数据库系统原理考试说明如下：

1）闭卷考试，可携带计算器。

2）中文试题，三个老师统一试卷。

3）题目类型包括选择题、填空题、判断题、简答题（概念）、计算题（范式）、画图题（ER图）等。

4）理论课不考实验课内容。

5）题目难度系数为中等、题量不大，够两个小时。

6）平时成绩占30%。缺交的作业要在第19周周日前补齐。

18:

> 选择题20分 填空题20分
>
> 概念
>
> - [x] 数据字典、元数据
> - [x] 独立性
> - [x] DBMS等全称含意 DS SQL
>
> 关系代数
>
> - [x] 关系代数运算 6个 只属于关系代数中的运算（投影 选择 连接 除法）
> - [x] 关系代数还要会**写** 基本符号会用 默写记下
>
> SQL
>
> - [x] 基本的SQL语言
> - [x] 视图创建和概念
> - [x] 连接操作重点
> - [x] 完整性约束 参照完整性约束基本概念 外码约束是特殊参照完整性约束 参照方和被参照方违反和维护（一定看看）
> - [x] 授权 被授权 允许传递出去 角色授权
>
> ER模型
>
> - [x] 数据库的设计过程（五个步骤记住）
> - [x] 画法 按课本规则画否则扣分
> - [x] 映射基数 映射约束
> - [x] 有了ER逻辑设计转换为概念模式6.7 6.9
> - [x] 弱实体集也要掌握
>
> 范式

- [x] 函数依赖 概念需要掌握 函数依赖概念
- [x] 求 判断 推导 范式（1 BCNF 3 4）
- [x] BCNF判断分解
- [x] 3范式判断分解 不需要知道为什么这么分解
- [x] 求属性闭包 计算量不会太大 简单算一算 和作业差不多 判断是不是超码 候选码
- [x] 各种范式的冗余度排序

索引

- [x] B+树
- [x] 索引基本概念
- [x] 主索引 辅助索引

查询处理

查询优化

计算量只要会算transfer 不用算seek（没有统一计算标准）重点掌握课后习题那几种（嵌套循环、块嵌套循环） ？ 赖老师没讲

- [x] 重点掌握等价表达式基本的计算规则 16条

17 18 19

- [x] ACID分别指什么
- [x] 冲突可串行化、有向图判断
- [x] 死锁的概念和产生
- [x] 加锁lock 两阶段加锁 严格-
- [x] 日志的基本概念、作用
- [x] 恢复算法（最基本的就行了）

19:

Introduction

Meta-data/data dictionary concept

DDL以一些指令（语句）作为输入，生成一些输出。

DDL的输出放在数据字典中，数据字典包含了元数据，元数据是关于数据的数据。

可把数据字典看作一种特殊的表，这种表只能由数据库系统本身来访问和修改，在读取和修改实际的数据前，数据库系统先要参考数据字典。

数据字典是指对数据的数据项、数据结构、数据流、数据存储、处理逻辑等进行定义和描述，其目的是对数据流程图中的各个元素做出详细的说明，使用数据字典为简单的建模项目。简而言之，**数据字典是描述数据的信息集合，是对系统中使用的所有数据元素的定义的集合。**

元数据为描述数据的数据（data about data），主要是描述数据属性（property）的信息。

一个关系数据库系统需要维护关于关系的数据，如关系模式等，这样“关于数据的数据”称为元数据。关于关系的关系模式和其他元数据存储在称为数据字典或系统目录的结构中。

Physical/logical data independence concept

虽然逻辑层的简单结构的实现可能涉及复杂的物理层结构，但逻辑层的用户不必知道这样的复杂性，这称作物理数据独立性。

物理独立性是指**用户的应用程序**与**存储在磁盘上的数据库中数据**是相互独立的。即，数据在磁盘上怎样存储由DBMS管理，用户程序不需要了解，应用程序要处理的只是数据的逻辑结构，这样当数据的物理存储改变了，应用程序不用改变。

