数据库期末复习

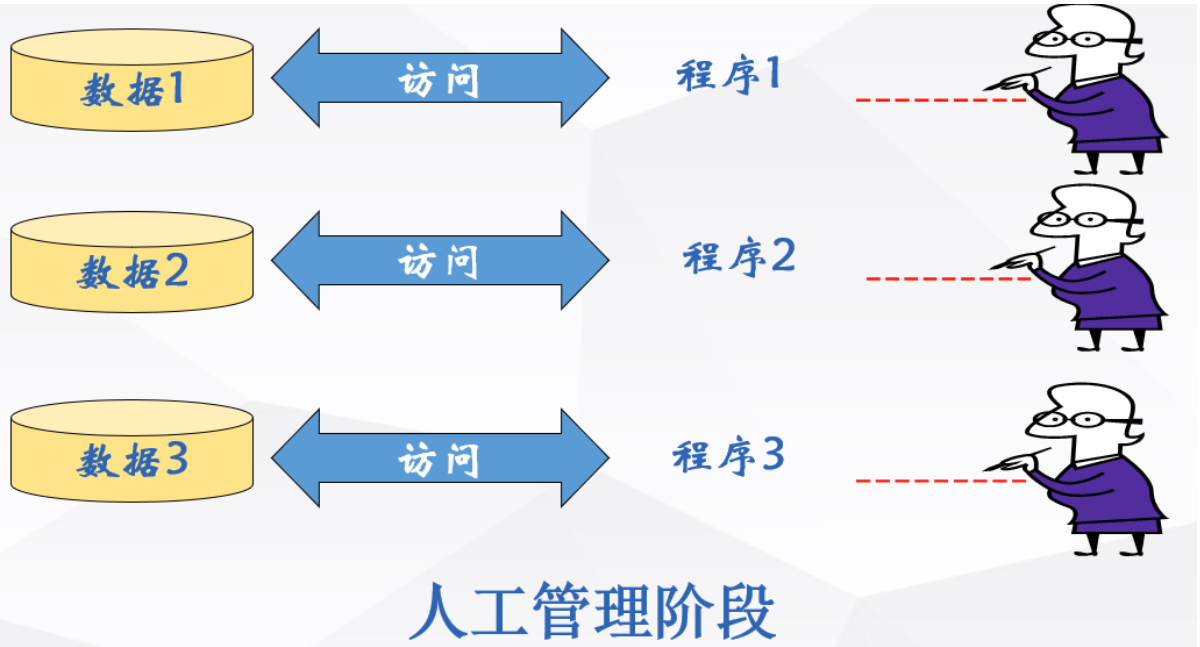
1 数据库概论

数据库的由来和发展

四个阶段：人工管理、文件系统、数据库阶段、高级数据库阶段。

人工管理

- 数据不保存在计算机内
- 没有专用的软件对数据进行管理
- 只有程序的概念，没有文件的概念
- 一组数据对应一个程序**（数据面向程序）**
- 数据处理只有批处理方式



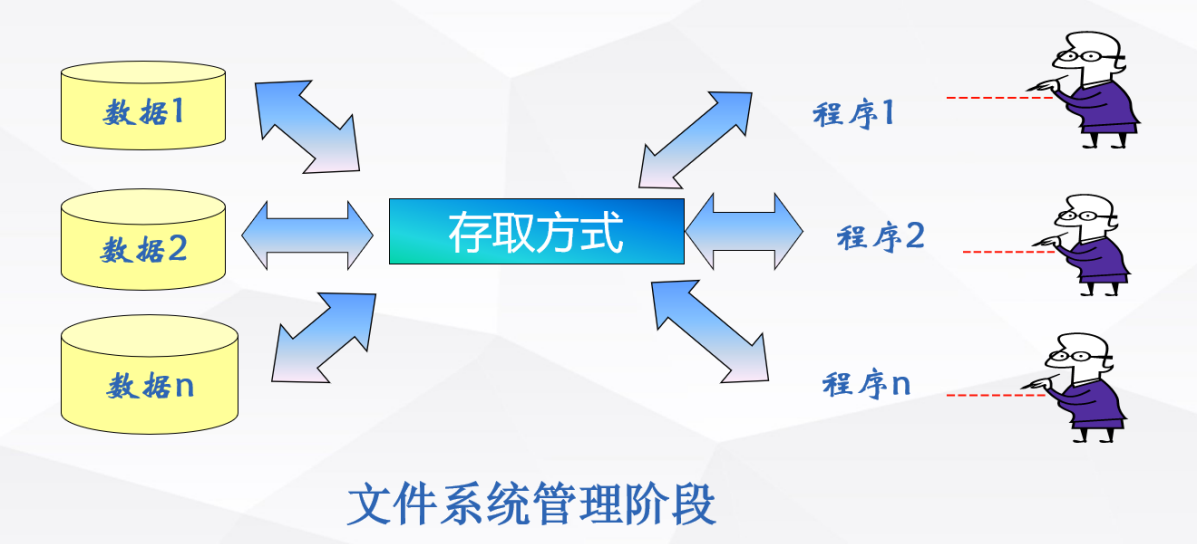
文件系统阶段

特点：

- 数据**以文件的形式**保存在外部存储器的磁盘上
- 数据的逻辑结构和物理结构有了简单区别。具有**设备独立性**，程序只需要用文件名，不必关心数据的物理位置。
- 文件组织多样化
- 数据不再属于某个特定的程序（**数据面向应用**）
- 对数据的操作以记录为单位。

缺陷：

- 数据冗余
- 数据不一致
- 数据联系弱



数据库阶段

- 采用**数据模型**表示复杂的数据结构
- 有较高的**数据独立性**
- 数据库系统为用户提供了方便的用户接口
- 通过DBMS，进行四方面的控制功能
    - 数据库的恢复
    - 数据库的并发控制
    - 数据的完整性
    - 数据的安全性
- 以**数据项**作为最小的数据存取单位，增加了系统的灵活性

高级数据库阶段

- 对象数据库技术
- 分布式数据库系统
- 开放数据库互联技术
- XML数据库技术
- 现代信息集成技术

数据描述

概念设计与逻辑设计的对应关系

概念设计	逻辑设计
实体	记录
属性	字段（数据项）
实体集	文件
实体标识符	关键码

物理数据描述：数据在存储设备上的存储方式的描述，物理数据是实际存放在存储设备上的数据。

逻辑数据描述：程序员或用户以操作的数据形式的描述，是抽象的概念化数据。

数据抽象的级别

- 概念模型：表达**用户**需求观点的数据全局逻辑结构的模型。**实体联系模型**属于概念模型。
- 逻辑模型：表达计算机实现观点的**DB全局逻辑结构**的模型。层次模型、网状模型、关系模型、对象模型
    - 表达了DB的整体逻辑结构
    - 从数据库实现的观点出发对数据建模
    - 独立于硬件，**依赖于软件**：选定DBMS后，将概念模型按照选定的DBMS的特点转换而来。
    - 数据库设计人员与应用程序员之间进行交流的工具
- 外部模型：表达用户使用观点的DB局部逻辑结构的模型
    - 是逻辑模型的一个逻辑子集
    - 独立于硬件、依赖于软件
    - 反映了用户使用数据库的观点
    - 简化了用户的观点
    - 有助于数据库的安全性保护
    - 对概念模型的支持
- 内部模型：表达DB物理结构的模型
    - 又称物理模型
    - 数据库最底层的抽象

