




数据库系统概念复习纲

题型设置

大题（4）

1. 关系代数（25）
 - a) 写结果
 - b) 处理空值空表
2. SQL（25）
 - a) 单表查询
 - b) 多表连接查询
 - c) 减法
 - d) 除法 
 - e) 聚集函数
 - f) with（复杂查询、临时表格）
 - g) update/delete 
3. E-R（10） 
4. 规范化（10）

简答（5-6）

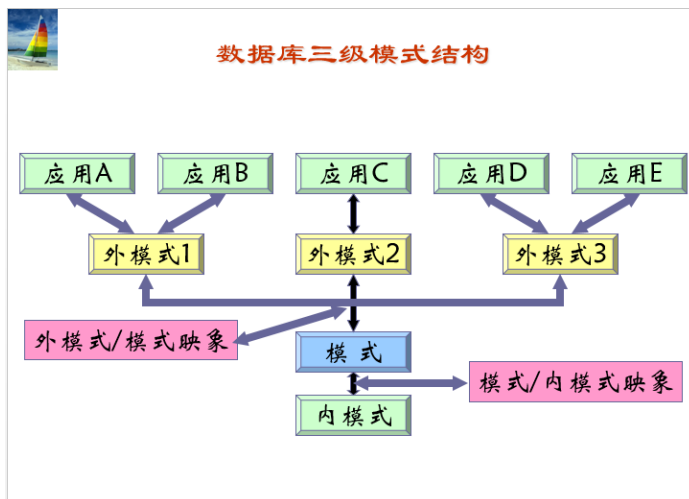
1. 事务可串行化：写调度
2. 索引
3. 查询优化（怎么做成最优的关系代数）
4. 数据库基本概念
5. 关系基本概念（空值的作用、空值的运算）
6. 码（外码）

Chapter 1

- DBMS：由一个相互关联的数据集合和一组用以访问这些数据的程序组成，这个数据集合通常称作数据库（Database），其中包含了关于某个机构的信息。或简述为系统软件，对数据库进行统一管理和控制。
- 数据：数据(Data)是数据库中存储的基本对象。数据即描述事物的符号记录。
- 数据结构：
 - 逻辑结构：数据之间存在的逻辑关系

- 物理结构：数据在计算机内的存储方式
- 数据库
 - 定义：数据库(Database,简称 DB)是长期储存在计算机内、有组织的、可共享的大量数据集合
 - 特征
 - ◆ 数据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
- 数据库系统：数据库系统（Database System，简称 DBS）是指在计算机系统中引入数据库后的系统
- 数据视图
 - 物理层:描述数据存储
 - 逻辑层:描述存储在数据库中的数据，以及数据之间的关系
 - 视图层:最高层次的抽象，只描述整个数据库的某部分数据（视图层提供了防止用户访问数据库的某些部分的安全性机制）
- 实例与模式
 - 模式 - 数据库的总体设计
 - ◆ 物理模式:在物理层描述的数据库设计
 - ◆ 逻辑模式（子模式）:在逻辑层描述的数据库设计
 - 实例 - 特定时刻存储在数据库中的信息的集合
 - ◆ 类似于一个变量的值
- 数据独立性：数据独立性是指应用程序与 DB 的数据结构之间相互独立
 - 物理数据独立性
 - ◆ 存储结构改变时，修改模式/内模式映象，使模式保持不变，从而应用程序可以保持不变，称为数据的物理独立性
 - 逻辑数据独立性
 - ◆ 当模式改变时，修改外模式/模式映象，使外模式保持不变，从而应用程序可以保持不变，称为数据的逻辑独立性

● 三级模式结构



- 描述：为了提高数据的物理独立性和逻辑独立性，使数据库的用户观点，即用户看到的数据库，与数据库的物理方面，即实际存储的数据库区分开来，数据库系统的模式是分级的，美国数据系统语言协商会提出模式、外模式、存储模式三级模式的概念
- 与独立性的联系：三级模式之间有两级映象。（其他见前一点数据独立性）
- 数据模型
 - ER 模型直接表示实体类型及实体间联系，与计算机系统无关，充分反映用户的需求，用户容易理解
 - 层次模型的数据结构为树结构，记录之间联系通过指针实现，查询较快，但DML属于过程化的，操作复杂
 - 网状模型的数据结构为有向图，记录之间联系通过指针实现，查询较快，并且容易实现M:N 联系，但DML 属于过程化的语言，编程较复杂。
 - 关系模型的数据结构为二维表格，容易为初学者理解。记录之间联系通过关键码实现。DML 属于非过程化语言，编程较简单。
 - 面向对象模型能完整描述现实世界的数据结构，具有丰富的表达能力，能表达嵌套、递归的数据结构。但涉及的知识面较广，用户较难理解
- 数据库语言
 - DML (Data Manipulation Language) :操纵那些按照某种适当的数据模型组织起来的数据的语言
 - DDL (Data Definition Language) :用于定义数据库模式以及其他特征的语言
- 数据存储和查询
 - 存储管理器是一个程序模块，提供了数据库中存储的低层数据与应用程序以及向系统提交的查询之间的接口
 - 查询处理器
 - ◆ DDL解释器：它解释DDL语句，并将这些定义记录在数据字典中
 - ◆ DML编译器：将查询语言中的DML语句翻译成为一个执行方案
 - ◆ 查询执行引擎：执行由DML编译器产生的低级指令

