住了
    - F中所有依赖都被保持住，则是保持依赖的分解

**BCNF分解：**

* BCNF判断推论：仅需检查给定F中的依赖是否违反即可
* BCNF分解后检测方法：对于Ri的每个子集α，α+要么不包含Ri-α的任何属性，要么包含Ri的所有属性。一旦有α违反了，则说明Ri不是BCNF。考虑依赖，它存在于F+中且使得Ri不是BCNF
* 算法：
  + result初始只有一个待分解的关系R
  + 检查result中每一个关系是否为BCNF（若result中仅有R，则**按定义检测**；若已经发生了分解，则**按分解后检测方法检测**）
  + 若有关系Ri不是BCNF，找到一个使得Ri不是BCNF的函数依赖
  + 
  + 当result中所有的关系都为BCNF时结束
* 说明：BCNF分解是无损分解，但不一定保持依赖

**3NF分解：**

* 说明：
  + 3NF分解既是无损分解，又是保持依赖的分解
  + 最好在执行算法前找到R的所有候选码
* 算法：
  + 计算F的正则覆盖Fc
  + 对于Fc中的第i个函数依赖，得到关系Ri=αβ
  + 若Ri中没有一个包含R的一个候选码，则再得到一个关系，其属性为R的一个候选码
  + 把被其他模式包含的模式删掉，直到没有可删除的模式便结束

**多值依赖：**

* 定义：给定r(R)的一个实例，对于其所有元组对t1和t2，若t1[α]=t2[α]，都存在t3、t4使得：

|  |  |  |  |
| --- | --- | --- | --- |
|  | α | β | R-β |
| t1 | α1 | β1 | w1 |
| t2 | α1 | β2 | w2 |
| t3 | α1 | β1 | w2 |
| t4 | α1 | β2 | w1 |

* 表示：
* 平凡情况：或
* 函数依赖与多值依赖的集合：D（其闭包为D+）
* 推论：
  + 若，则
  + 若，则

**第四范式：**

* 定义：所有D+中的多值依赖，要么为平凡的，要么α是R的超码
* 4NFBCNF

**4NF分解：**

* D在Ri上的限定：
  + D+中只含Ri中属性的**函数依赖**
  + 形如的**多值依赖**，其中且属于D+
* 4NF分解后检测方法：求出Di后用定义判断Ri是否为4NF
* 算法：与BCNF分解类似，其中找出不是4NF的关系后要找出使其不是4NF的**多值依赖**，并分解

**Chapter 11：数据分析**

**-----------------------------------------------------------**

* 关联规则：两个事件（前提A和结论B）之间的关联
  + 支持度：同时满足前提和结论的个体所占比例P(AB)
  + 置信度：当前提为真时结论为真的概率P(B|A)

**Chapter 14：索引**

**-----------------------------------------------------------**

**基本类型：**

* 顺序索引：基于值的顺序排序
* 散列索引：将值平均分布到若干桶中。值所属的桶由散列函数决定

**评价因素：**

访问类型、访问时间、插入时间、删除时间、空间开销

**搜索码：**

用于在文件中查找记录的属性或属性集

**顺序索引：**

* 聚集索引（主索引）：搜索码定义了文件的次序
* 非聚集索引（辅助索引）：搜索码顺序与文件顺序不同
* 索引项（索引记录）：搜索码值+指针
* 稠密索引：对文件中**每个搜索码值（不是数据项，因此不一定一对一）**都有一个索引项
* 稀疏索引：只为某些搜索码值建立索引项**（只有用聚集索引时才可以使用稀疏索引）**
* 多级索引：两级及以上的索引（为了缩小索引，提高搜索效率）
* 更新：
  + 插入：
    - 稠密索引：
      * 若搜索码值未出现，则在适当位置插入新的索引项
      * 若出现且索引项存储了同搜索码值的所有项，则在索引项中增加一个指向新记录的指针
      * 若出现但索引项对于同搜索码值的项只存了一个指针，则把新记录放在同搜索码值的其他记录之后
    - 稀疏索引（设每个块保留一个索引项）：
      * 若系统创建新块，新记录作为新块的第一个搜索码值插入索引
      * 若新记录由最小搜索码值，则系统更新所在块的索引项
      * 否则不做改动
  + 删除：
    - 稠密索引：
      * 若待删除的记录是同搜索码值的唯一一条记录，则删除相应的索引项
      * 若不唯一且索引项存储了同搜索码值的所有项，删除索引项指向待删除记录的指针
      * 若不唯一但索引项对于同搜索码值的项只存了一个指针，如果待删除记录是同搜索码值的第一项，则更新索引项使其指到下一条
    - 稀疏索引：
      * 若索引中并不包含对应索引项，则不做改动
      * 若待删除记录是同搜索码值的唯一一条记录，则用下一个搜索码值替换对应的索引项。若下一个搜索码值已经有索引项，则删除对应的索引项
      * 若不唯一，则更新索引项使其指向下一条同搜索码值的记录