  四种模型的关系：

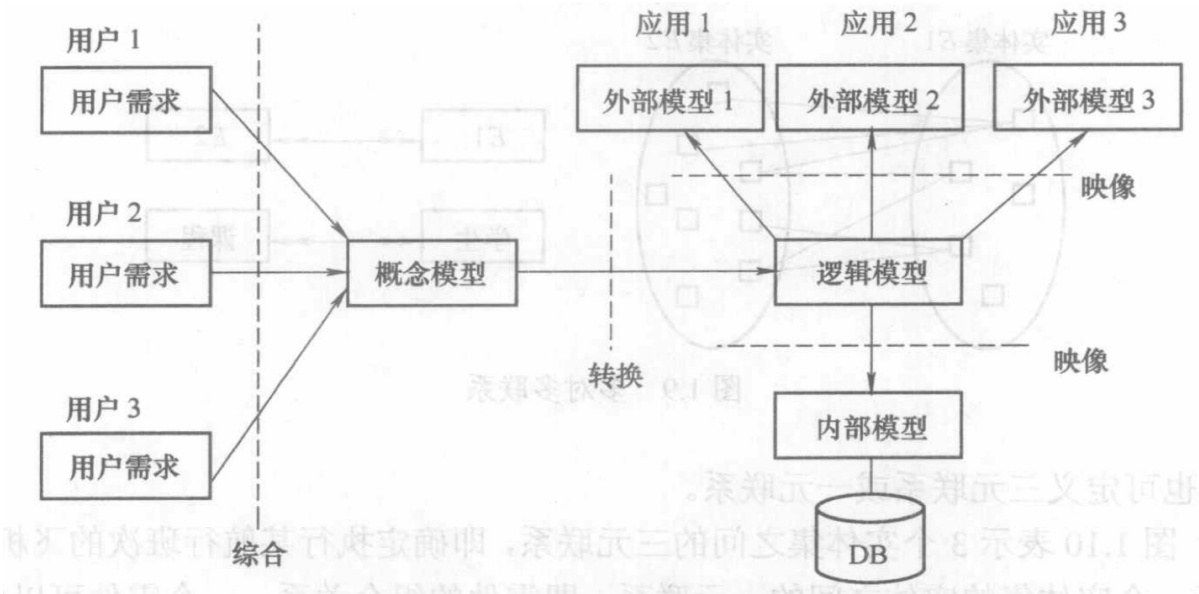


图 1.12　4 种模型之间的相互关系

三层模式，两级映像

三层模式：外模式、（逻辑）模式、内模式

- 外模式：用户与数据库系统的接口，用户用到的那部分数据的描述。由若干个外部记录类型组成。

- 逻辑模式：数据库中全部数据的整体逻辑结构的描述。由若干个逻辑记录类型组成，还包括记录间联系、数据的完整性和安全性等要求。
- 内模式：数据库在物理存储方面的描述。定义所有内部记录类型、索引和文件的组织方式以及数据控制方面的细节。

两级映像：三层模式之间的影响，来说明外部记录、逻辑记录和内部记录之间的对应性。

- 两个映像分别在外模式、内模式中描述（而不在逻辑模式中）。

高度的数据独立性

数据独立性：应用程序和数据库的数据结构之间相互独立，不受影响。在修改数据结构时，尽可能不修改应用程序。分为物理数据独立性和逻辑数据独立性。

- 物理独立性：当物理结构（内模式）发生变化时，通过修改模式/内模式间的映像（对应关系），可使得应用程序不变。
- 逻辑独立性：当全局逻辑结构（概念模式）发生变化，通过修改外模式/模式间的映射，使得应用程序不变。

一般说来，关系数据库系统在支持数据独立性方面优于层次、网状系统。

数据库管理系统

DBMS的工作模式

根据**数据模型**的不同，DBMS可以分成：层次型、网状型、关系型、对象型等。

DBMS的主要功能

- 数据库定义：提供三层结构、两级映像，定义数据的完整性约束、保密限制等约束。（DDL）
- 数据库的操纵：查询和更新（插入、删除、更新）（DML）
- 数据库的保护：数据库的恢复、并发控制、数据完整性控制、数据安全性控制。
- 数据库的维护：数据载入、转换、转储，数据库的改组及性能监控。
- 数据字典。

DBA

数据库管理员（DataBase Administrator）负责设计、建立、管理和维护数据库并协调用户对数据库要求的个人或工作团队。

DBA应熟悉计算机软硬件系统、具备较全面的数据处理知识，熟悉应用的业务种类、数据情况及业务流程。

（1）与用户联络，定义数据库的模式；

（2）定义内模式；

（3）与用户联络：定义外模式、设计应用程序；

（4）定义安全性规则，授予用户访问数据库的权限；

（5）定义完整性规则，监督数据库的运行；

（6）数据库的转储与恢复工作。

2 关系模型和关系运算理论

基本概念

关系模型：用二维表格表示实体集，用关键码表示实体之间联系的数据模型。

元数：关系中属性的个数。

元组：每一行。

基数：元组个数。

各种键：

- 超键：关系中能唯一表示元组的属性或数据集。（超键= 候选键+ 任意其他属性）
- 候选键：不含有多余属性的超键。
- 主键：用户选作元组标识的候选键。
- 外键：模式R中属性K是其他模式的主键，则K在R中是外键。
    - 外键一定是另一个表的**候选键**（注意不是主键），外键可以为空。

关系：每一个属性值不可分解，不允许出现重复元组，不考虑元组间的顺序，属性无序。

关系模型的3类完整性规则

- 实体完整性规则：主键的属性上不能有空值。
- 参照完整性规则：不允许引用不存在的实体。（外键）
- 用户定义的完整性规则：例如，用户定义年龄在15~30岁之间。

关系模型的3层体系结构

- 关系模式：由DDL定义，不涉及物理存储方面的描述，仅仅描述数据本身的一些特性。
- 子模式：用户所用到的那部分数据的描述，需定义用户对数据进行操作的权限。
- 存储模式：关系存储时作为文件看待。

关系模型的3个组成部分

- 数据结构：数据库中全部数据及其相互联系被组织成“关系”（二维表格）的形式。关系模型的基本数据结构是关系。
- 数据操纵：关系模型提供一组完备的高级关系运算。关系运算分为关系代数、关系演算和关系逻辑。
- 数据完整性规则：必须满足实体完整性、参照完整性和用户定义的完整性3类完整性规则。

关系查询语言和关系运算

DML分为查询语句和更新语句。关于查询的理论称为关系运算理论。

- 关系代数语言：查询操作是以**集合操作**为基础的运算
- 关系演算语言：查询操作时以**谓词演算**为基础的运算
- 关系逻辑语言：查询操作是以**if-then逻辑**操作为基础的运算

关系代数

5个基本操作

- 并 $\cup$
- 差 $-$
- 笛卡尔积（交叉连接） $\times$

> AxB={<x,y>|x∈A^y∈B}
> 例如，A={a,b},B={0,1,2},则
> AxB={<a,o>,<a,1>,<a,2>,<b,0>,<b,1>,<b,2>,}
> BxA={<0,a>,<0,b>,<1,a>,<1,b>,<2,a>,<2,b>}

