数据库系统概论

An Introduction to Database System

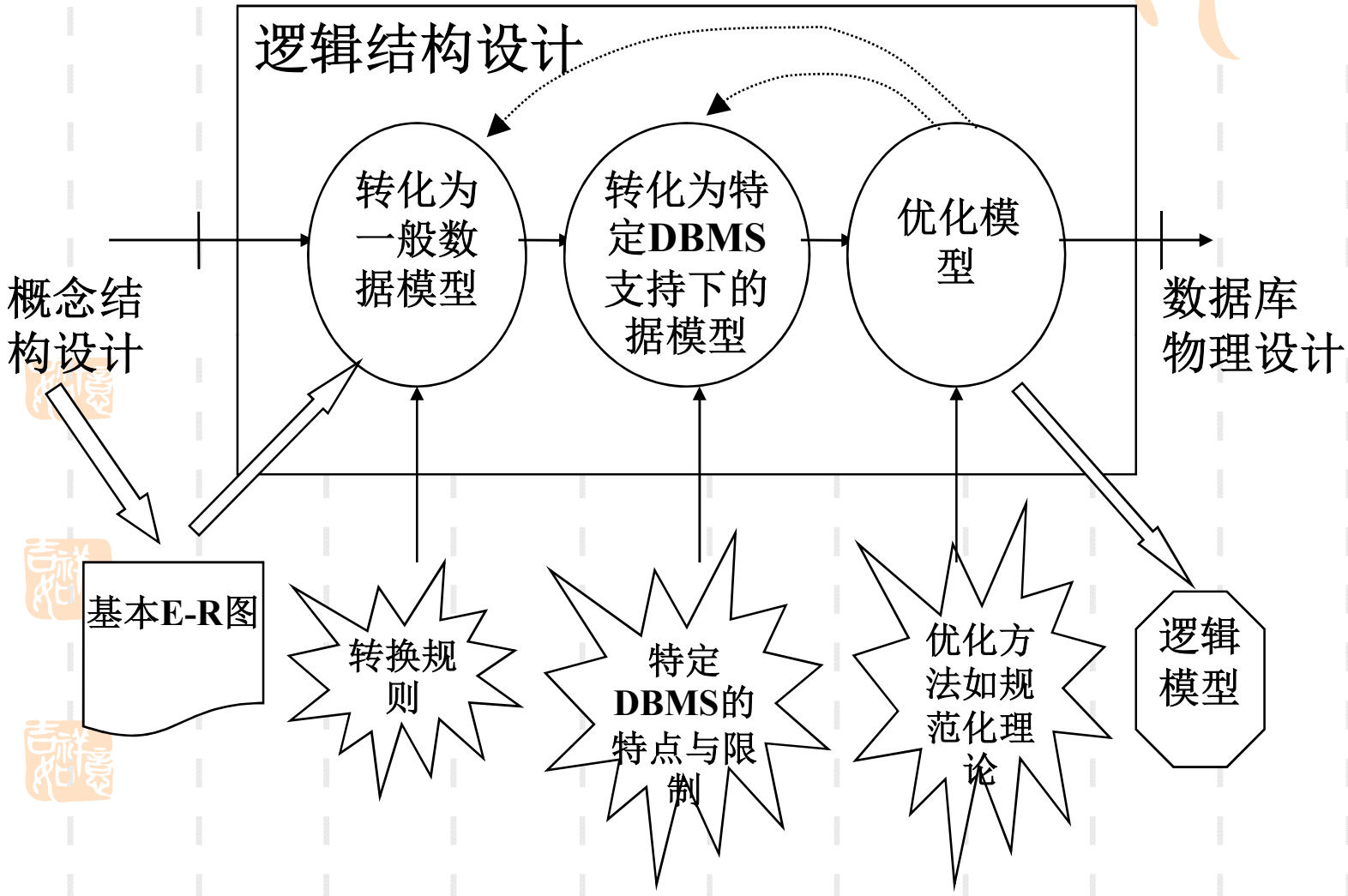
第七章　数据库设计(续2)

第七章 数据库设计

7.4 逻辑结构设计

- 逻辑结构设计的任务
  - 把概念结构设计阶段设计好的基本E-R图转换为与选用DBMS产品所支持的数据模型相符合的逻辑结构
- 逻辑结构设计的步骤
  - 将概念结构转化为一般的关系、对象关系模型
  - 将转换来的关系、对象关系模型向特定DBMS支持下的数据模型转换
  - 对数据模型进行优化

逻辑结构设计(续)


逻辑结构设计
概念结构设计
转化为一般数据模型
转化为特定DBMS支持下的据模型
优化模型
数据库物理设计
基本E-R图
转换规则
特定DBMS的特点与限制
优化方法如规范化理论
逻辑模型