**辅助索引：**

辅助索引**必须是稠密的**

**多码索引：**

* 包含多个属性的搜索码叫复合搜索码
* 复合搜索码比较到第一个不等的属性并取不等的结果。只有全相等才返回相等的结果（字典序排列）

**B+树索引：**

* 性质：多级索引的平衡树
* 评价：增加了插入和删除，以及空间的开销。但对频繁更新的文件来说可以接受，它避免了开销更大的文件重组
* 结点：有n-1个搜索码值K1, K2, ..., Kn-1以及n个指针P1, P2, ..., Pn
  + 特点：
    - 为提高利用率，至少要半满，叶结点最少包含个**值**
    - 同一树中，左边叶结点的所有搜索码值都小于右边叶结点的所有搜索码值
    - 非叶结点至少容纳个**指针（根结点例外）**
    - 考虑到索引存在一个块中效率更高，一般**n = 块大小/索引项大小**，意味着一个结点能容纳的项数
  + 叶结点：前n-1个指针指向具有搜索码值Ki的记录，Pn指向同级下一个叶结点
* 搜索：
  + 单个搜索：
    - C从根结点开始
    - 到达叶结点层前，找到满足v≤C.Ki最小的i
      * 若i不存在则沿结点最后一个指针下去一层
      * 若i存在，Ki刚好是v则沿C.Pi+1下去一层；Ki不是v则沿C.Pi下去一层
    - 到达叶结点层后，第i个搜索码是v则返回Pi；否则返回空，不存在v
  + 范围搜索：有下界搜索和上界搜索
* 插入：找到需插入的叶结点并插入
  + 叶结点拆分：插入后叶结点溢出时发生。前个**值和指针**放在前一个结点，前一个结点最后一个指针指向后一个结点，剩下的值和指针放在后一个结点。同时设后一个结点第一个搜索码值为v，指向后一个结点的指针为p。在父节点中添加v和p
  + 非叶结点拆分：插入后非叶结点溢出时发生。前个**指针及其夹住的值**放在前一个结点，剩下的指针及其夹住的值放在后一个结点。同时会产生一个“夹在两结点间”的一个值，将这个值以及指向新生成的结点的指针加到再上一层的结点中
* 删除：找到需删除的项后删除，未找到则不做操作
  + 判断被删结点剩下的**值数**是否不小于个
    - 若是，不必合并，删除索引项即可
    - 若不是，优先向右合并，无右侧节点向左合并**（无论是向哪一侧合并，都把所有的值放在靠左节点，因为靠右节点及其在父节点中的值和指针将要删除）**。父节点删除合并前未半满的指针及其搜索码。再判断父节点**指针数**是否不小于个
      * 若不是，向左合并指针，产生的值空位由祖父结点下移一个值填补
      * 若不是且左侧结点已满，则向左侧借最后一个指针，左值上移，父值下移
    - 若叶结点左右都无法合并，向左侧结点借一个值直到半满，再更新父节点值

**B树索引：**

* 结构：每个搜索码值只允许在树中出现一次，因此**非叶结点**对**每个搜索码值增加了一个指针域**，指向文件记录或该搜索码值的指针桶
* 与B+树差异：
  + B树去除了搜索码值的冗余存储，B+树没有
  + 对于相同的非叶结点大小，B树存储搜索码个数较少，B+树较多
  + 搜索一个值，B树可能提前找到，搜索时间差异大；B+树总要到达叶结点层，搜索时间差异小
  + 由于非叶结点存储搜索码值数量较B+树要少，B树会深一点
  + B树的删除要考虑不在叶结点删除的情况，更为复杂；B+树较为简单

**散列索引：**

* 桶：可以存储一条或多条记录的存储单元
* 一个变种：溢出链（该种索引叫**闭寻址或闭散列**）。用链表表示一系列相同散列函数值的桶，在桶溢出时，在链表中新增一个桶并存放新增记录。设某桶链表上，桶A为第一个，则其余桶称为桶A的溢出桶

**位图索引：**

* 背景：为简化多码上的简单查询
* 要求：所涉及的**码的取值只有有限个**
* 用法：构建位的一维数组，设该索引为属性A值v创建，按顺序表示每条记录属性A的情况。若值是v则为1，若不是则为0
* 应用：根据查询要求，对位图进行位与、位或、位非运算，得出筛选结果

**Chapter 17：事务**

**-----------------------------------------------------------**

**特性（ACID）：**

* 原子性：要么完全执行，要么完全不执行
* 一致性：以隔离方式执行事务以保持数据库的一致性
* 隔离性：在事务看来，其他事务要么在其前执行完，要么在其执行完后再执行
* 持久性：事务执行完后，对于数据库的改变是持久的

**状态：**

* 活跃：初始态
* 部分提交：最后一条语句被执行之后
* 失效：发现正常执行不能继续之后
* 终止：事务已回滚并撤销所有对数据库的更改
* 提交：事务成功完成之后

**调度：**

* 定义：指令在系统中执行的时间顺序
* 要求：
  + 一组事务的调度**必须包含**各个事务的**所有指令**
  + 这些指令**必须保持**它们在每个**单独事务中出现的顺序**
* 类型：
  + 串行的：各个事务中的指令在调度中一起出现（n!种）
  + 可串行化的：并发执行下，等价于一个串行调度的调度
    - 冲突可串行化：一个串行调度通过一系列非冲突指令的交换，得到一个等价的调度S，那么调度S就是冲突可串行化的
    - 视图可串行化：①对每一个数据项，每一次**读到的值是哪个事务写的**，要在不同调度中一致；②执行过程**最后一次写是哪个事务做的**，要在不同调度中一致