A	B	C
1	2	3
4	5	6
7	8	9

(a) 关系R

A	B	C
2	4	6
4	5	6

(b) 关系S

图 2.8 例 2.2 中的两个关系

A	B	C
1	2	3
7	8	9

(b) $R-S$

R.A	R.B	R.C	S.A	S.B	S.C
1	2	3	2	4	6
1	2	3	4	5	6
4	5	6	2	4	6
4	5	6	4	5	6
7	8	9	2	4	6
7	8	9	4	5	6

(c) $R\times S$

C	A
3	1
6	4
9	7

(d) $\pi_{C,A}(R)$

- 投影 π

> $\pi_i(R)$ 在R表中取 i列

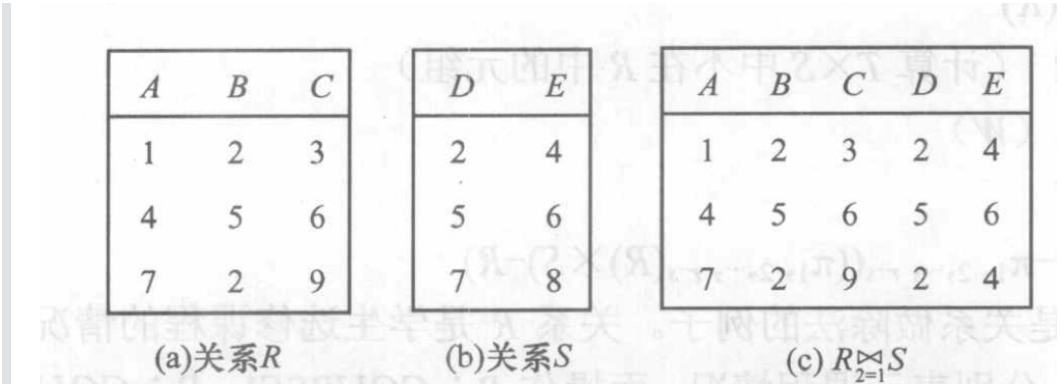
- 选择σ

  根据条件对关系做水平分割，选取符合条件的元组。

4个组合操作

- 交 $\cap$
- 连接 $\underset{i\theta j}{\bowtie}$

> 从笛卡尔积中选取属性值满足某一 θ操作的元组
>
> θ为 $=$时称为等值连接。
>
> θ下方的 i,j可以为列的序号，也可以是列名



A	B	C
1	2	3
4	5	6
7	2	9

(a)关系R

D	E
2	4
5	6
7	8

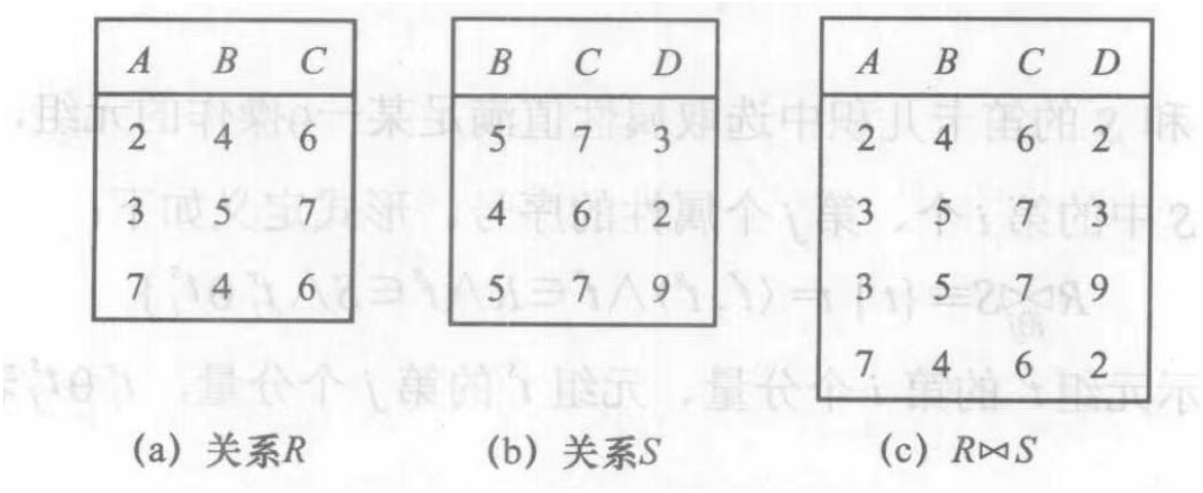
(b)关系S

A	B	C	D	E
1	2	3	2	4
4	5	6	5	6
7	2	9	2	4

(c) $R\underset{2=1}{\bowtie}S$

- 自然连接 ⋈

  消除了公共属性的笛卡尔积。（如下图，不再是R.B,S.B,而是B）



A	B	C
2	4	6
3	5	7
7	4	6

(a) 关系R

B	C	D
5	7	3
4	6	2
5	7	9

(b) 关系S

A	B	C	D
2	4	6	2
3	5	7	3
3	5	7	9
7	4	6	2

(c) $R\bowtie S$

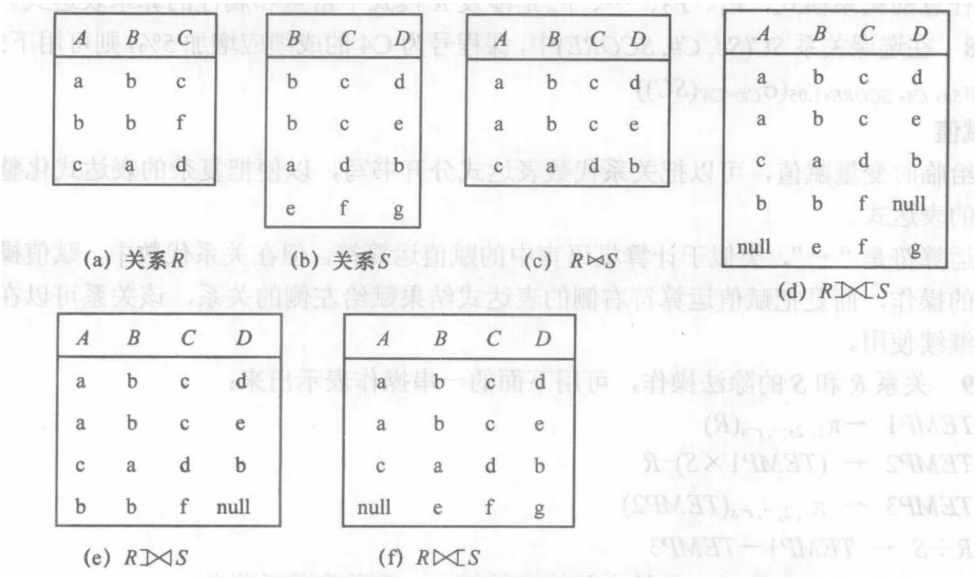
- 除法（不考）

实例：

$$\pi_{SID,SNAME}(\sigma_{TNAME='LIU'}(S \bowtie SC \bowtie C \bowtie T))$$

7个扩充操作

- 改名（不考）
- 广义投影
- 赋值
- 外连接 R⟕S
  - 外连接，将自然连接中原本舍弃的元组保留⟗
  - 左外连接，只将自然连接中R中原该舍弃的保留⟕
  - 右外连接，只将自然连接中S中原该舍弃的保留⟖

- 

A	B	C
a	b	c
b	b	f
c	a	d

(a) 关系R

B	C	D
b	c	d
b	c	e
a	d	b
e	f	g

(b) 关系S

A	B	C	D
a	b	c	d
a	b	c	e
c	a	d	b

(c) $R \bowtie S$

A	B	C	D
a	b	c	d
a	b	c	e
c	a	d	b
b	b	f	null
null	e	f	g

(d) $R \overset{\text{left}}{\bowtie} S$

A	B	C	D
a	b	c	d
a	b	c	e
c	a	d	b
b	b	f	null

(e) $R \overset{\text{right}}{\bowtie} S$

A	B	C	D
a	b	c	d
a	b	c	e
c	a	d	b
null	e	f	g

(f) $R \overset{\text{full}}{\bowtie} S$

- 外部并
  - 不要求关系模式相同
  - 构建出的新关系由R,S的所有属性组成，公共属性只取一次，新的元组由属于R或属于S的元组构成，新的属性上填上空值。

A	B	C	D
a	b	c	null
b	b	f	null
c	a	d	Null
null	b	c	d
null	b	c	e
null	a	d	b
null	e	f	g