逻辑独立性是指**用户的应用程序**与**数据库的逻辑结构**是相互独立的，即，当数据的逻辑结构改变时，用户程序也可以不变。

意义 数据与程序的独立，把数据的定义从程序中分离出去，加上数据的存取又由DBMS负责，从而简化了应用程序的编制，大大减少了应用程序的维护和修改。

DDL & DML

DDL语言（Data-Definition Language）

数据库定义功能，这些语句定义了数据库模式的实现细节，对用户来说通常**不可见**

存储在数据库的数据值必须满足某些**一致性约束**，例如，假设大学要求一个系的账户余额不能为负值，DDL提供了指定这种约束的**工具**。

域约束 check 每当有新数据插入，约束检测

参照完整性 reference 拒绝破坏参照完整性的操作

断言 只有不破坏断言的数据库更新才被允许 （域约束和参照完整性不能表达的约束）

授权 对用户加以区别，赋予不同的权限

DML语言（Data-manipulation language）

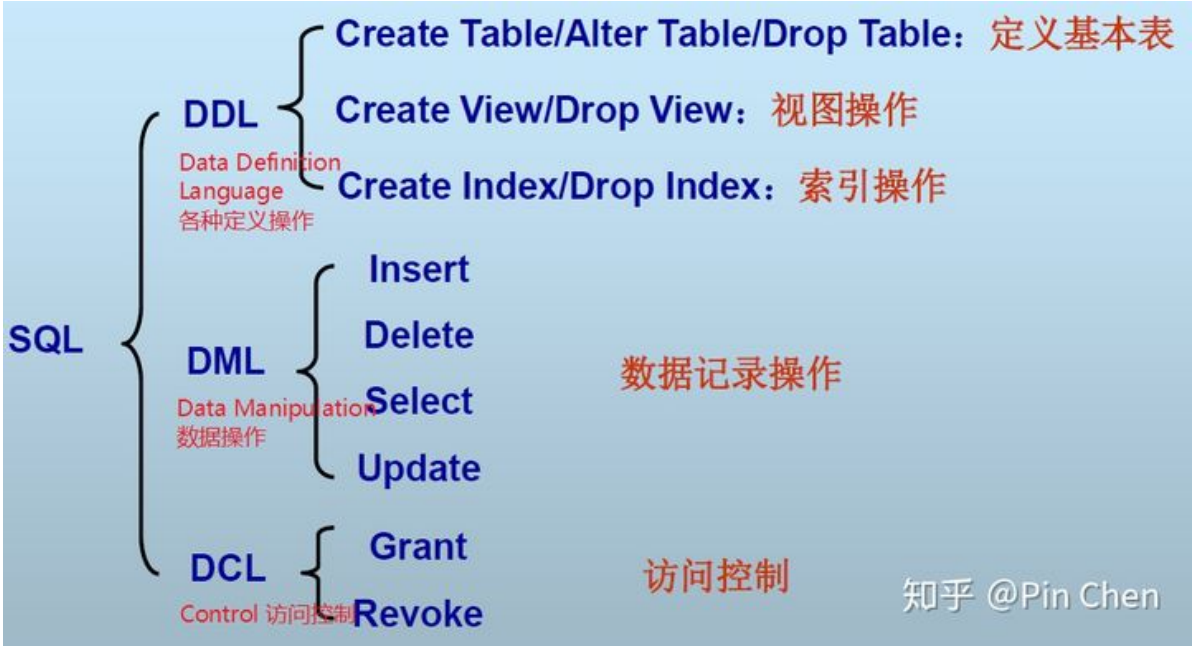
数据操纵语言，使得用户可以访问或操纵那些按照某种适当的数据模型组织起来的数据。

对数据库进行检索、插入、修改、删除

两类：过程化DML、声明式DML

过程化与非过程化语言的相对优点
非过程化语言在特定领域或执行特定功能上对程序员更方便友好，如 SQL 用于查询数据库中的数据等等，减少了程序员的工作量，且其实现方式一般经过优化，比直接手写更加高效。

过程化语言能实现的功能更加广泛，对于一些过程化语言比较难表达的任务也可以有效执行。



E.F. (Ted) Codd

> Edgar F. Codd（通常被称为Ted）是一个才华横溢的人。他的成就之一，是在二十世纪七十年代初开发了一个关系型数据管理模型--存储和操作大量业务数据的一个复杂、完整的理论。根据Codd的设计构建的关系数据库成为了当今企业的基础；银行依赖关系数据库来跟踪资金流动；零售商使用它们来监控库存水平；人力资源部门使用它们来管理员工账户；图书馆、医院和政府机构在其中存储数百万条记录；事实上，世界上几乎所有的企业都在使用某种容量的关系数据库。自从Codd公布其理论以来的30年中，关系数据库已经成为一个年收入近130亿美元的行业。

SQL

> DDL、DML
>
> 完整性
>
> 视图定义
>
> 事务控制
>
> 嵌入式SQL和动态SQL（web编程?
>
> 授权

Writing SQL statements for basic queries

> 基本查询

Having & Group by **分组聚集**

> **Group by** 在group by子句中**所有属性**上取值相同的元组被分在同一个组中
>
> **having** having子句中的谓词在形成分组后才起作用
>
> 计算流程：

1. 先根据from计算一个关系
2. 如果有where，根据where过滤元组，筛选出一个关系
3. 如果有group by，进行分组
4. 如果有having 对每个分组应用having
5. select利用最后剩下的分组产生查询结果，对每个分组应用聚集函数

连接 内连接、左、右、全外连接

... join ... using + 属性 自然连接

... join ... on + 表达式

以上连接方式相同属性会重复出现，即便是相同的

不保留未匹配元组的连接运算成为内连接

View concept 视图

概念 使用户看到整个逻辑模型是不合适的，出于安全考虑，可能需要向用户隐藏特定的数据。视图是一种"虚关系"，它在概念上包含查询的结果，并不预先计算存储，而是在使用的时候才通过执行查询被计算出来。

创建视图 create view .. as select ...

物化视图 特定数据库允许存储视图关系，但保证：如果用于定义视图的实际关系改变，视图也跟着改变。

一般不允许对视图进行更新

Foreign key and referential integrity constraints 外键和参照完整性约束

参照完整性 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，这种情况成为参照完整性。

这种要求成为参照完整性约束。

完整性约束保证授权用户对数据库所做的修改不会破坏数据的一致性，因此，完整性约束防止的是对数据的意外破坏。

单个关系上的约束 not null、 unique、 check

事务中对完整性约束的违反 解决方法：1、可延迟的（对指定约束的检查延迟到该事务结束时执行） 2、可为null

创建索引 如果用户提交的SQL查询可以从索引的使用中获益，那么SQL查询处理器就会自动使用索引。

Grant/revoke/with grant option 授权

授权、收回 包括：增删查改，每种类型的授权称为一个权限。

grant ...(...) on ... to ... (with grant option(允许权限传递));

revoke ...(...) on ... from ... (restrict(防止级联收回))