Chapter 2 & Chapter 6

- 关系数据库
 - 关系数据库是表的集合
 - 关系模型
 - ◆ 列首称为属性
 - ◆ 每个属性有一组允许的值，称为该属性的域
 - 关系是一系列域上的笛卡尔积的子集
 - 元组（代替表中的行）是以所有元组集为域的变量
 - 域是原子的
- 空值或值null是所有可能的域的成员，表明值未知或不存在。
 - 即空值就是表示“无意义”，当实体在某个属性上没有值时设为null;
 - 或者表示“值未知”，即值存在，但目前没有获得该信息;
 - 当空值参与运算，结果为空值。
- 码：能唯一标识实体的属性集，他是整个实体集的性质，而不是单个实体的性质。

- 外码：一个关系模式 r_1 可能在它的属性中包含另一个关系模式 r_2 的主码，这个属性在上称作在 r_1 上参照 r_2 的外码（ r_1 和 r_2 可以是同一个关系）。
 - ◆ 关系 r_1 称作外码依赖的参照关系
 - ◆ 关系 r_2 称作外码的被参照关系
- 超码：一个或多个属性的集合，这些属性的组合可以使我们在一个关系中唯一地标识一个元组。
- 候选码：它们的任意真子集都不能成为超码，这样的最小超码成为候选码。
- 主码：代表被数据库设计者选中的、用来在同一关系中区分不同元组的候选码。
- 实体完整性约束：关系的主码中的属性值不能为空值

● 关系代数运算

■ 基本运算

- ◆ 一元运算
 - ◇ 选择
 - ◇ 投影
 - ◇ 更名
- ◆ 多元运算
 - ◇ 笛卡儿积
 - ◇ 并
 - ◇ 集合差

■ 其它运算

- ◆ 集合交
- ◆ θ 连接
 - ◇ 定义：从两个关系的广义笛卡儿积中选取给定属性间满足一定条件的元组

$$R \bowtie_{A \theta B} S = \{ \overline{rs} \mid r \in R \wedge s \in S \wedge r[A] \theta s[B] \}$$

A, B 为 R 和 S 上度数相等且可比的属性列
 θ 为算术比较符，为等号时称为等值连接

◇ 表示： $R \bowtie_{A \theta B} S = \sigma_{r[A] \theta s[B]}(R \times S)$

- ◆ 自然连接
 - ◇ 对于 r 中的每个元组 t_r 和 s 中的每个元组 t_s 所组成的元组对
 如果 t_r 和 t_s 在 $R \cap S$ 的属性上有相同的值，则在结果中加入一个元组 t ，
 并且 t 在 r 上和 t_r 有相同的值 t ， 在 s 上和 t_s 有相同的值
- ◆ 除
 - ◇ 给定两个关系 $r(R)$ 和 $s(S)$ ，并且 $S \subset R$ ， 则 $r \div s$ 是满足 $t \times s \subseteq r$ 的最大的关系 $t(R-S)$
 - ◇ 特性：①只取除尽记录②只除重复字段③多字段下强调其余字段全相同
- ◆ 赋值
- ◆ 外连接
 - ◇ 所有的包含null 比较运算都被定义为false
 - ◇ 分类：左外连接、右外连接、全外连接

■ 扩展运算

- ◆ 广义投影
 - ◇ 广义投影运算通过允许在投影列表中使用算数函数来对投影进行扩展

$$\Pi_{F_1, F_2, \dots, F_n}(E)$$

- ◇ E 是任意关系代数表达式
- ◇ F_1, F_2, \dots, F_n 是涉及常量以及 E 的模式中属性的算术表达式

◆ 聚集函数和聚集运算

- ◇ 分类:
 - avg: 平均值 min: 最小值 max: 最大值 sum: 求和 count: 计数
- ◇ 表示: 在关系代数中

$$G_1, G_2, \dots, G_n \mathcal{G} F_1(A_1), F_2(A_2, \dots, F_n(A_n))(E)$$

- ◇ E 是任一关系代数表达式
- ◇ G_1, G_2, \dots, G_n 是用于分组的一系列属性（可以为空）
- ◇ 每个 F_i 是一个聚集函数
- ◇ 每个 A_i 是一个属性名

■ 空值处理

- ◆ 元组的一些属性可以为空值，用 null 表示
- ◆ null 代表一个值未知或不存在，空值是一种状态，不是一个明确的值
- ◆ 含有 null 的算术表达式的结果为 null
- ◆ 聚集函数直接忽略空值（比如在SQL里）
- ◆ 在去重和分组运算中，null 和其他值一样，两个null 被看作是相同的值（比如在SQL里）
 - ◇ 含有空值的比较运算的结果是一个特殊的值：unknown
 - 使用 unknown 的三值逻辑：
 - OR: (unknown or true) = true,
(unknown or false) = unknown
(unknown or unknown) = unknown
 - AND: (true and unknown) = unknown,
(false and unknown) = false,
(unknown and unknown) = unknown
 - NOT: (not unknown) = unknown

● 元组关系演算（与域关系演算近似，故略之）

■ 谓词演算公式

- ◆ 属性和常量的集合
- ◆ 比较运算符的集合（示例：<, ≤, =, ≠, >, ≥）
- ◆ 连词的集合：and (∧), or (∨), not (¬)
- ◆ 蕴含 (⇒): $x \Rightarrow y$, 如果x为真，则y也为真
$$x \Rightarrow y \equiv \neg x \vee y$$
- ◆ 量词的集合：
 - ◆ $\exists t \in r(Q(t))$ ≡ 在关系 r 里“存在”一个元组 t 则谓词Q(t)为真
 - ◆ $\forall t \in r(Q(t))$ ≡ Q对关系 r 里的“所有”元组 t 都为真