图 2.14　外部并的例子

- 聚集操作
  - 输入一个值的集合，得到一个单一的值作为结果。

> $count_{SID}(\sigma_{age='18'}(S))$

关系代数的启发式优化算法

- 早选择
- 早投影
- 避免直接做笛卡尔积

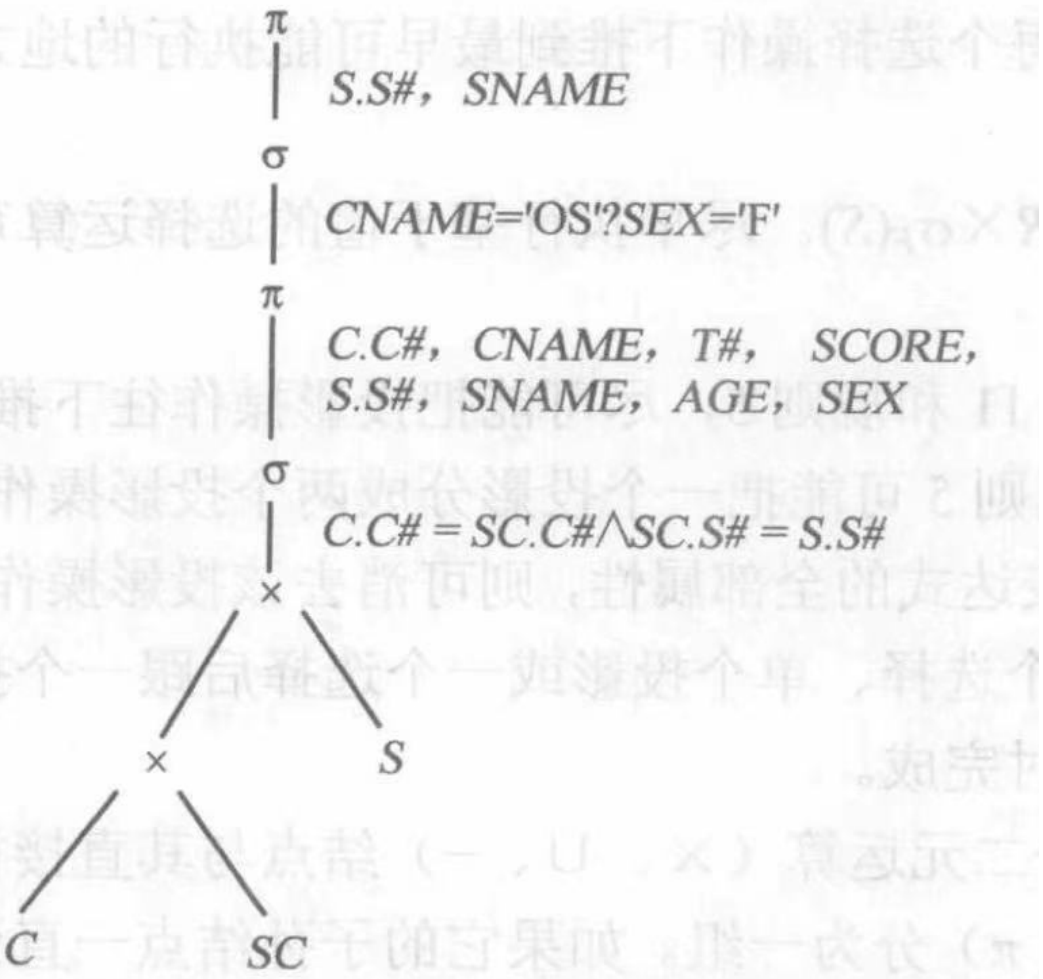
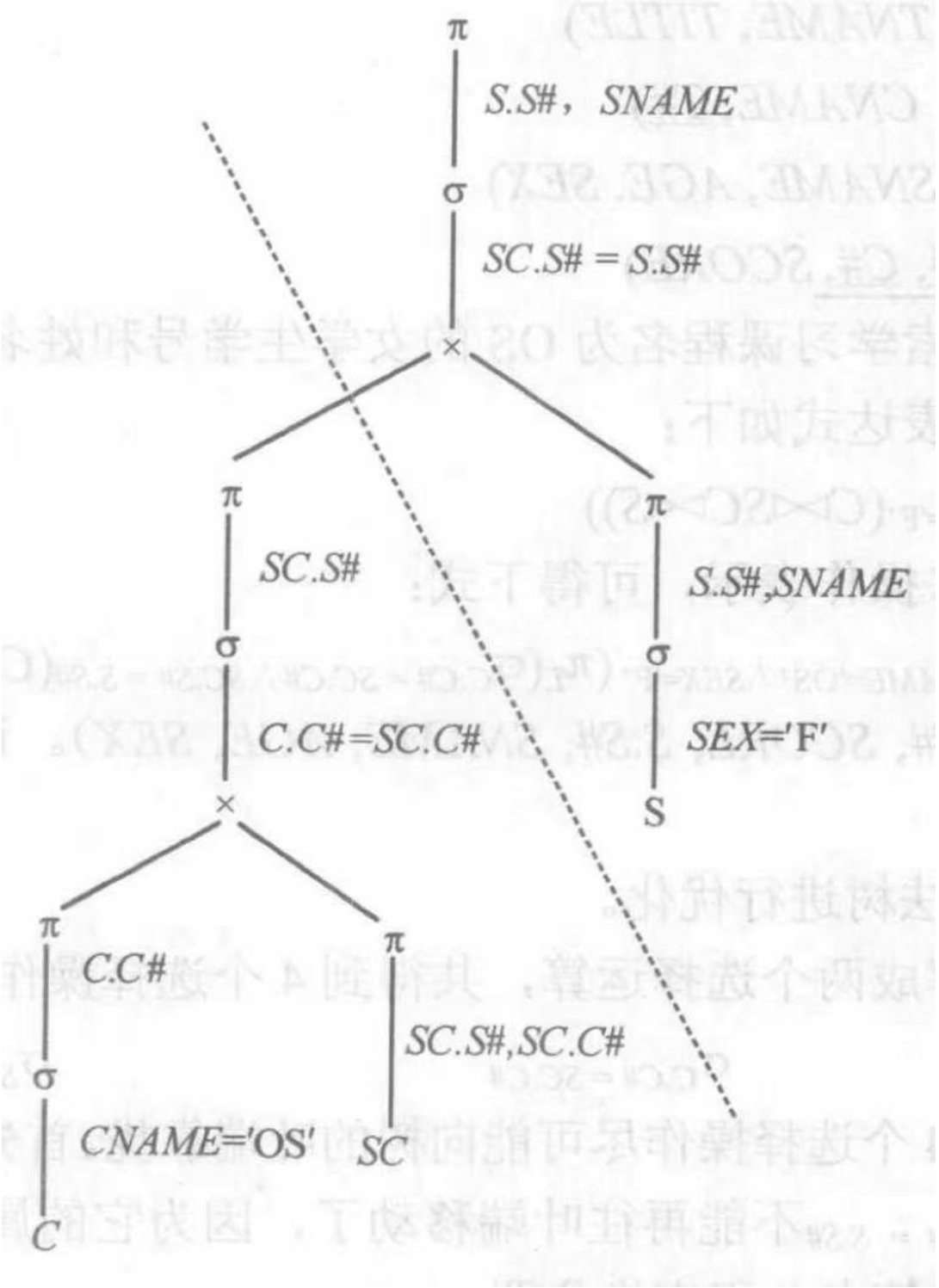


图 2.18　关系代数表达式的初始语法树



经验之谈：一般 σ后可能都有 π

一定要记得分组！

关于这里不细写了，看一看第一次作业的作业题推一遍应该就能会了。

关系演算

好像没讲，应该不考。

关系逻辑

好像没讲，应该不考

3 SQL语言

SQL的组成

- 数据定义语言DDL：定义SQL模式、基本表、视图、索引。
- 数据操纵语言DML：数据查询和更新，更新包括插入、删除、修改。
- 嵌入式SQL语言的使用规定
- 数据控制语言DCL：对基本表和视图的授权，完整性规则的描述，事务控制

DDL

- 模式创建：CREATE SCHEMA <模式名> AUTHORIZATION 用户名
- 模式撤销：DROP SCHEMA <模式名> [CASCADE | RESTRICT]
    - CASCADE：级联，SQL模式及其下属的基本表、视图、索引等所有元素全部撤销。
    - RESTRICT：约束，只有当SQL模式中没有任何下属元素时，才能撤销SQL模式，否则不执行SQL语句。
- 创建表：CREATE TABLE <基本表名>(<列名类型>,...,<完整性约束>,...)