()可以仅对某些属性作用。

角色、角色链，授权图

references权限 外码约束限制了被参照关系上的删除和更新操作

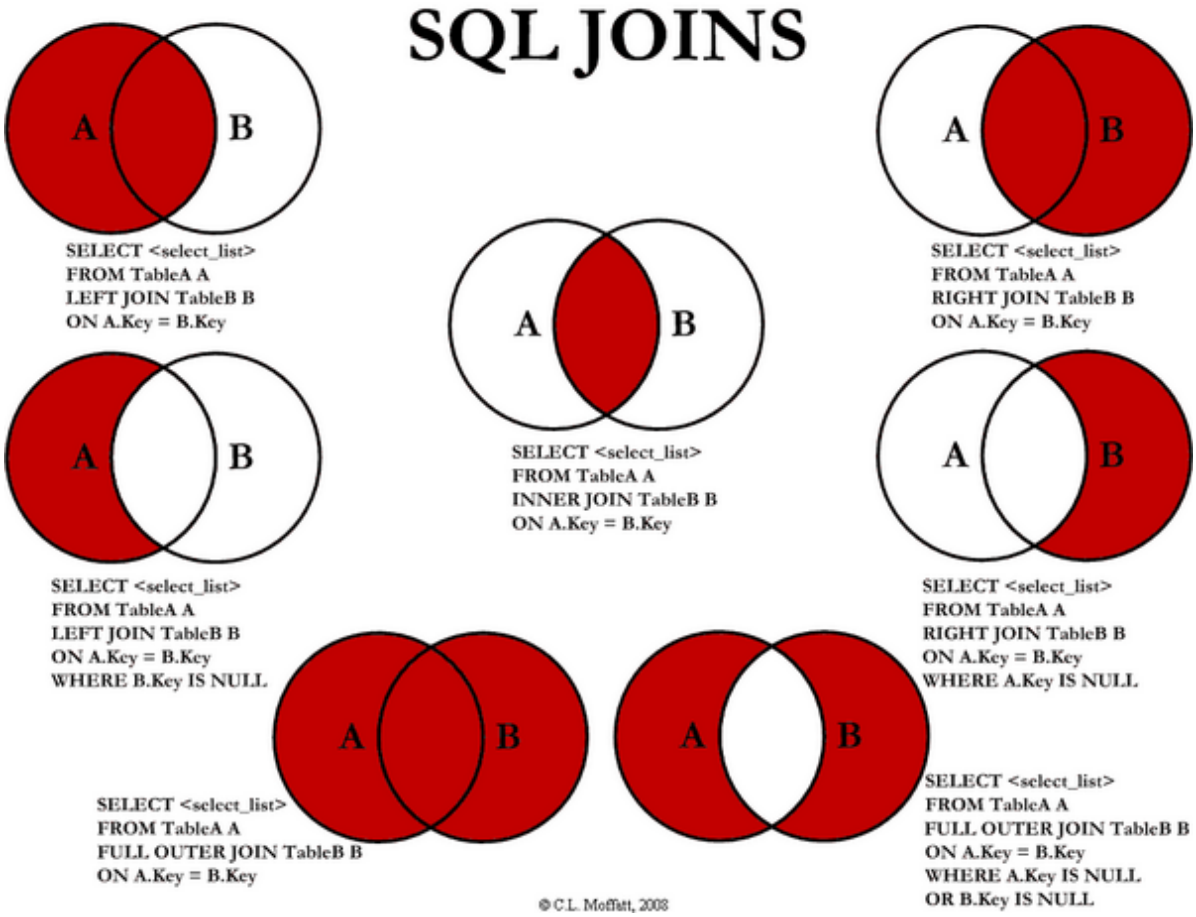
Relational Algebra 关系代数

一元运算：选择、投影、更名

二元运算：并、集合差、笛卡儿积

five basic operations

符号(名字)	使用示例
σ(选择)	$\sigma_{salary >= 85000}(instructor)$
	返回输入关系中满足谓词的行
Π(投影)	$\Pi_{ID,salary}(instructor)$
	对输入关系的所有行输出指定的属性。从输出中去除重复元组
⋈(自然连接)	$instructor \bowtie department$
	从两个输入关系中输出这样的元组对：它们在具有相同名字的所有属性上取值相同
×(笛卡儿积)	$instructor \times department$
	从两个输入关系中输出所有的元组对(无论它们在共同属性上的取值是否相同)
∪(并)	$\Pi_{name}(instructor) \cup \Pi_{name}(student)$
	输出两个输入关系中元组的并



Writing relational algebra expressions for basic queries

第2、6章习题

1、基本运算

选择运算 σ

投影 $\prod$

关系运算组合（）

并运算，符号 $\cup$

$\prod_{course\_id}(\sigma_{semester=〃Fall〃 \wedge year=2009}(section))$

$\prod_{course\_id}(\sigma_{semester=〃Fall〃 \wedge year=2009}(section)) \wedge \prod_{course\_id}(\sigma_{semester=〃Spring〃 \wedge year=2010}(section))$（必须保证做并运算的关系是相容的）

差运算，符号 $-$

$\prod_{course\_id}(\sigma_{semester=〃Fall〃 \wedge year=2009}(section)) - \prod_{course\_id}(\sigma_{semester=〃Spring〃 \wedge year=2010}(section))$

笛卡儿积运算 符号$\times$

$\sigma_{instructor.ID=teaches.ID}(\sigma_{dept\_name=〃Physics〃}(instructor \times teaches))$

更名运算 符号ρ

假设关系代数表达式E是n元的 $\rho_{x(A_1,A_2,...,A_n)}(E)$

利用更名运算可以计算出非最高工资，再用差运算、投影找出最高工资

2、附加的关系代数运算

集合交运算 $\cap$

自然连接 $\bowtie$ 要求笛卡儿积的两个关系在所有相同属性上的值一致

E-R model ER模型

5 steps of DB design 数据库设计的5个步骤

第一阶段：用户需求分析

进行数据库设计首先必须准确了解和分析用户需求（包括数据与处理）。

第二阶段：概念设计

概念结构设计是整个数据库设计的关键，它通过对用户需求进行综合，归纳与抽象，形成了一个独立于具体DBMS的概念模型。

- 将用户需求转化为选定的某种概念模型（常见是E-R模型）
- 结果：得到(以E-R模型表示的)数据库概念模式

功能需求规格说明

第三阶段：逻辑设计

逻辑结构设计是将概念结构转换为某个DBMS所支持的数据模型，并将进行优化。

- 将概念模式转化为选定的某种数据模型（常见是关系模型）

第四阶段：物理设计

物理设计是为逻辑数据结构模型选取一个最适合应用环境的物理结构（包括存储结构和存取方法）。

E-R diagram construction/ logical design E-R图构建与逻辑设计

三个基本概念

实体集 相同类型即具有相同性质（或属性）的一个实体集合。

联系集 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。

属性 单值、多值、派生

基本结构

- 分成两部分的矩形代表实体集，有阴影的包含实体集的名字，另一部分包含实体集中所有属性的名字。
- 菱形代表联系集
- 未分割的矩形代表联系集的属性
- 线段将实体集连接到联系集
- 虚线将联系集属性连接到联系集
- 双线显示实体在联系集中的参与度
- 双菱形代表连接到弱实体集的标志性联系集