■ 元组关系

- ◆ 元组关系演算是非过程化的查询语言，查询表达的形式为
$$\{t \mid P(t)\}$$
- ◆ 表示它是所有使谓词P为真的元组 t 的集合

- ◆ t 为元组变量, $t[A]$ 表示元组 t 在属性 A 上的值
- ◆ $t \in r$ 表示元组 t 在关系 r 中
- ◆ P 是一个类似于谓词演算的公式
- 元组关系演算与关系代数的等价性
 - ◆ 投影 $\Pi_A(R) = \{t \mid \exists s \in R (t[A] = s[A])\}$
 - ◆ 选择 $\sigma_F(A)(R) = \{t \mid t \in R \wedge F(t[A])\}$
 - ◆ 广义笛卡儿积 $R(A) \times S(B) = \{t \mid \exists u \in R \exists s \in S (t[A] = u[A] \wedge t[B] = s[B])\}$
 - ◆ 并 $R \cup S = \{t \mid t \in R \vee t \in S\}$
 - ◆ 差 $R - S = \{t \mid t \in R \wedge \neg t \in S\}$

Chapter 3 & Chapter 4 & Chapter 5

- 解释与限定
 - 主码约束
 - ◆ 主码值不允许空, 也不允许出现重复
 - ◆ 意义: 关系对应到现实世界中的实体集, 元组对应到实体, 实体是相互可区分的, 通过主码来唯一标识, 若主码为空, 则出现不可标识的实体, 这是不容许的
 - 参照完整性
 - ◆ 定义: 保证在一个关系中给定属性集上的取值也在另一关系的特定属性集的取值中出现
 - ◆ 相关: A 是一个属性的集合, R 和 S 是两个包含属性 A 关系, 并且 A 是 S 的主码 如果对于每个在 R 中出现的 A 在 S 中也出现, 则 A 被称为 R 的外码
 - ◆ 样例: 如果关系 R_2 的外部码 Fk 与关系 R_1 的主码 Pk 相对应, 则 R_2 中的每一个元组的 Fk 值或者等于 R_1 中某个元组的 Pk 值, 或者为空值。
 - 嵌入式 SQL
 - ◆ 操作方式的协调
 - ◇ 执行方式的差别
 - SQL: 一次一集合
 - 主语言: 一次一记录
 - ◇ 游标
 - 在查询结果的记录集合中移动的指针
 - 若一个 SQL 语句返回单个元组, 则不用游标
 - 若一个 SQL 语句返回多个元组, 则使用游标
 - 故 INSERT、DELETE 和 UPDATE 语句; 以及对于 SELECT 语句, 如果已知查询结果肯定是单值时, DML 语句不用游标
 - 指令与描述 (在此仅给出简单表示, 详细说明见书本)
 - select [distinct][all][*][operator] A (from T)
 - where [operator] (= > < and or not) P 【注意 null 处理】
 - from T (A,B 笛卡尔积 / A natural join B / A join B using P)
 - as (rename)
 - ◆ 【记在 select 语句 A 后】 (即选出值作为新字段)
 - ◆ 【记在 from 语句 T 后】 (即选出表作为新表名)

- like P 【记在 where A 之后】（取符合 P 记录的 A 字段值）
 - ◆ %为全局通配符 _为单字通配符
 - ◆ escape 转义 【格式： like W escape T】（将 T 作为转义符匹配 W 记录）
- order by 【记在 select 之后】 [desc/asc]（默认不加为升序）
- between ... and ... 【记在 where 之后】（选出其间的记录）
- union (all) / intersect (all) / except (all)（字段名取出须完全相同）（在没有 all 的时候自动去重，否则保持重复）
- is null / or / and / not 【记在 where 语句 P 之后或之间】
- 聚集函数 avg / min / max / sum / count 【记在 select 语句之后 包裹属性】
- group by 【记在所有语句最后】（字段如无记录则不显示）
 - ◆ 分组加入条件 having 【格式： group by A having C】
- in 【记在 where 语句的属性之后，用于语句复合】
 - ◆ not in（不是不等于，同时支持元组返回）
- 比较函数 >some <some >=some <=some =some <>some >all <all >=all <=all =all <>all
【记在 where 语句属性之后 记比较结果返回】
- exists（返回存在的是否为空集）【记在 where 语句之后，本语句为复合语句，里面包裹一个完整 select 语句】
 - ◆ not exists（反义使用）
- unique（返回唯一性）【记在 where 语句之后，本语句为复合语句，里面包裹一个完整 select 语句】
 - ◆ not unique（反义使用）
- from clause（from 语句的嵌套子查询，子查询为临时关系，仅在语句内有效）
 - ◆ lateral（跟在复合语句之前，允许子查询访问父查询）【记在 from 语句之后】
- with（临时定义表格，仅对此次 SQL 有效）【格式： with T(A1) as (select clause)】
- delete 【格式： delete from T where P】（从 T 删除满足 P 的字段）
- insert 【格式： insert into T (A...) values (A...)】（从 T 中插入对应的（若存在）或全部字段的一条记录进入 T）
 - ◆ insert 语句可用 select 语句 where 嵌套复合进行条件匹配插入