    ```
    CREATE TABLE C
    (
    CID CHAR(4),
    CNAME CHAR(10) NOT NULL,
    PRIMARY KEY(CID)
    FOREIGN KEY(TID) REFERENCES T(TID) -- 注：外键和对应的引用的键可以不同名，指出对应性即可
    )
    ```

- 添加列：ALTER TABLE <基本表名> ADD <列名><类型> --*注：不能定义为NOT NULL*
- 删除列：ALTER TABLE <基本表名> DROP <列名>[CASCADE | RESTRICT]
- 修改列的类型、宽度：ALTER TABLE <基本表名> MODIFY <列名><类型>
- 撤销表：DROP TABLE <基本表名> [CASCADE | RESTRICT]
- 创建索引：CREATE [UNIQUE | CLUSTER] INDEX <索引名> ON <基本表名>(<列名序列>)
    - UNIQUE：每个索引值对应唯一的数据记录。
    - ASC升序，DESC降序，默认升序。

        ```
        CREATE UNIQUE INDEX SC_INDEX ON SC(S# ASC, C# DESC)
        ```

- 撤销索引：DROP INDEX <索引名>

DML

仅列一下关键字和基本用法供回忆，具体掌握还得实际去练。

```
SELECT <目标表的列名或列表达式序列>
FROM <基本表名或视图序列>
```

> [WHERE <行条件表达式>]
>
> [GROUP BY <列名序列>
>
> [HAVING <组条件表达式>]]
>
> [ORDER BY <列名[ASC|DESC]>,...]

- SELECT

  查询结果是一个表，可嵌套SELECT。可做并、交、差。(UNION,INTERSECT,EXCEPT)
- WHERE 字句的条件表达式F中可使用下列运算符：
  - <,<=,>,>=,=,<>或!=
  - AND,OR,NOT
  - IN,NOT IN
  - EXISTS,ALL,SOME,UNIQUE
    - 注意ALL和SOME是同义词，均表示左边的元组与右边的集合中至少一个元组满足运算。早期用ANY，后来改为SOME。
  - AVG,MIN,MAX,SUM,COUNT
  - SELECT语句嵌套
- EXISTS：其后的式子的返回值为空，返回false，否则返回true。也就是说，在子查询中，只要有一个为true，就不为空，就返回true；而not exists就会变成，在子查询中，只要为空，就返回true
- DISTINCT
- <表达式1> [NOT] BETWEEN <表达式2> AND <表达式3>
- <字符串> [NOT] LIKE <匹配模式>
  - % 与0个或多个字符组成的字符串匹配。

    > where 姓名 LIKE '王%' 姓王的姓名
  - _ 与单个字符匹配

    > where 姓名 LIKE '王_' 姓王且全名为2个汉字的同学
    >
    > '_丽%'第二个字为丽
- <表达式> IS [NOT] NULL
- <元组> [NOT] IN (<集合>)
- [NOT] UNIQUE (<集合>) 判断集合是否有重复元组
  - 不存在重复元组，返回TRUE
- 外连接 [LEFT | RIGHT] JOIN <表名> ON (属性名1=属性名2)
- INSERT
  - INSERT INTO <基本表名> [(列名序列)] VALUES (<元组值>),(<元组值>)...(<元组值>)
  - INSERT INTO <基本表名> <SELECT 查询语句>
  - INSERT INTO <基本表名1> TABLE <基本表名2>

  只有当属性个数、顺序和基本表的结构上完全一致，列名序列才可省略
- DELETE FROM <基本表名> WHERE <条件表达式>

- UPDATE <基本表名> SET <列名>=<值表达式>[,<列名>=<值表达式>...] | ROW=(<元组>) [WHERE <条件表达式>]
- CREATE VIEW <视图名>（<列名序列>） AS <SELECT 查询语句>
    - 允许用户更新的视图，在定义时必须加上“WITH CHECK OPTION”
    - 定义在多个基本表上的视图，或者用聚合操作的视图，或不包含基本表主键的视图，不允许更新。
- DROP VIEW <视图名>

DCL

- CREATE USER XXX IDENTIFIED BY XXX [WITH GRANT OPTION]
- GRANT CONNECT TO XXX
- GRANT CREATE TABLE/INDEX/VIEW TO XXX
- GRANT DELETE/UPDATE/INSERT ON X(表名) TO XXX
- REVOKE DELETE/UPDATE/INSERT ON X(表名) FROM XXX
- CHECK（〈条件表达式〉）

  Check (score between 0 and 100);

  CHECK（S# IN（SELECT S# FROM S））
- On update/delete Cascade/Restrict 添加外键约束

```
CREATE TABLE SC
        ( S#    CHAR(4),
          C#    CHAR(8),
          GRADE SMALLINT,
          PRIMARY KEY (S#, C#),
          FOREIGN KEY (S#) REFERENCES S (S#)
             ON DELETE CASCADE,
          FOREIGN KEY (C#) REFERENCES C (C#)
             ON UPDATE RESTRICT,
          CHECK (GRADE BETWEEN 0 AND 100)
        );
```

4 关系数据库的规范化设计

一个关系模式包括**外延**和**内涵**两方面的内容。

- 外延：关系、表或当前值。外延与时间有关。
- 内涵（**关系模式**）：与时间独立，是对数据的定义以及数据完整性约束的定义。
    - 对数据的定义：对关系、属性、域的定义和说明
    - 对数据完整性约束的定义：
        - 静态约束：涉及数据之间的联系（**函数依赖**），主键和值域的设计
        - 动态约束：定义各种操作（插入、删除、修改）对关系值的影响。

关系模式的冗余和异常问题

模式可能出现的问题：

数据冗余：在关系中要出现多个元组，某些属性要重复多次

操作异常：由于数组冗余，在对数据进行操作时会引起各种异常。

- 修改异常：一个改了，而其他相同的没改
- 插入异常：主键出现空值
- 删除异常：如果要删除选课，则教师也要删除，这不合适。

关系模式的非形式化设计准则

- 尽可能只包含有直接联系的属性，不要包含有间接联系的属性。
- 尽可能使得相应关系中不出现插入、删除和修改等操作异常现象。
- 尽可能使得相应关系中避免放置经常为空值的属性。
- 尽可能使得关系的等值连接在主键和外键的属性上进行，并且保证连接以后不会生成额外的元组。（**无损分解**）

函数依赖

函数依赖（FD）是最基本、最重要的一种依赖。是关键码概念的推广。

> 对于关系r的任意两个元组，如果X值相同，则要求Y值也相同。即，有一个X值就有一个Y值与之对应，或者说Y值由X值决定。这种依赖称为函数依赖。

$X \to Y$：Y函数依赖于X，X函数决定Y

逻辑蕴含

若从函数依赖的集合F能推出一个函数依赖$X \to Y$，则称F逻辑蕴含$X \to Y$：

$F \models X \to Y$

FD的推理规则

Armstrong公理：

- 自反性：若$Y \subseteq X \subseteq U$，则$X \to Y$在$R$上成立
- 增广性：若$X \to Y$在R上成立，且$Z \subseteq U$，则$XZ \to YZ$在R上成立
- 传递性：若$X \to Y$和$Y \to Z$在R上成立，则$X \to Z$在R上成立
- 合并性：$\{X \to Y, X \to Z\} \models X \to YZ$
- 分解性：$\{X \to Y, Z \subseteq Y\} \models X \to Z$
- 伪传递性：$\{X \to Y, WY \to Z\} \models WX \to Z$
- 复合性：$\{X \to Y, W \to Z\} \models XW \to YZ$
- 通用一致性定理：$\{X \to Y, W \to Z\} \models X \cup (W - Y) \to YZ$

闭包

通过推理规则，求出所有可能的函数依赖。求闭包是一个NP完全问题，指数级时间问题。

$F^+ = \{X \to Y | F \models X \to Y\}$

关键码

如果$X \to U$ 在 R 上成立，则X是R的一个超键。如果X是能找到的最小的超键，则X是R上的一个候选键。

候选键可能有多个

- 找到只在左边、右边和左右都出现的属性。只左必定是候选键的一部分，只右一定不是候选键的一部分。
- 只左与左右的元素组合，推函数依赖，看其闭包是否为属性全集

推理规则的完备性

从FD集F使用推理规则集推出的FD必定在F^+中。

求最小函数依赖集F_{min}

最小函数依赖集G：

- G的每个FD的右边都是单属性。
- G中没有冗余的FD。

  即G中不存在这样的函数依赖$X \to Y$，使得$G - \{X \to Y\}$与G等价
- G中每个FD的左边没有冗余的属性。

  即G中不存在这样的函数依赖$X \to Y$，对于X的真子集W，使得
  $G - \{X \to Y\} \cup \{W \to Y\}$ 与G等价。
- 不唯一