Mapping cardinality 映射基数

映射基数表示一个实体通过一个联系集能关联的实体个数，必然是以下情况之一：

1. 一对一
2. 一对多
3. 多对一
4. 多对多

“一”的一方有箭头，“多”的没有

a) 一对一

b) 一对多

c) 多对多

例如，考虑图 7-10，在 *advisor* 和 *student* 之间的边有 1..1 的基数约束，意味着基数的最小值和最大值都是 1。也就是，每个学生必须有且仅有一个导师。从 *advisor* 到 *instructor* 边上的约束 0..* 说明教师可以有零个或多个学生。因此，*advisor* 联系是从 *instructor* 到 *student* 的一对多联系，更进一步地讲，*student* 在 *advisor* 联系中的参与是全部的，表示一个学生必须有一个导师。

图 7-10　联系集上的基数约束

弱实体集

没有足够的属性以形成主码的实体集称作弱实体集，有主码的实体集称作强实体集。

弱实体集必须与另一个称作标识或属主实体集的实体集关联才能有意义，每个弱实体必须和一个标识实体关联，也就是说，弱实体集存在依赖于标识实体集。我们称标识实体集拥有它所标识的弱实体集。将弱实体集与其标识实体集相联的联系称为标识性联系。

图 7-14　包含弱实体集的 E-R 图

- 双线表明全部参与
- 弱实体集的分辨符以虚下划线标明，而不是实线
- 关联弱实体集和标识性强实体集的联系集以双菱形表示

大学E-R图：

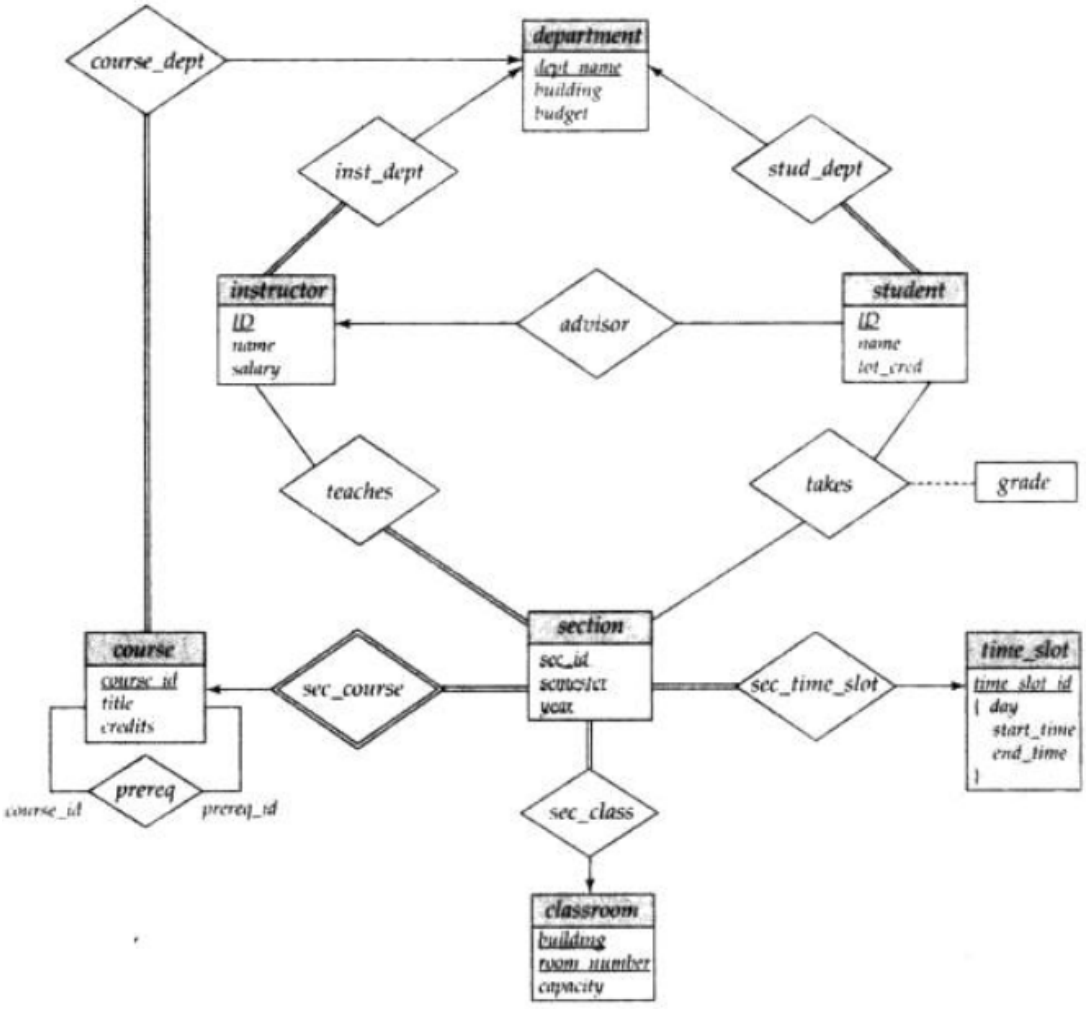


图 7-15　大学的 E-R 图

有了E-R图后转化为关系模式

具有简单属性的强实体集的表示 主码

具有复杂属性的强实体集的表示 属性展开 （多值属性需要再建立新的关系，添加外码约束）

弱实体集的表示 弱实体集属性 ∪ 其所依赖的强实体集的主码

联系集的表示 所有参与R的实体集的主码 ∪ R的描述性属性

- 选择主码
  - 多对多：参与实体集的主码属性的并集
  - 一对一：任意一个实体集的主码
  - 一对多、多对一：“多”的一方的实体集的主码
- 建立外码约束
  - R中来自E主码属性的那些属性参照表示关系模式E的主码