Chapter 7

- E-R 关系
 - 实体相关
 - ◆ 实体：是现实世界中可区别于所有其他对象的一个“事物”或“对象”
 - ◆ 实体集：是相同类型即具有相同性质（或属性）的一个实体集合
 - ◆ 属性：实体具有属性，属性是实体中每个成员所拥有的描述性性质，实体中的每一个属性都有一个值
 - 联系相关
 - ◆ 联系：是指多个实体间的相互关联
 - ◆ 联系集：是相同类型联系的集合，是 $n \geq 2$ 个（可能相同的）实体集上的数学联系
 - ◆ 联系也可以具有描述性属性
 - ◆ 联系集的度：参与联系集的实体集数目，数据库系统中的大部分联系集都是二元的

- 属性相关
 - ◆ 域或值集：每个属性都有一个可取值的集合
 - ◆ 属性类型：
 - ◇ 简单 (Simple) and 复合 (composite) 属性.
 - ◇ 单值 (Single-valued) and 多值 (multivalued) 属性
例如: 多值属性: *phone_numbers*
 - ◇ 派生 (Derived) 属性: 可以从别的相关属性或实体派生出来
例如: *age, given date_of_birth*
- 参与
 - ◆ 实体集之间的关联称为参与, 即实体参与联系
- 约束
 - 映射基数
 - ◆ 表示一个实体通过一个联系集能关联的实体的个数
 - ◆ 分类
 - ◇ 一对一 (One to one)
 - ◇ 一对多 (One to many)
 - ◇ 多对一 (Many to one)
 - ◇ 多对多 (Many to many)
 - 码
 - ◆ 实体集 (请参见第二章同名部分)
 - ◆ 联系集
 - ◇ 参与实体集合的主键组合形成联系集的超码
 - ◇ 这意味着一对实体在特定联系集中至多有一个关系
- 实体-联系图
 - 主要构件
 - ◆ 分成两部分的矩形代表实体集
 - ◆ 菱形代表联系集
 - ◆ 属性在实体集矩形中列出
 - ◆ 构成主码的属性以下划线标明
 - 基数约束
 - ◆ 我们在所讨论的联系集和实体集之间画一个箭头 (\rightarrow) 或一条线段(—)来表示基数约束
 - ◆ 箭头表示对一、线段表示对多
 - 三元关系
- 弱实体集
 - 定义: 如果一个实体集的所有属性都不足以形成主码, 又没有足够的属性以形成主码的实体集称作弱实体集.
 - 依赖关系: 弱实体集存在依赖于标识实体集
 - 基数约束关系: 标识性联系是从弱实体集到标识实体集多对一的, 并且弱实体集在联系中的参与是全部的
 - 标识: 关联弱实体集和标识性强实体集的联系集即标识性联系以双菱形表示
 - 分辨符:
 - ◆ 弱实体集的分辨符是使得我们可以区分依赖于特定强实体集的弱实体集中的实体的属性的集合.

- ◆ 弱实体集的主码是有标识实体集的主码加上该弱实体集的分辨符构成
- ◆ 弱实体集的分辨符以虚下划线标明，而不是实线
- 转换为关系模式
 - 用具有 n 个不同属性的模式 E 来表示强实体集
 - 用包含标识性强实体集的主键作为列构成的表来表示弱实体集
 - 多对一和一对多联系集中全部参与的“many”一方可以通过增加额外的属性来表示，包含“one”一方的主键
- 扩展的 E-R 特性
 - 特化与概化
 - ◆ 属性继承：超类与子类
 - ◆ 约束
 - ◇ 不相交与有重叠
 - ◇ 完全性约束：全部概化与部分概化
 - 聚集
 - ◆ 将关系作为抽象实体
 - ◆ 允许关系间存在关系
 - ◆ 将关系抽象用于新实体
- 符号总结





Chapter 8

- 基本概念
 - 函数依赖与多值依赖
 - ◆ 函数依赖规定某些元组不能出现在关系中，也称为相等产生依赖；多值依赖要求某种形式的其它元组必须在关系中，称为元组产生依赖。
 - ◆ $X \rightarrow Y$ 的有效性仅决定于 X 、 Y 属性集上的值； $X \twoheadrightarrow Y$ 的有效性与属性集范围有关
 - 有损分解和无损分解：是否丢失了信息，是否能重建原始的关系
- 原子域：域元素被认为是不可分的单元，称域是原子的
- 第一范式：称一个关系模式 R 属于第一范式，如果 R 的所有属性的域都是原子的

- 函数依赖
 - 考虑一个关系模式 R $\alpha \subseteq R$ and $\beta \subseteq R$
 函数依赖 $\alpha \rightarrow \beta$ 满足的条件是对实例中所有元组对 t_1 和 t_2 ，