步骤：

- 做右侧拆分，$X \to AB$拆成$X \to A, X \to B$
- 做左侧拆分，判断$AB \to X$是否可以用$A \to X$或者$B \to X$来取代。
- 逐一检查各剩下的函数依赖是否可以从其他函数依赖推出：逐个删除函数依赖，求剩下的函数依赖集从被删除函数依赖左侧开始的闭包。若闭包中包含了被删除的函数依赖的右侧属性，则可以删除该冗余函数依赖。

例：设$F = \{A \to BC, AD \to E, B \to E, E \to G, B \to G\}$

1. 右侧拆分

   $F = \{A \to B, A \to C, AD \to E, B \to E, E \to G, B \to G\}$
2. 左侧拆分

   $F = \{A \to B, A \to C, A \to E, D \to E, B \to E, E \to G, B \to G\}$
3. 逐一检查各剩下的函数依赖是否可以从其他函数依赖推出

3.1删除$A \to B$，$B \notin A^+ = \{ACEG\}$，所以$A \to B$不可删除

3.2删除$A \to C$，$C \notin A^+ = \{ABEG\}$，所以$A \to C$不可删除

3.3删除$A \to E$，$E \in A^+ = \{ABCEG\}$，所以$A \to E$可以删除

3.4删除$B \to E$，$E \notin B^+ = \{BG\}$，所以$B \to E$不可删除

3.5删除$E \to G$，$G \notin E^+ = \{E\}$，所以$E \to G$不可删除

3.6删除$B \to G$，$G \in B^+ = \{EG\}$，所以$B \to G$可以删除

模式分解

设有关系模式$R(U)$，属性集为$U, R_1, R_2, \ldots, R_k$都是U的子集。且$R_1 \cup R_2 \cup \ldots \cup R_k = U$。关系模式$R_1, \ldots, R_k$的集合用$\rho$表示，$\rho = \{R_1, \ldots, R_k\}$用$\rho$代替$R$的过程，称为关系模式的分解。$\rho$称为$R$的一个分解。

无损分解

分解后的关系做自然连接，连接的结果不变

优点：

- 消除数据冗余的操作异常现象
- 可以存储悬挂元组，存储泛关系中无法存储的信息

缺点：

- 分解以后，检索操作需要做笛卡尔积或连接操作，付出时间代价
- 可能产生寄生元组，损失了信息。

是否为无损分解

- chase方法（不考）
- $U_1 \cap U_2 \to (U_1 - U_2) | (U_2 - U_1)$ 若成立，则为无损分解。
- 例：

已知：$R < U, F >, U = \{A, B, C\}$

$F = \{A \to B\}, \rho_1 = \{R_1(AB), R_2(AC)\}$

解：$U_1 \cap U_2 = A, U_1 - U_2 = B$

$\because A \to B$

$\therefore U_1 \cap U_2 \to U_1 - U_2$

$\therefore \rho_1$是无损连接分解

保持函数依赖

- 如果不能保持FD，那么**数据的语义就会出现混乱**
- **保持函数依赖和无损分解之间没有任何必然联系**

是否保持函数依赖

分解的各个函数依赖集的并等价于原F

$$设R(A,B,C,D),F=\{A\to B,C\to D\}$$
$$\rho=\{R_1(A,B),R_2(C,D)\}$$
$$\Pi_{R_1}(F)=\{A\to B\}$$
$$\Pi_{R_2}(F)=\{C\to D\}$$
$$\Pi_{R_1}(F)\cup\Pi_{R_2}(F)等价于F，因此分解\rho保持函数依赖$$

关系模式的范式

1NF

每个关系r的属性值都是不可分的原子值

2NF

在1NF的基础上，每个非主属性完全函数依赖于候选键。

- 主属性：包含在任何一个候选码中的任何一个属性。
- 完全函数依赖：左部不可约依赖。

> 对于FD $W\to A$，如果存在$X\subset W$ 有 $X\to A$成立，那么称$W\to A$是局部函数依赖。否则，称为完全函数依赖。

3NF

在2NF的基础上，每个非主属性都不传递依赖于R的候选键。

> 如果$X\to Y,Y\to A,$且$Y\nrightarrow X$和$A\notin Y$，那么称$X\to A$是传递依赖。

BCNF

在3NF的基础上，每个属性都不传递依赖于R的候选键。

任何一个关系模式都能**无损连接地**分解成BCNF的集合。注意：不一定保持函数依赖

范式	说明
1	每个属性不可再分
2	消除非主属性对键的部分依赖
3	消除非主属性对键的传递依赖
BC	如果在关系R中，U为主键，A属性是主键的一个属性， 若存在A->Y，Y为主属性，则该关系不属于BCNF

分解成3NF的算法

任何一个关系模式都可无损连接且保持函数依赖地分解成3NF的集合。

1. 求R的候选键，并计算F的最小函数依赖集，然后将左边相同的FD合并
2. 对F_{min}中的每一个FD $X\to Y$，构造一个模式XY，形成模式集
3. 如果现有模式集中的每个模式都不包含原关系R的候选键，则选一个候选键构造一个关系模式，放入模式集，然后输出。否则，直接输出。

- 例：

$$R<U,F>,U=\{A,B,C,D\},F=\{A\to B,B\to C,AB\to D\}$$

1. 求R的候选键，并计算F的最小函数依赖集，然后将左边相同的FD合并
    1. 求R的候选键

$$\text{只出现在左边：}A$$
$$\text{左右都出现：}B$$
$$\text{只出现在右边：}C,D$$
$$A^+=\{ABCD\}=U$$
$$\therefore R\text{的候选键为}A$$