Norms 范式

码 一个属性，一种能区分给定关系中的不同元组的方法

超码 一个或多个属性的集合，能区分给定关系中的不同元组的方法，可能包含无关紧要的属性

候选码 最小的超码

主码 被设计者选中的候选码

外码 参照关系、被参照关系

参照完整性约束 要求在参照关系中的任意元组在特定属性上的取值必然等于被参照关系中某个元组在特定属性上的取值

有损分解、无损分解

原子域 一个域是原子的，如果该域的元素被认为是不可分的单元。（非原子：多值、复杂属性）

1NF 如果R的所有属性的域都是原子的。

函数依赖

如果存在模式（$dept\_name, budget$），则$dept_name$可以作为主码。这条规则被定义为函数依赖。基于函数依赖，数据库设计者可以发现一个模式应拆分或分解成两个或多个模式的情况。

- 给定$r(R)$的一个实例，我们说这个实例满足函数依赖$\alpha \rightarrow \beta$的条件是：对实例中所有元组对t_1和t_2，若$t_1[\alpha] = t_2[\alpha]$，则$t_1[\beta] = t_2[\beta]$。
- 如果在$r(R)$的每个合法实例中都满足函数依赖$\alpha \rightarrow \beta$，则我们说该函数依赖在模式$r(R)$上成立hold。
- 若有$K \rightarrow R$，则K为一个超码。
- 使用符号F^+来表示F集合的闭包，也就是从给定F集合推导出所有函数依赖的集合。F^+包含F中所有的函数依赖。
- 函数依赖被称为**平凡**的，因为它们在所有关系中都满足。通常讨论的是非平凡的。

Armstrong axioms 阿姆斯特朗公理

函数依赖推理的公理体系，包括自反律、增广律和传递律。

设 U为关系模式R的属性集总体，F是U上的一组函数依赖。则对于关系模式R是否为F所蕴含，推理规则如下：

①自反律。若Y⊆X⊆U，则X→Y为F所蕴含。

②增广律。若X→Y为F所蕴含，且Z⊆U，则XZ→YZ为F所蕴含。

③传递律。若X→Y及Y→Z为F所蕴含，则X→Z为F 所蕴含。

④合并律。若$\alpha \to \beta$，$\alpha \to \gamma$成立，则$\alpha \to \beta\gamma$成立

⑤分解率。若$\alpha \to \beta\gamma$成立，则$\alpha \to \beta$，$\alpha \to \gamma$成立

⑥伪传递率。若$\alpha \to \beta$，$\gamma\beta \to \delta$成立，则$\alpha\gamma \to \delta$成立。

Concept of BCNF

我们能达到的较满意的范式之一，它消除所有基于函数依赖能够发现的冗余。

条件 对F^+中所有形如$\alpha \to \beta$的函数依赖（其中$\alpha \subseteq R$且$\beta \subseteq R$），下面至少有一项成立：

- $\alpha \to \beta$是平凡的函数依赖（即$\beta \subseteq \alpha$）
- α是模式R的一个超码

一个数据库设计属于BCNF的条件是，构成该设计的关系模式集中的每个模式都属于BCNF。

证明不属于BCNF 设R为不属于BCNF的一个模式，则存在至少一个非平凡的函数依赖$\alpha \to \beta$，其中α不是R的超码。

BCNF decomposition BCNF分解

设R为不属于BCNF的一个模式，则存在至少一个非平凡的函数依赖$\alpha \to \beta$，其中α不是超码。用以下两个模式取代R：(必背)

- $(\alpha \cup \beta)$
- $(R - (\beta - \alpha))$

Concept of dependency preserving 依赖保持

该 E-R 图指明了“一个学生可以有多位导师，但是对应于一个给定的系最多只有一个 ”的约束。

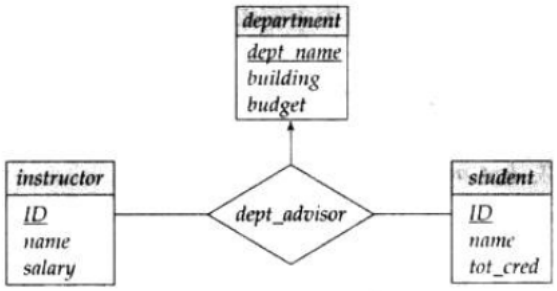


图 8-6　联系集 *dept_advisor*

对应于新的 E-R 图，*instructor*、*department* 和 *student* 的模式没有变。然而，从 *dept_advisor* 导出的模式为：

dept_advisor (*s_ID*, *i_ID*, *dept_name*)

那么，下面的函数依赖在 *dept_advisor* 上成立：

$$i\_ID \rightarrow dept\_name$$
$$s\_ID, dept\_name \rightarrow i\_ID$$

第一个函数依赖产生于我们的需求“一位教师只能在一个系担任导师”。第二个函数依赖产生于我们的需求“对于一个给定的系，一个学生可以有至多一位导师”。

注意，在这种设计中，每次一名教师参与一个 *dept_advisor* 联系的时候，我们都不得不重复一次系的名称。我们看到 *dept_advisor* 不属于 BCNF，因为 *i_ID* 不是超码。根据 BCNF 分解规则，得到：

$$(s\_ID, i\_ID)$$
$$(i\_ID, dept\_name)$$

上面的两个模式都属于 BCNF。（事实上，你可以验证，按照定义任何只包含两个属性的模式都属于 BCNF。）然而注意，在 BCNF 设计中，没有一个模式包含函数依赖 $s\_ID, dept\_name \rightarrow i\_ID$ 中出现的所有属性。

分解后的模式中没有包含原有的函数依赖，称为不是保持依赖的。

第三范式允许保持依赖

3NF

条件 对于F^+中所有形如$\alpha \rightarrow \beta$的函数依赖，以下至少一项成立：

- $\alpha \rightarrow \beta$是一个平凡的函数依赖
- α是模式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（区别于BCNF）（不直观，背吧）

注意第3个条件并没有说单个候选码必须包含$\beta - \alpha$中的所有属性，$\beta - \alpha$中的每个属性A可能包含于不同的候选码中。

BCNF比3NF更严格。

以上例子不属于BCNF但属于3NF。

4NF

设R(U)是属性集U上的一个关系模式。X，Y，Z是U的子集,并且Z=U-X-Y。关系模式R(U)中多值依赖X→→Y成立，当且仅当对R(U)的任一关系r,给定的一对(x，z)值有一组Y的值，这组值仅仅决定于x值而与z值无关。