$$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$
 如果 $r(R)$ 的每个合法实例都满足函数依赖 $\alpha \rightarrow \beta$ ，则我们说该函数依赖在模式 $r(R)$ 上成立 (hold)
 - 和码的关系
 - ◆ 如果函数依赖 $K \rightarrow R$ 在 $r(R)$ 上成立，则 K 是 $r(R)$ 的一个超码
 - ◆ K 是 R 的候选码 当且仅当 $K \rightarrow R$ ，且不存在 $\alpha \subset K, \alpha \rightarrow R$
 - 使用
 - ◆ 判定关系的实例是否满足给定函数依赖集 F ，如果一个关系实例 r 在函数依赖集 F 下是合法的，那么 r 满足 F
 - ◆ 说明合法关系集上的约束，如果 R 上的所有合法关系都满足函数依赖集 F ，那么 F 在 $r(R)$ 上成立
 - 平凡的依赖：有些函数依赖称为平凡的，因为他们在所有关系中都满足以下条件：
 如果 $\beta \subseteq \alpha$ ， $\alpha \rightarrow \beta$ 是平凡的
 - 闭包
 - ◆ 给定函数依赖集 F ，必定有一些其他的函数依赖被 F 逻辑蕴涵
 例如：如果 $A \rightarrow B$ 且 $B \rightarrow C$ ，那么可以推断 $A \rightarrow C$
 - ◆ 能够从给定 F 集合推导出的所有函数依赖的集合称为 F 的闭包
 - ◆ 使用 F^+ 符号来表示 F 集合的闭包
 - ◆ F^+ 是 F 的一个超集
 - ◆ 计算 F^+ 的过程：运用自反律、增补律、传递律
 - ◆ 属性集的闭包：对于函数依赖不断取并集
 - Armstrong 公理
 - ◆ if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ (自反律)

- ◆ if $\alpha \rightarrow \beta$, then $\gamma \alpha \rightarrow \gamma \beta$ (增补律)
- ◆ if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ (传递律)
- ◆ 若 $\alpha \rightarrow \beta$ 和 $\alpha \rightarrow \gamma$ 成立, 则 $\alpha \rightarrow \beta\gamma$ 成立 (合并律)
- ◆ 若 $\alpha \rightarrow \beta\gamma$ 成立, 则 $\alpha \rightarrow \beta$ 和 $\alpha \rightarrow \gamma$ 成立 (分解律)
- ◆ 若 $\alpha \rightarrow \beta$ holds 和 $\gamma \beta \rightarrow \delta$ 成立, 则 $\alpha \gamma \rightarrow \delta$ 成立 (伪传递律)

■ 正则覆盖

- ◆ 定义: F 的正则覆盖是一个与 F 相等的最小的函数依赖集, 没有冗余的依赖, 也没有冗余的函数依赖部分
- ◆ 具体条件: 满足下列条件的函数依赖集 F 称为正则覆盖, 记作 F_c :
 - ◇ 1) F_c 与 F 等价
 - ◇ 2) F_c 中任何函数依赖都不含无关属性
 - ◇ 3) F_c 中函数依赖的左半部都是唯一的
- ◆ 无关属性: 考虑函数依赖集 F 及其中的函数依赖 $\alpha \rightarrow \beta$.
 - ◇ 如果 $A \in \alpha$ 并且 F 逻辑蕴涵 $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$, 则称该属性 A 在 α 中是无关的.
 - ◇ 若果 $A \in \beta$ 并且函数依赖集 $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ 逻辑蕴涵 F , 则属性 A 在 β 中是无关的.
- ◆ 性质: F 的正则覆盖 F_c 是一个依赖集
 - ◇ F 逻辑蕴涵 F_c 中的所有依赖
 - ◇ F_c 逻辑蕴涵 F 中的所有依赖
 - ◇ F_c 中任何函数依赖都不含无关属性
 - ◇ F_c 中函数依赖的左半部都是唯一的
- ◆ 计算函数依赖集 F 的正则覆盖

repeat

使用合并律将 F_c 中所有形如 $\alpha_1 \rightarrow \beta_1$ 和 $\alpha_1 \rightarrow \beta_2$ 的依赖替换为 $\alpha_1 \rightarrow \beta_1 \beta_2$

在 F_c 中寻找一个函数依赖 $\alpha \rightarrow \beta$, 它在 α 或 β 中具有一个无关属性 /*

Note: 使用 F_c , 而非 F * 检验无关属性/

如果找到无关属性, 则将它从 F_c 中的 $\alpha \rightarrow \beta$ 中删除

until (F_c 不变)

- ◆ 正则覆盖未必唯一

■ 无损分解构建

- ◆ 对于 $R = (R_1, R_2)$, 我们要求对于 R 上所有可能的关系 r , 有

$$r = \Pi_{R_1}(r) \quad \Pi_{R_2}(r)$$

将 R 分解成 R_1 和 R_2 , 若 F^+ 中存在以下至少一个依赖, 则是无损分解:

◇ $R_1 \cap R_2 \rightarrow R_1$

◇ $R_1 \cap R_2 \rightarrow R_2$

以上的函数依赖是无损分解的充分条件; 这些依赖是必要条件当且仅当所有约束都是函数依赖

■ 保持依赖

- ◆ 令 F_i 函数依赖集 F^+ 只包含 R_i 中属性的函数依赖集合
- 分解 保持依赖, 如果

$$(F_1 \cup F_2 \cup \dots \cup F_n)^+ = F^+$$

- Boyce-Codd 范式
 - 定义: A 具有函数依赖集 F 的关系模式 R 属于 BCNF 的条件是, 对 F^+ 中所有形如 $\alpha \rightarrow \beta$ 的函数依赖 (其中 $\alpha \subseteq R$ 且 $\beta \subseteq R$), 下面至少有一项成立
 - ◆ $\alpha \rightarrow \beta$ 是平凡的函数依赖 (即 $\beta \subseteq \alpha$)
 - ◆ α 是模式 R 的一个超码
 - 将模式分解为 BCNF
 - ◆ 假设有一关系模式 R 和一个非平凡依赖 $\alpha \rightarrow \beta$ 违反 BCNF:
 - 将 R 分解
 - $(\alpha \cup \beta)$
 - $(R - (\beta - \alpha))$
 - ◆ 其分解后不能保持依赖
- 第三范式【不存在传递依赖】
 - 定义: 关系模式 R 属于第三范式(3NF)的条件, 对于所有 $\alpha \rightarrow \beta$ in F^+ 以下至少一项成立:
 - $\alpha \rightarrow \beta$ 是一个平凡的函数依赖 (i.e., $\beta \subseteq \alpha$)
 - α 是 R 的一个超码
 - $\beta - \alpha$ 中的**每个属性** A 都包含于 R 的一个候选码中.
 - 与 BCNF 的关系
 - ◆ 任何满足 BCNF 的模式也满足 3NF (因为它的每个函数依赖都将满足前两个条件中的一条)
 - ◆ 在某种意义上, 3NF 的第三个条件代表 BCNF 条件的最小放宽, 以确保每一个模式都有保持依赖的 3NF 分解
 - 与 BCNF 的特点分析
 - ◆ 3NF 的优点
 - ◇ 满足无损分解
 - ◇ 保持依赖
 - ◆ BCNF 的特点
 - ◇ 无损分解
 - ◇ 可能不满足保持依赖
- 分解算法
 - BCNF 分解
 - ◆ 为了检查非平凡的函数依赖 $\alpha \rightarrow \beta$ 是否违反 BCNF
 - ◇ 计算 α^+ (α 的属性闭包)
 - ◇ 验证它是否包含 R 中的所有属性, 即验证它是否是 R 的超码
 - ◆ 简单验证: 检查关系模式 R 是否属于 BCNF, 仅需检查给定集合 F 中的函数依赖是否违反 BCNF 就足够了, 不用检查 F^+ 中的所有函数依赖
 - 3NF 分解 (较为复杂, 请参阅书本)
- 多值依赖
 - 定义: 令 R 为一关系模式, 并令 $\alpha \subseteq R$ 且 $\beta \subseteq R$. 多值依赖 $\alpha \twoheadrightarrow \beta$ 在 R 上成立的条件是, 在关系 $r(R)$ 的合法实例中, 对于 r 中任意一对满足 $t_1[\alpha] = t_2[\alpha]$ 的元组对 t_1 和 t_2 , r 中都存在元组 t_3 和 t_4 , 使得:

$$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$$

$$\begin{aligned}
t_3[\beta] &= t_1[\beta] \\
t_3[R - \beta] &= t_2[R - \beta] \\
t_4[\beta] &= t_2[\beta] \\
t_4[R - \beta] &= t_1[R - \beta]
\end{aligned}$$

- 规则：由多值依赖的定义，我们可以得出以下规则，对于 $\alpha, \beta \subseteq R$
 - ◆ 若 $\alpha \rightarrow \beta$, 则 $\alpha \twoheadrightarrow \beta$
 - ◆ 即每一个函数依赖也是一个多值依赖
- 闭包 D^+
- 第四范式
 - 定义：函数依赖和多值依赖集为 D 的关系模式 $r(R)$ 属于4NF的条件是，对 D^+ 中所有形如 $\alpha \twoheadrightarrow \beta$ 的多值依赖(其中 $\alpha \subseteq R$ 且 $\beta \subseteq R$),至少有以下之一成立
 - ◆ $\alpha \twoheadrightarrow \beta$ 是一个平凡的多值依赖(i.e., $\beta \subseteq \alpha$ or $\alpha \cup \beta = R$)
 - ◆ α 是 R 的一个超码

Chapter 10

- 文件与记录的组织
 - 定长记录
 - 变长记录
 - 顺序文件组织
 - 多表聚簇文件组织
- 数据字典：(也称为系统目录，用于存储元数据，即关于数据的数据)
- 索引概念
 - 搜索码：用于在文件中查找记录的属性或属性集
 - 索引建立原则：5%原则（查询结果的范围能缩小到5%,那么建立索引才有实质性的效果）
 - 索引类型
 - ◆ 顺序索引
 - 基于搜索码值的顺序排序
 - ◆ 散列索引
 - 基于将值平均分布到若干散列桶中
 - 一个值所属的散列桶是由一个函数决定，该函数称为散列函数
 - 顺序索引
 - ◆ 主索引（聚集索引）：包含记录的文件按照某个搜索码指定的顺序排序，那么该搜索码对应的索引称为主索引
 - ◆ 辅助索引（非聚集索引）
 - 定义：搜索码指定的顺序与文件中记录的物理顺序不同的索引
 - 实现：索引记录指向一个包含所有具有特定搜索码值的实际记录的指针的桶，另外辅助索引必须是稠密索引
 - ◆ 稠密索引：在稠密索引中，文件中的每个搜索码值都有一个索引项
 - ◆ 稀疏索引：在稀疏索引中，只为搜索码的某些值建立索引项；并且只有索引是聚集索引时才能使用稀疏索引

- 散列索引
 - ◆ 散列桶
 - 定义: 表示能存储一条或多条记录的一个存储单位, 通常一个桶就是一个磁盘块
 - 生成: 在散列文件组织中可以直接从搜索码使用散列函数得到包含记录的桶
 - 映射: 带有不同搜索码值的记录可以映射到同一个桶中, 因此, 为了找到一个记录, 桶要被依次检索
 - 桶溢出
 - ◆ 散列函数
 - 定义: 散列函数 h 是一个从所有搜索码值的集合 K 映射到桶地址的集合 B 的函数
 - 作用: 散列函数用来定位要访问, 插入或删除的记录
 - ◆ 散列索引: 将搜索码及其相应的指针组织成散列文件结构
 - ◆ 与顺序索引的比较
 - 对于具有指定码值的记录检索, 散列是一个更好的选择
 - 如果经常进行范围查询, 顺序索引是首选