    2. 计算F_{min}
        1. 右侧拆分（无）
        2. 左侧拆分

$$F=\{A\to B,B\to C,A\to D\,B\to D\}$$

        3. 逐一检查各剩下的函数依赖是否可以从其他函数依赖推出

3.1删除$A\to B$，$B\notin A^+=\{AD\}$，所以$A\to B$不可删除

3.2删除$B\to C$，$C\notin B^+=\{BD\}$，所以$B\to C$不可删除

3.3删除$A\to D$，$E\in A^+=\{ABCD\}$，所以$A\to D$可以删除

3.4删除$B\to D$，$D\notin B^+=\{BC\}$，所以$B\to D$不可删除

        4. $$F_{min}=\{A\to B,B\to C,B\to D\}$$

    3. 将左边相同的FD合并

$$F=A\to B,B\to CD$$

    2. 对F_{min}中的每一个FD $X\to Y$，构造一个模式XY，形成模式集

$$R_1(AB),R_2(BCD)$$

由于R_1包含了候选键A，所以$\rho=\{R_1(AB),R_2(BCD)\}$就是一个满足条件的分解。

数据库设计与ER模型

数据库设计的优劣将直接影响信息系统的质量和运行效果。

目标：为用户和各种应用系统提供一个信息基础设施**高效率**的运行环境。

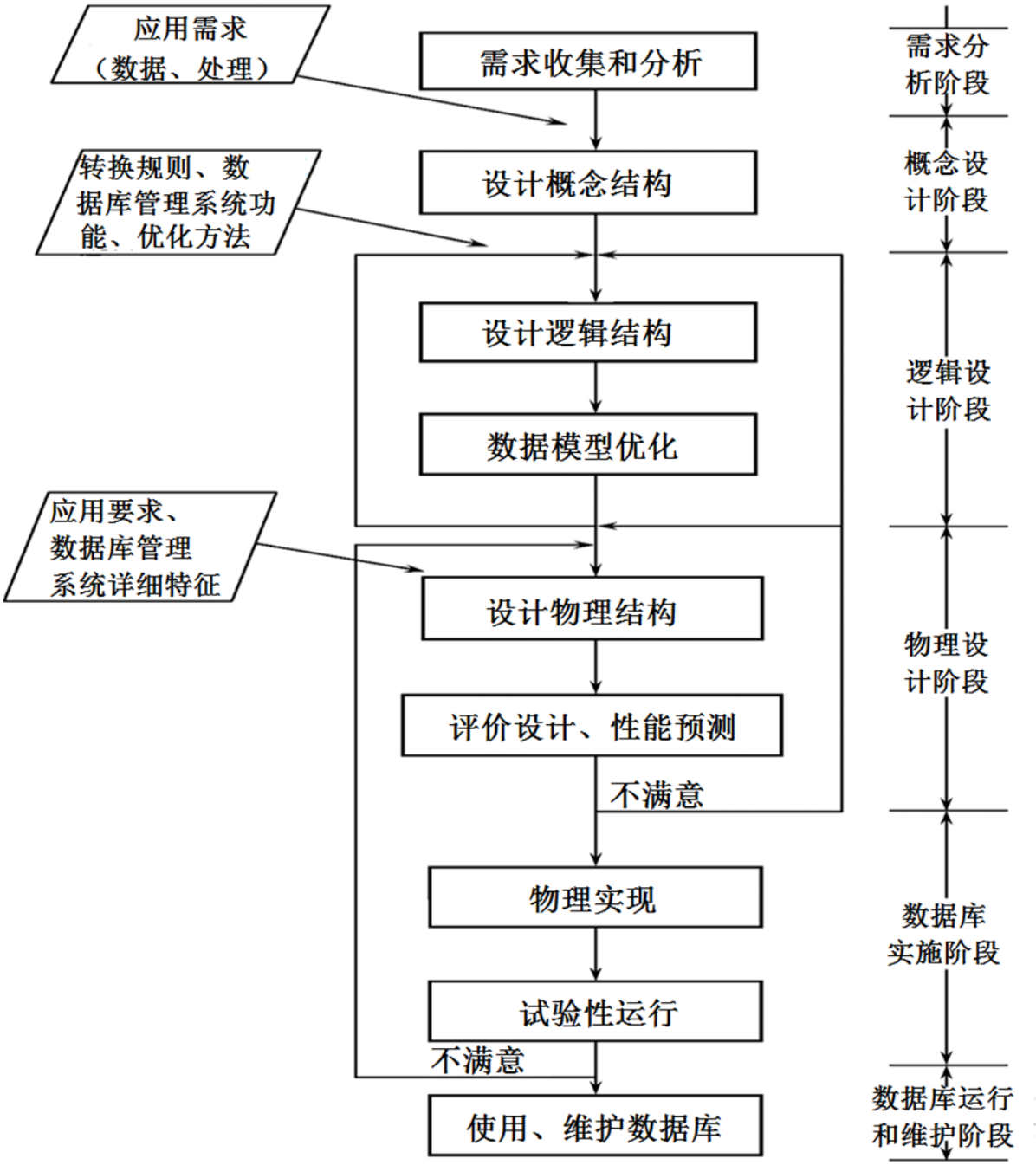
- 数据库数据的存取效率高；
- 数据库存储空间的利用率高；
- 数据库系统运行管理的效率高。

特点：结构（数据）设计和行为（处理）设计相结合。

数据库设计的全过程

软件生存期：软件开发和运营的全过程。是指从软件的规划、研制、实现、投入运行后的维护，直到它被新的软件所取代而停止使用的整个期间。

数据库系统生存期：规划、需求分析、概念设计、逻辑设计、物理设计、实现、运行维护。



规划（PPT上没有这一步）

- 系统调查：对应用单位作全面的调查，发现其存在的主要问题，并画出组织层次图，以了解企业的组织机构。
- 可行性分析：从技术、经济、效益、法律等诸方面对建立数据库的可行性进行分析，然后写出可行性分析报告，组织专家进行讨论其可行性。
- 确定数据库系统的总目标，并对应用单位的工作流程进行优化和制定项目开发计划。

需求分析

计算机人员（系统分析员）和用户双方共同收集数据库所需要的信息内容和用户对处理的需求，并以需求说明书的形式确定下来，作为以后系统开发的指南和系统验证的依据。

- 独立于任何DBMS。
- 决定了构建数据库的速度和质量。

主要步骤：

- 分析用户活动，产生业务流程图。
- 确定系统范围，产生系统关联图。
- 分析用户活动涉及的数据，产生数据流图。
- 分析系统数据，产生数据字典。

  数据字典：对数据描述的集中管理，存储和检索各种数据描述（元数据）。数据字典是进行详细的数据收集和数据分析所获得的主要成果。包括数据项、数据结构、数据流、数据存储和处理过程5个部分。

概念设计

产生反映用户信息需求的数据库概念结构（概念模型）。概念模型是独立于计算机硬件结构、独立于支持数据库的DBMS。

- 独立于任何DBMS。
- 概念设计的主要步骤
  - 进行数据抽象，设计局部概念模型。

    局部用户的信息需求是构建全局概念模型的基础。

    数据抽象方法：聚集、概括。
    - 聚集：将若干对象和它们之间的联系组合成一个新的对象。
    - 概括：将一组具有某些共同特性的对象抽象成更高一层意义上的对象。
  - 将局部概念模型综合成全局概念模型。

    综合各局部概念结构就可得到反映所有用户需求的全局概念结构。在综合过程中，主要处理**各局部模式中对各种对象定义的不一致问题**。
  - 评审

    消除了所有冲突后，就可把全局结构提交评审。评审分为**用户评审**与**DBA及应用开发人员评审**两部分。
    - 用户评审：确认全局概念模型是否准确完整地反映了用户的信息需求和现实世界事物的属性间的固有联系。
    - DBA和应用开发任意评审：确认全局结构是否完整，各种成分划分是否合理，是否存在不一致性，各种文档是否齐全等。文档应包括：局部概念结构描述、全局概念结构描述、修改后的数据清单和业务活动清单等。
- 方法：**ER方法**。

逻辑设计阶段

把概念设计阶段设计好的**概念模型**转换成与选用的具体机器上的DBMS所支持的数据模型相符合的**逻辑结构**。

- 与选用的DBMS密切相关。
- 逻辑设计的主要步骤
  - 把概念模型转换成逻辑模型

    如果概念模型用ER模型，逻辑模型采用关系模型，那么这一步就是把ER模型转换成关系模型，也就是把**ER模型**中的**实体类型和联系类型**转换成**关系模式**。
  - 设计外模型

外模型是逻辑模型的逻辑子集。是应用程序和数据库系统的接口，允许应用程序有效地访问数据库中的数据，而不破坏数据库的安全性。

- 设计应用程序与数据库的接口

  在设计完整的应用程序之前，对应用程序应设计出数据存取功能的梗概，提供应用程序与数据库之间通信的逻辑接口。

- 评价模型

  对逻辑模型进行评价。

  - 定量分析：处理频率和数据容量。处理频率是在数据库运行期间应用程序的使用次数。数据容量是数据库中记录的个数。数据库增长过程的具体表现就是这两个参数值增加。
  - 性能测量：逻辑记录的访问数目、一个应用程序传输的总字节数、数据库的总字节数。

- 修正模型

  使模型适应信息的不同表示。

物理设计阶段

数据库的物理结构：数据库的**存储记录格式、存储记录安排和存取方法。**数据库的物理设计是完全依赖于给定的硬件环境和数据库产品的。

- 与选用的DBMS密切相关。
- 评价重点：时间和空间效率。

主要步骤

- 物理结构设计：存储记录结构设计，确定数据存放位置，存取方法的设计。
  - 存取方法：B+树索引、Hash索引、聚簇索引
    - 聚簇：位于不同关系的相关元组集中存放在连续的物理块中称为聚簇。
- 约束和具体的程序设计：完整性和安全性考虑，程序设计。