条件 对D^+中所有形如$\alpha \rightarrow\rightarrow \beta$的多值依赖，至少有以下之一成立

- $\alpha \rightarrow\rightarrow \beta$是一个平凡的多值依赖
- α是R的一个超码

Computing the attribute closure, candidate key or primary key **计算属性闭包、候选码或主码**

计算闭包，直到闭包不变算法终止，可用于判断是否候选码或主码

Indexing **索引**

两种基本类型：

- 顺序索引 基于值的顺序排序
- 散列索引 基于将值平均分布到若干散列桶中，一个值所属的桶由散列函数决定

搜索码 用于在文件中查找记录的属性或属性集

Primary/secondary index concepts **主/辅助索引概念**

- 聚集索引（主索引）：包含记录的文件按照某个搜索码指定的顺序排序，**该搜索码对应的索引称为聚集索引**
- 非聚集索引（辅助索引）：搜索码指定的顺序与文件中记录的物理顺序不同的索引

都是相对**搜索码**而言的。

Dense/sparse index concepts **密集/稀疏索引概念**

- **稠密索引** 在稠密索引中，文件中的每个搜索码都有一个索引项。
  - 在稠密聚集索引中，索引项包括搜索码值以及指向具有该搜索码值的第一条数据记录的指针。
  - 在稠密非聚集索引中，索引必须存储指向所有具有相同搜索码值的记录的指针列表。
- **稀疏索引** 在稀疏索引中，只为搜索码的某些值建立索引项。
  - 只有当关系按搜索码排列顺序存储时才能使用稀疏索引，也就是聚集索引才能有稀疏索引。

B+ tree construction, query, insertion & deletion B+**树**

Query processing **查询处理**

概念 查询处理是从数据库中提取数据时涉及的一系列活动，包括：将用高层数据库语言表示的查询语句翻译为能在文件系统的物理层上使用的表达式，为优化查询而进行各种转换，以及查询的实际执行。

计算原语 加了"如何执行"注释的关系代数运算

查询执行计划 用于执行一个查询的原语操作序列

Concept of query optimization **查询优化的概念**

对于一个给定查询，尤其是复杂查询，通常会有许多种可能的策略，查询优化就是从这多种策略中找出最有效的查询执行计划的一种处理过程。不期望用户能够写出一个高效处理的查询，而是期望系统能够构造一个能让查询执行代价最小化的查询执行计划。

Query cost of selection operation **选择操作的查询成本**

假设磁盘子系统传输一个块的数据平均消耗t_T秒，磁盘块平均访问时间（磁盘搜索时间加上旋转延迟）为t_S秒，b_r表示在文件中的块数量，h_i表示B+树高度,，b是包含具有指定搜索码记录的块数。

A1（线性搜索）	$cost = t_1 + b_r \times t_T$	在线性搜索中，系统扫描每一个文件块，对所有记录都进行测试，看它们是否满足条件。
A1（线性搜索，码属性等值比较）	平均情形 $cost = t_1 + (b_r/2) * t_T$	
A2（主索引，码属性等值比较）	$(h_i + 1) * (t_T + t_1)$	对于具有主索引的码属性的等值比较，可以使用索引检索到满足相应等值条件的唯一一条记录。
A3（主索引，非码属性等值比较）	$h_i * (t_T + t_S) + b * t_T + t_S$	当选择条件是基于非码属性A的等值比较时，我们可以利用主索引检索到多条记录（必然是连续存储的）。
A4（辅助索引，等值比较）	$(h_i + 1) * (t_T + t_1)$	使用等值条件的选择可以使用辅助索引。若等值条件是码属性上的，则该策略可检索到唯一一条记录：若索引字段是非码属性，则可能检索到多条记录。
A4（辅助索引，A3（主索引，非码属性等值比较）	$(h_i + n) * (t_T + t_1)$	

Number of I/Os for nested-loop join, block nested-loop join, indexed nested-loop join 嵌套循环连接、块嵌套循环连接、索引嵌套循环连接的 I/O 数

$r \bowtie_\theta s$

嵌套循环连接

外层循坏为r，内层为s

所需考虑的元组对数目为$n_s * n_r$

最坏情况需要$n_r * b_s + b_r$次块传输（缓冲区只能容纳每个关系的一个数据块） $n_r + b_r$次磁盘搜索

最好情况下为$b_r + b_s$（内存有足够空间同时容纳两个关系） 2次磁盘搜索

块嵌套循环连接

内层关系的每一块与外层关系的每一块形成一对。

最坏情况需要$b_r * b_s + b_r$次块传输 $2b_r$次磁盘搜索

最好情况下为$b_r + b_s$ 2次磁盘搜索

索引嵌套循环连接

$b_r(t_T + t_S) + n_r * c$，c为使用连接条件对关系s进行单次选择操作的代价

Basic steps of External merge sort 外部归并排序的基本步骤

外排序 不能全部放在内存中的关系的排序称为外排序，最常用的外排序算法为外部排序归并算法。令M表示内存可容纳的磁盘块数。

1. 第一阶段，建立多个排好序的归并段。每个归并段都是排序过的，但仅包含关系中的部分记录。

```
i = 0;
repeat
    读入关系的M块数据或剩下的不足M块数据
    在内存中对关系的这一部分进行排序
    将排好序的数据写到归并段文件R1中
    i= i + 1;
until 到达关系末尾
```

2. 第二阶段，对归并段进行归并。暂时假定归并段的总数N小于M，这样我们可以为每个归并段文件分配一个块，此外剩下的空间还应能容纳存放结果的一个块。归并阶段的工作流程如下：

   为N个归并段文件R_i各分配一个内存缓冲块，并分别读入一个数据块

```
repeat
    在所有缓冲块中按序挑选第一个元组；
    把该元组作为输出写出，并将其从缓冲块中删除；
    if 任何一个归并段文件Ri的缓冲块为空并且没有到达Ri末尾
        then 读入Ri的下一块到相应缓冲块
until 所有的缓冲块均为空
```