Chapter11

- 查询优化: 在所有等效执行计划中选择具有最小查询执行代价的计划
- 关系代数运算的执行
 - 选择运算
 - 连接运算
 - ◆ 嵌套循环连接
 - ◆ 块嵌套循环连接
 - ◆ 索引循环嵌套连接
 - ◆ 归并连接
 - ◆ 散列连接
 - 表达式计算
 - ◆ 物化: 输入一个关系或者已完成的计算, 产生一个表达式的结果, 在磁盘中物化它, 重复该过程
 - ◆ 流水线: 一个正在执行的操作的部分结果传送到流水线的下一个操作, 使得两操作可同时进行
 - 查询优化
 - ◆ 等价规则

自然连接运算满足结合律

$$\diamond (E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

投影运算在下列条件下对 θ 连接运算具有分配率

(a) 假设连接条件 θ 只涉及 $L_1 \cup L_2$ 中的属性

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_{\theta} E_2) = (\Pi_{L_1}(E_1)) \bowtie_{\theta} (\Pi_{L_2}(E_2))$$

(b) 考虑连接 $E_1 \bowtie_{\theta} E_2$

- 令 L_1 和 L_2 分别代表 E_1 和 E_2 的属性集
- 令 L_3 是 E_1 中出现在连接条件 θ 中但不在 $L_1 \cup L_2$ 中的属性
- 令 L_4 是 E_2 中出现在连接条件 θ 中但不在 $L_1 \cup L_2$ 中的属性

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_{\theta} E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_{\theta} (\Pi_{L_2 \cup L_4}(E_2)))$$

◇

◇ 一个例子

查询: 找到 2009 年在音乐系讲授课程的所有教师的名字, 以及他们所教课程
的名称

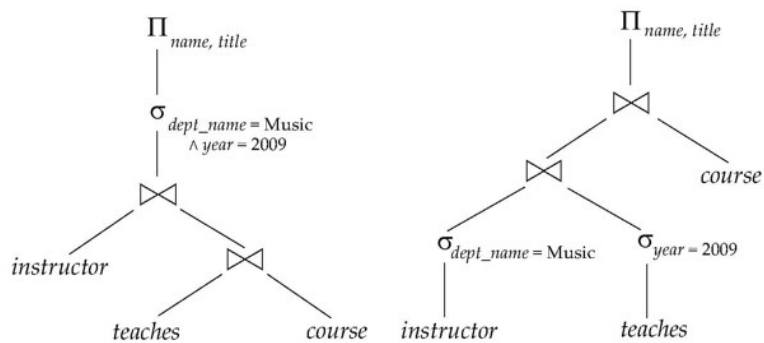
$$\Pi_{name, title}(\sigma_{dept_name = \text{“Music”} \wedge year = 2009}(\text{instructor} \text{ teaches } \Pi_{course_id, title}(\text{course})))$$

使用连接的结合律转换 (规则 6a)

$$\Pi_{name, title}(\sigma_{dept_name = \text{“Music”} \wedge year = 2009}((\text{instructor} \text{ teaches}) \Pi_{course_id, title}(\text{course})))$$

第二种形式提供了一个应用“尽早执行选择”规则的机会, 得到如下子表达式

$$\sigma_{dept_name = \text{“Music”}}(\text{instructor}) \quad \sigma_{year = 2009}(\text{teaches})$$



(a) Initial expression tree

(b) Tree after multiple transformations

执行图

Chapter12

● 事务概念

- 事务(transaction)是访问并可能更新各种数据项的一个程序执行单元(Unit)
- 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) 事务的隔离执行必须保证数据库的一致性
 - ◆ 隔离性(Isolation) 系统必须保证事务不受其它并发执行事务的影响
 - ◆ 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

- 事务状态
 - 活动的：初始状态，事务执行时处于这个状态
 - 部分提交的：最后一条语句执行后
 - 失败的：发现正常的执行不能继续之后
 - 中止的：事务回滚并且数据库已恢复到事务开始执行前的状态后。此时，系统有两种选择：
 - ◆ 重启事务：引起事务终止是由于硬件错误或者不是由事务内部逻辑所产生的软件错误
 - ◆ 杀死事务：由于事务的内部逻辑错误引起，需要重写应用程序
 - 提交的：成功完成之后
- 并发执行
 - 优势
 - ◆ 提高吞吐量和资源利用率。可以得到更高的事务吞吐量
 - ◆ 减少等待时间。短事务不需要等待前面的长事务完成
 - 并发控制机制 - 保证隔离性的一系列机制
控制事务之间的交互，以防止它们破坏数据库的一致性
- 调度(Schedule) - 表示指令在系统中执行的顺序
 - 串行调度
 - 并行调度
- 可串行化
 - 如果一个（可能并发的）事务等价于一个串行调度，则该调度是可串行化的
 - 冲突

令 l_i 和 l_j 分别为事务 T_i 和 T_j 的指令， l_i 和 l_j 冲突当且仅当存在一些数据项 Q ， l_i 和 l_j 都访问 Q ，并且至少其中之一是对 Q 的写操作