数据库实施

- 用DDL定义数据库结构
- **组织数据入库**（最主要）
  - 人工方法
  - 计算机辅助数据入库
- 编制与调试应用程序
- 数据库试运行

数据库的运行与维护

- 数据库的转储和恢复。
- 数据库安全性、完整性控制。
- 数据库性能的监督、分析和改进。
- 数据库的重组织和重构造。

ER模型

由陈品山（Peter Chen）提出。

基础概念太简单不写。

ER模型到关系模式集的转换

- 实体间关系1:1，两个实体类型转换成的两个关系模式中**任意**一个关系模式的属性中加入另一个关系模式的**键和联系类型的属性**。
- 1:N，N端实体加入1端实体的键和联系类型的属性。
- M:N，则联系类型也转换成关系模式，其属性为**两端实体类型的键加上联系类型的属性**，而**键为两端实体键的组合**。

采用ER模型的逻辑设计步骤

- 导出初始关系模式集
- 规范化处理
- 模式评价
- 模式修正
- 设计子模式

ER图的集成

- 合并：解决各部分ER图之间的冲突。
    - 属性冲突：属性域冲突，即属性值的类型、取值范围或取值集合不同；属性取值单位冲突。
    - 命名冲突：同名异义、异名同义、命名冲突。
    - 结构冲突：同一对象在不同的应用中具有不同的抽象；同一实体在不同子系统的ER图中所包含的属性个数和属性排列次序不完全相同；实体间的联系在不同的ER图中为不同的类型*(两个相同的实体，在一个ER图中是多对多，另一个是一对多)*。
- 修改和重构：消除不必要的冗余，生成基本ER图。

系统实现技术

事务

构成单一逻辑工作单元的操作集合。

- 原子性
- 一致性：数据不会因事务的执行而遭受破坏 。必须使数据库状态**从一个一致状态变为另一个一致状态**。
- 隔离性：多个事务**并发**执行时，系统应保证与这些事务先后单独执行时的结果一样。
- 持久性：一个事务一旦完成全部操作后，它对数据库的所有更新应**永久地反映在数据库中**。

数据库的恢复

- 恢复的基本原则：冗余，即数据库重复存储。
- 具体实现方法

- 平时做好两件事：转储和建立日志

  周期地（比如一天一次）对整个数据库进行拷贝，转储到另一个磁盘或磁带一类存储介质中。
  建立日志数据库。记录事务的开始、结束及数据每一次插入、删除和修改前后的值，并写到“日志”库中。

- 一旦发生数据库故障，分两种情况进行处理
  如果数据库已被破坏，则装入last数据库备份，再利用日志库将这两个数据库状态之间的所有更新重新做一遍。
  如果数据库未被破坏，但某些数据不可靠，则撤消所有不可靠的修改，把数据库恢复到正确的状态。

故障类型和恢复方法

- 事务故障

  可以预期的事务故障：如存款余额透支等；可在事务代码中加入判断和ROLLBACK，这时执行回退操作（UNDO）
  非预期事务故障：如运算溢出、数据错误、死锁等；系统直接执行UNDO处理

- 系统故障

  硬件故障、软件错误或掉电等等；内存内容丢失，但不破坏数据库；恢复子系统在系统重新启动时，分2种情况处理：对未完事务，进行UNDO，对已提交事务但更新还在缓冲的事务，进行REDO

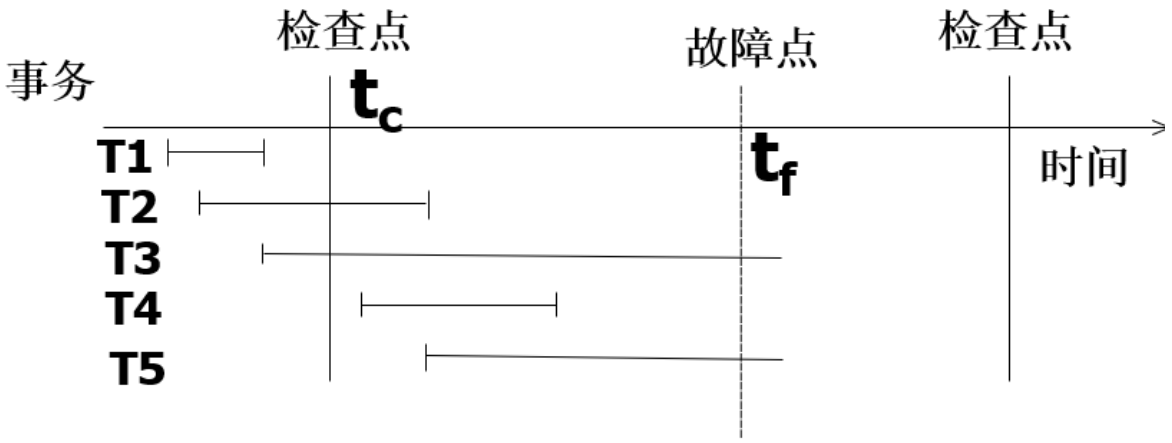
- 介质故障(硬故障)

  磁盘物理故障或遭受病毒破坏

检查点技术

DBMS定时设置检查点，在检查时刻才真正做到把DB的修改写到磁盘，并在日志文件中写入一条检查点记录（以便恢复时使用）。当DB需要恢复时，只有那些在检查点后面的事务需要恢复。

大大减少了DB的恢复时间。



如何处理这五类事务？

事务T1不必恢复；

事务T2和事务T4必须重做（REDO）；

事务T3和事务T5必须撤消（UNDO）

数据库并发控制

如果没有并发控制子系统对事物进行控制，则并发操作通常会带来四个问题

- 丢失更新
- 读脏数据
- 错误求和
- 不可重复读：两次读取同一个值，得到的结果不同

解决办法：封锁技术

- X锁：排他型封锁。如果事务T对某个数据R实现了X锁，那么T对数据R解除封锁之前，不允许其他事务T再对该数据加任何类型的锁。
- S锁：共享型封锁。事务T对某数据加上S锁之后，仍允许其他事务再对该数据加S锁。但在对该数据的所有S锁都解除之前，不允许任何事物对该数据加X锁。

- 封锁的粒度：封锁对象的大小。封锁的粒度越大，系统中能被封锁的对象就越少，并发度就越少，但同时系统的开销也就越小。
- 活锁：某个事物一直处于等待状态，得不到封锁的机会。
- 饥饿：每个事务都申请对某数据项加S锁，且每个事务在授权加锁后一小段时间内释放封锁。此时若有另外一个事务T2想要在该数据项上加X锁，则永远轮不上封锁的机会。
- 死锁：两个或两个以上的事务都处于等待状态，且每个事务都在等待其中另一个事务解除封锁，才能继续执行下去，结果造成任何一个事务都无法继续执行。检测方法：事务依赖图是否有环。

数据库的完整性

数据的正确性、有效性和相容性。

> 例如：学生的学号必须唯一，性别只能是男或女，本科生年龄的取值范围为14-30的整数。

完整性约束

- 域约束：涉及某个域

> 定义一个新的域COLOR，可用下列语句实现：
> CONSTRAINT VALID_COLORS
> CHECK (VALUE IN
> ('Red', 'Yellow', 'Blue', 'Green', '??? ')) ;

- 基本表约束：涉及一个基本表

> 候选键的定义
> UNIQUE (<列名序列>) 或 PRIMARY KEY(<列名序列>)
> 外键的定义
> FOREIGN KEY (<列名序列>)
> REFERENCES <参照表> [(<列名序列>)]
> [ON DELETE <参照动作>]
> [ON UPDATE <参照动作>]
> "检查约束"的定义
> CHECK（〈条件表达式〉）
>
> no action 同 restrict

- 断言：涉及多个关系

数据库的安全性

完整性措施：防范不合语义的数据，保护数据以防止**合法用户无意**中造成的破坏。

安全性措施：防范非法用户和非法操作，保护数据以防止**非法用户故意**造成的破坏。

安全性级别：

- 环境级：计算机系统的机房和设备加以保护，防止物理破坏。
- 职员级：正确授予用户访问数据库的权限
- OS级：防止未经授权的用户从OS处着手访问数据库
- 网络级：由于大多数DBS都允许用户通过网络进行远程访问，因此网络软件内部的安全性很重要。
- DBS级：DBS的职责是检查用户的身份是否合法及使用数据库的权限是否正确。