该算法对N各归并段进行归并，因此它称为N路归并。

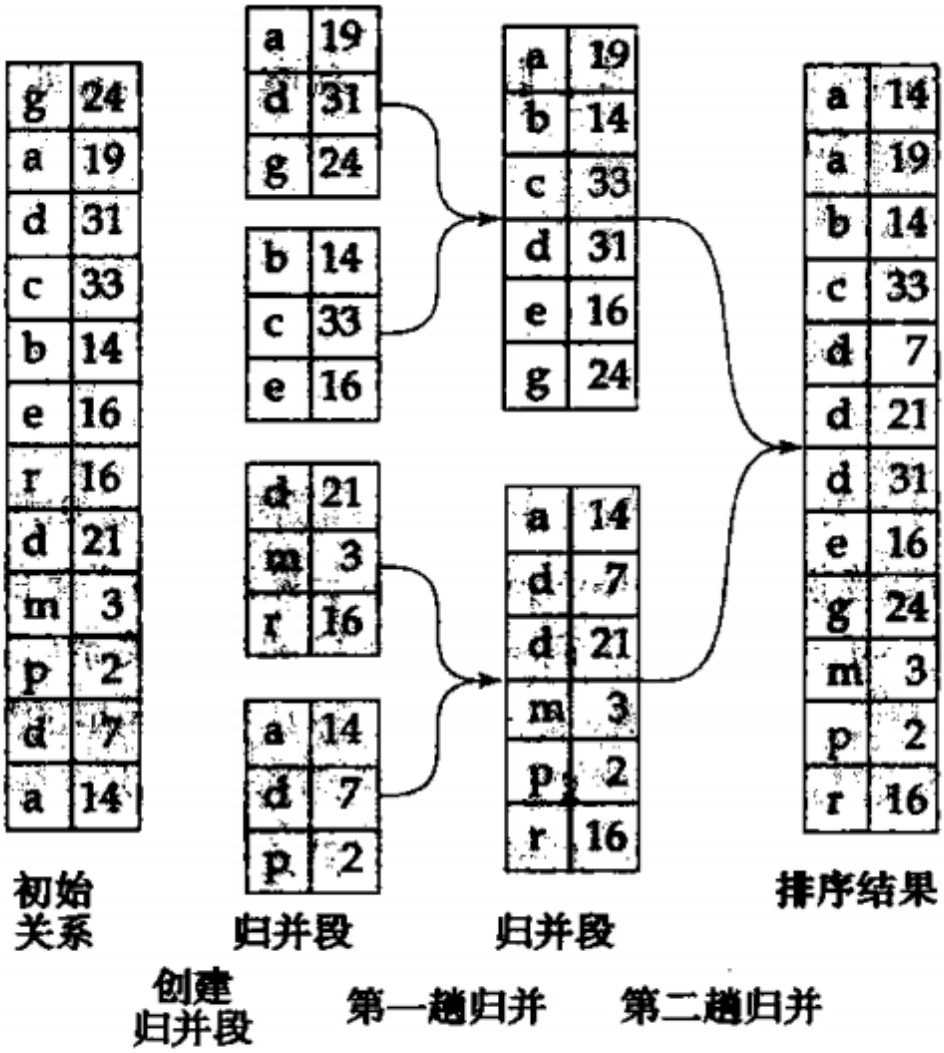


图 12-4　使用归并排序的外排序

等价规则

等价规则指出两种不同形式的表达式是等价的。

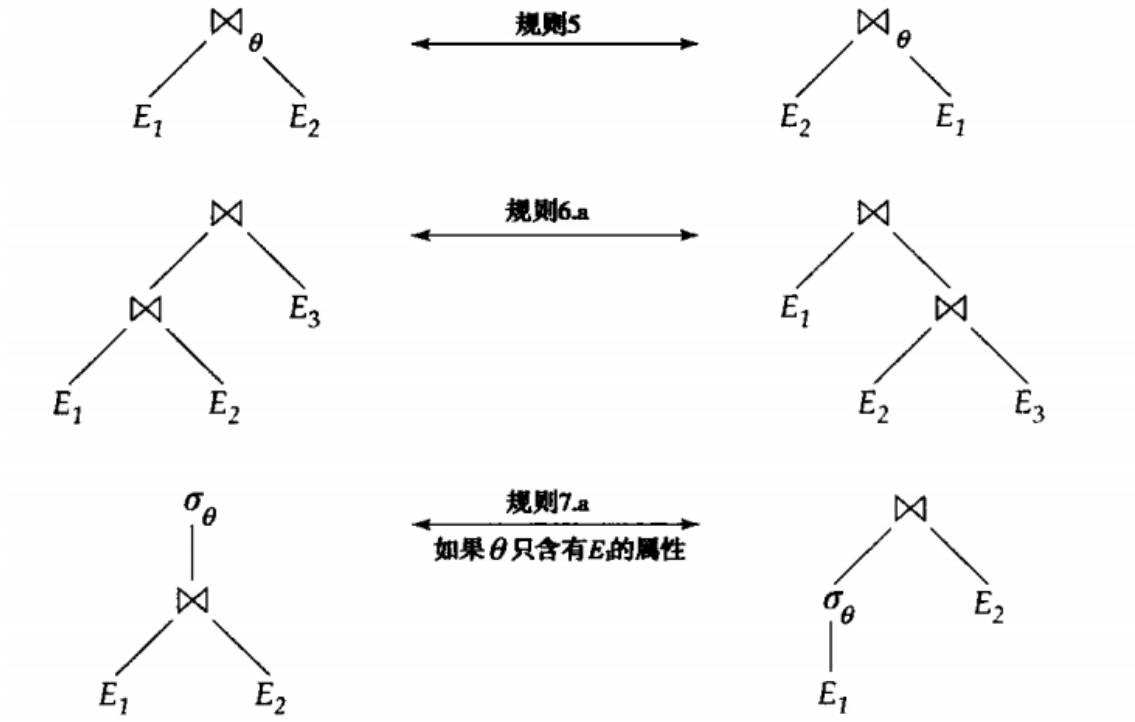


图 13-3　等价表达式的图形化表示

Transaction 事务

Concept of transaction and ACID 事务和ACID概念

事务 事务是访问并可能更新各种数据项的一个程序执行单元。

ACID 事务是一个单一的、不可分割的单元。若事务由于某些原因执行失败，则事务先前对数据库造成的影响都要撤销。称为**原子性**。

数据库系统必须采取特殊处理来确保事务正常执行而不被来自并发执行的数据库语句所干扰，这种特性称为**隔离性**。

数据库系统崩溃后，事务的操作也是持久的，称为**持久性**。

事务必须保持数据库的一致性——如果一个事务作为原子从一个一致的数据库状态开始独立的运行，则事务结束时数据库也必须再次是**一致的**。

- 原子性(Atomicity)： 事务的所有操作在数据库中要么正确反映出来， 要么完全不反应。其用途在于保证系统的一致性， 避免数据出错。 若部分操作反映而部分操作不反映则可能导致系统状态不再反映现实世界的真实状态， 即不一致状态。
- 一致性(Consistency)： 隔离执行事务时保持数据库系统的一致性。 用途在于保证系统数据正确， 数据库在事务执行前是一致的， 那么在事务执行后也仍然一致。 若没有一致性要求， 数据可能会被凭空创造或销毁， 如银行系统中的存款。
- 隔离性(Isolation)： 尽管多个事务可能并发执行， 但每个事务都感觉不到系统中有其他事务在并发执行。 即， 对任何一对事务 Ti,Tj,在 Ti 看来， Tj 在 Ti 开始前已经完成， 或者完成后才开始， 在 Tj 看来亦然。 该特性的用途是确保事务并发执行后的系统状态与这些事务以某种次序一个接一个地执行后的状态是等价的。
- 持久性(Durability)： 一个事务成功完成后， 即使出现系统故障， 它对数据库的影响必须是永久的。 该特性保证了系统可靠， 数据不会轻易丢失， 面对特殊复杂环境也能够保护数据 。

Relation between conflict serializability and view serializability **冲突可串行化与视图可串行化的关系**

> **冲突等价** 如果调度S可以经过一系列非冲突指令交换转换成S'，我们称S和S'是冲突等价的。
>
> **冲突可串行化** 若调度S与一个串行调度冲突等价，则称调度S是冲突可串行化的。（等价于两个事务先后执行）
>
> 确定一个调度是否冲突可串行化的方法。设S是一个调度，由S构造一个有向图，称为优先图，由两部分组成$G=(V,E)$，其中V是顶点集，E是边集，顶点集由所有参与调度的事务组成，边集由满足下列3个条件之一的边$T_i \rightarrow T_j$组成：
>
> - 在T_j执行read（Q）之前，T_i执行write（Q）
> - 在T_j执行write（Q）之前，T_i执行read（Q）