 - ◆ 1. $l_i = \text{read}(Q), l_j = \text{read}(Q)$. l_i 和 l_j 不冲突
 - ◆ 2. $l_i = \text{read}(Q), l_j = \text{write}(Q)$. 冲突
 - ◆ 3. $l_i = \text{write}(Q), l_j = \text{read}(Q)$. 冲突
 - ◆ 4. $l_i = \text{write}(Q), l_j = \text{write}(Q)$. 冲突
 - 冲突可串行化
 - ◆ 如果通过一系列非冲突指令的交换，调度 S 可以转换为调度 S' ，我们说 S 和 S' 是冲突等价(conflict equivalent)的
 - ◆ 我们说调度 S 是冲突可串行化(conflict serializable)的，如果它与一个串行调度冲突等价
- 可恢复性
 - 可恢复的调度(Recoverable schedule) — 如果事务 T_j 读取了 T_i 所写的数据项,则 T_i 先于 T_j 提交
 - 级联回滚——单个事务失败，导致一系列事务回滚。考虑下列调度，其中事务都中止(调度是可恢复的)
 - 无级联调度(Cascadeless schedules) — 不会发生级联回滚；对每一事务对 T_i 和 T_j ，如果 T_j 读取了 T_i 所写的数据项, 则 T_i 必须在 T_j 的读操作之前提交
- 并发控制
 - 基于锁的协议
 - ◆ 锁分类
 - ◇ 1.exclusive(X)模式。数据项既可以读又可以写。使用 lock-X 指令申请

- ◇ 2.shared(S)模式。数据项是只读的。使用 lock-S 指令申请
- ◆ 封锁协议是在请求和释放锁时的一系列规则，适用于所有事务。封锁协议限制了可能的调度集
- ◆ 不足
 - ◇ 死锁
 - ◇ 饥饿
- ◆ 两阶段封锁协议

这是一个确保冲突可串行化调度的协议

 - ◇ 阶段 1: 增长阶段
 - 事务可以获取锁
 - 事务不能释放锁
 - ◇ 阶段 2: 收缩阶段
 - 事务可以释放锁
 - 事务不能获取锁
 - ◇ 协议确保可串行化。可以证明事务按照它们的封锁点的顺序可串行化 (即事务获取最后一次封锁的时刻)
 - ◇ 两阶段封锁不能确保避免死锁，可以保证冲突可串行化
 - ◇ 强两阶段封锁协议更加严格：事务提交或终止前，不能释放任何锁。在该协议中，事务按照提交的顺序可串行化
- ◆ 等待图（有环即冲突不可解）
- ◆ 意向锁
- 死锁
 - ◆ 意义：在 DBS 运行时，死锁状态是我们不希望发生的，因此死锁的发生本身是一件坏事。但是坏事可以转换为好事。如果我们不让死锁发生，让事务任意并发做下去，那么有可能破坏 DB 中数据，或用户读了错误的数据库。从这个意义上讲，死锁的发生是一件好事，能防止错误的数据库发生。
 - ◆ 处理与恢复：在发生死锁后，系统的死锁处理机制和恢复程序就能起作用，抽取某个事务作为牺牲品，把它撤消，做 ROLLBACK 操作，使系统有可能摆脱死锁状态，继续运行下去。
 - ◆ 预防：死锁预防协议确保系统不会进入死锁状态，一些预防策略：
 - ◇ 要求每个事务在执行前封锁所有的数据项（预声明）
 - ◇ 给所有数据项施加偏序关系，要求事务按照偏序关系封锁数据项(基于图的协议)
- 基于时间戳的协议
 - ◆ W-timestamp(Q): 表示成功执行 write(Q)的所有事务的最大的时间戳
 - ◆ R-timestamp(Q): 表示成功执行 read(Q)的所有事务的最大的时间戳
- 恢复系统
 - 操作
 - ◆ input(B) 将物理块 B 传入主存.
 - ◆ output(B) 将缓冲块 B 传到磁盘, 并且替换相应的物理块
 - ◆ read(X) 将数据项 X 的值赋给局部变量 xi.
 - ◆ write(X) 将局部变量 xi 的值赋给缓冲块中的数据项 {X}.
 - ◇ 注意: output(BX) 不必紧随 write(X)立即执行. 系统可在它认为适当的时候执行 output 操作

- 基于日志的恢复
 - ◆ 日志记录
 - ◇ $\langle T_i \text{ start} \rangle$
 - ◇ $\langle T_i, X, V_1, V_2 \rangle$
 - ◇ $\langle T_i \text{ commit} \rangle$
 - ◆ 日志方法
 - ◇ 立即修改技术允许在事务提交之前，对还未提交的事务进行更新
 - ◇ 延迟修改技术只有在事务提交时才执行更新
 - ◆ 事务的 undo/redo
 - ◇ $\text{undo}(T_i)$ 将事务 T_i 更新过的所有数据项的值都恢复成旧值，从 T_i 的最后一条日志记录向前进行
每次数据项 X 恢复到它的旧值 V ，就写下一个特别的日志记录 $\langle T_i, X, V \rangle$
当事务的 undo 操作完成后，写下一个日志记录 $\langle T_i \text{ abort} \rangle$
 - ◇ $\text{redo}(T_i)$ 将所有 T_i 更新过的数据项的值都设置成新值，从 T_i 的第一条日志记录向后进行
不产生日志
 - ◆ 检查点