> - 在T_j执行write（Q）之前，T_i执行write（Q）
>
> 如果优先图中存在边$T_i \rightarrow T_j$，则在任何等价于S的串行调度S'中，T_i必出现在T_j之前。如果调度S的优先图有环，则调度S是非冲突可串行化的，如果优先图无环，则调度S是冲突可串行化的。
>
> 串行化顺序可通过拓扑排序得到。

deadlock **死锁**

> 如果需要“修改”一条数据，首先数据库管理系统会在上面加锁，以保证在同一时间只有一个事务能进行修改操作。锁定(Locking)发生在当一个事务获得对某一资源的“锁”时，这时，其他的事务就不能更改这个资源了，这种机制的存在是为了保证数据一致性。数据库死锁是指两个或多个事务互相等待彼此所持有的锁，而等待的过程中不会把自己持有的锁释放掉，进入了一种每个事务都不能正常执行的状态。

2PL(two-phase locking) design 2PL**（两相锁定）设计**

> 保证可串行性的一个协议是两阶段封锁协议。该协议要求每个事务分两个阶段提出加锁和解锁申请：
>
> 1. 增长阶段：事务可以获得锁，但不能释放锁
> 2. 缩减阶段：事务可以释放锁，但不能获得新锁
>
> 最初，事务处于增长状态，事务根据需要获得锁。一旦该事务释放了锁，它就进入了缩减阶段，并且不能再发出加锁请求。
>
> **严格两阶段封锁协议** 这个协议除了要求封锁是两阶段的之外，还要求事务持有的所有排他锁必须在事务提交后才可释放。这个要求保证未提交事务所写的任何数据在该事务提交之前均以排他方式加锁，防止其他事务读这些数据。
>
> **强两阶段封锁协议** 要求事务提交之前不得释放任何锁。
>
> **严格两阶段封锁协议的好处与弊端** 严格两阶段封锁协议要求事务在提交之前不得释放任何排他锁， 好处在于可以避免级联回滚。 弊端在于相比一般的两阶段封锁协议， 为了满足上述约束， 可以进行并行化的调度方式更少， 事务之间的并行程度更低。 所有的两阶段封锁协议都可以根据封锁点的顺序构造事务的可串行化顺序， 但都不能避免死锁

Logging functionality **日志功能**

使用最为广泛的记录数据库修改的结构就是日志，日志是日志记录的序列，它记录数据库中的所有更新活动。

更新日志记录描述一次数据库写操作，具有如下几个字段：

- **事务标识** 是执行write操作的事务的唯一标识
- **数据项标识** 是所写数据项的唯一标识。通常是数据项在磁盘上的位置，包括数据项所驻留的块的块标识和块内偏移量。
- **旧值** 数据项的写前值
- **新值** 数据项的写后值

Basic recovery algorithm 基本恢复算法

1. 事务回滚（非系统崩溃）
    - 从后往前扫描日志，补偿日志记录
    - 一旦发现了$< T_i, start >$日志记录，就停止从后往前的扫描，并往日志中写一个$< T_i, abort >$日志记录
2. 系统崩溃后的恢复
    - 在重做阶段，系统通过从最后一个检查点开始正向地扫描来重做所有事物的更新。
    - 在撤销阶段，系统回滚undo-list中的所有事务。通过从尾端开始反向扫描日志来执行回滚。