7.4 逻辑结构设计

7.4.1 E-R图向关系模型的转换

7.4.2 数据模型的优化

7.4.3 设计用户子模式

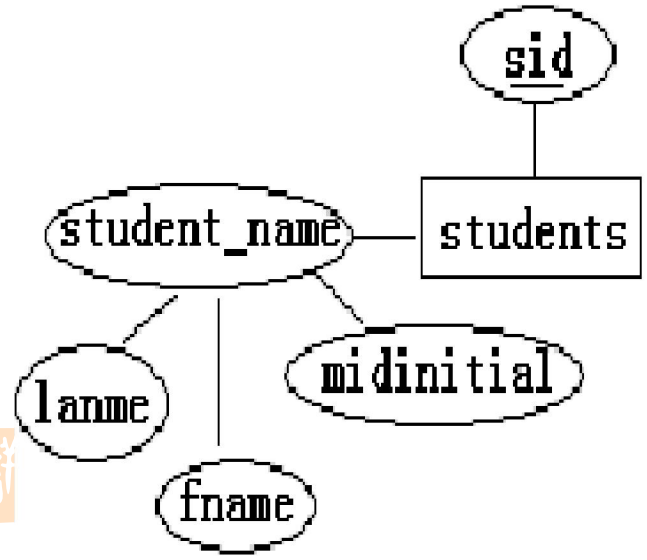
E-R图向关系模型的转换

- E-R图向关系模型的转换要解决的问题
  - 如何将实体型和实体间的联系转换为关系模式
  - 如何确定这些关系模式的属性和码
- 转换内容
  - 将E-R图转换为关系模型：将实体、实体的属性和实体之间的联系转换为关系模式。

E-R 图→关系表

- 转换规则1：
  - E-R 图中每个实体转换为一张表
  - 表名用实体名命名
  - 单值属性转换为一列
  - 复合属性转换为多列
  - 标识符转换为主键.

Example



students

sid	lname	fname	midinitial
1134	Smith	John	L.
...	...	...	...

E-R 图→关系表

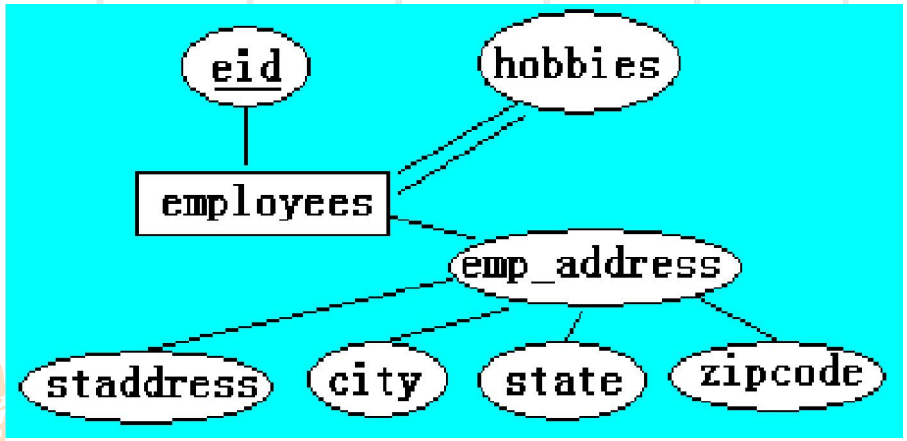
转换规则2：

给定实体E有主标识符P，和多值属性

- 多值属性转换为单独的表
- 表用多值属性命名。（如果可能表名用复数形式命名）
- 这张表的列由主标识符P和多值属性组成（ P和多值属性都可以是属性组）
- 这张表的主键由主标识符P和多值属性组成

Example

- Translate the *employees* entities and the attached multi-valued attribute, *hobbies* , to two table.



employees

eid	staddress	city	state	zipcode
197	7 Beacon St	Boston	Ma	2122
221	19 Brighton	Boston	Ma	2103
…	…	…	…	…

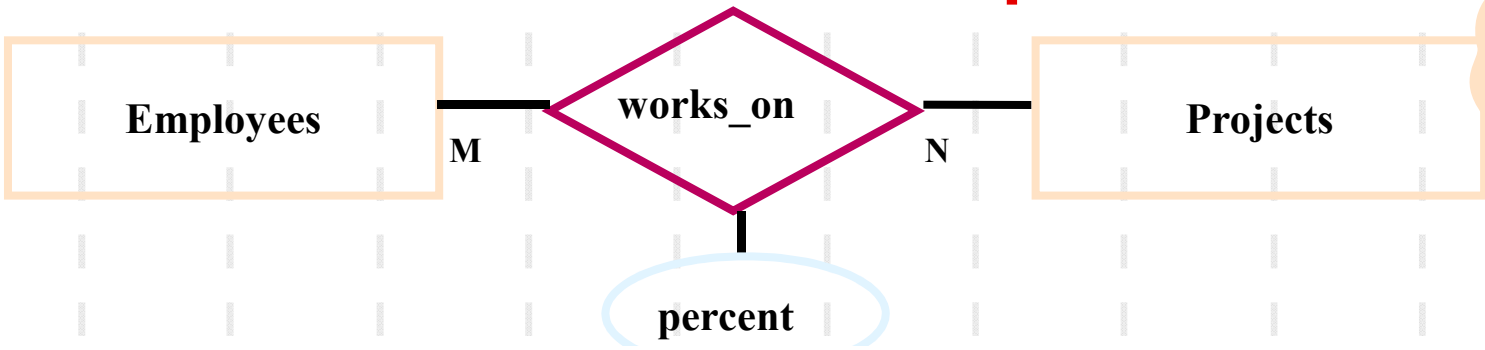
hobbies

eid	hobby
197	chess
197	painting
197	dancing
221	reading
…	…

E-R 图→关系表

- 转换规则3：
- 实体E 和 F的 N-N关系R →表T
  - 实体E 和 F的主键→表T的主键
  - 关系的属性→表T的列

Example



employees

eid	straddr	city	state	zipcode
197	7 Beacon St	Boston	Ma	02102
221	19 Brighton ST	Boston	Ma	02103
303	153 Mass Ave	Cambridge	Ma	02103

work_on

eid	prid	percent
197	p11	50
197	p13	25
197	p21	25
221	p21	100
303	p13	40
303	p21	60

projects

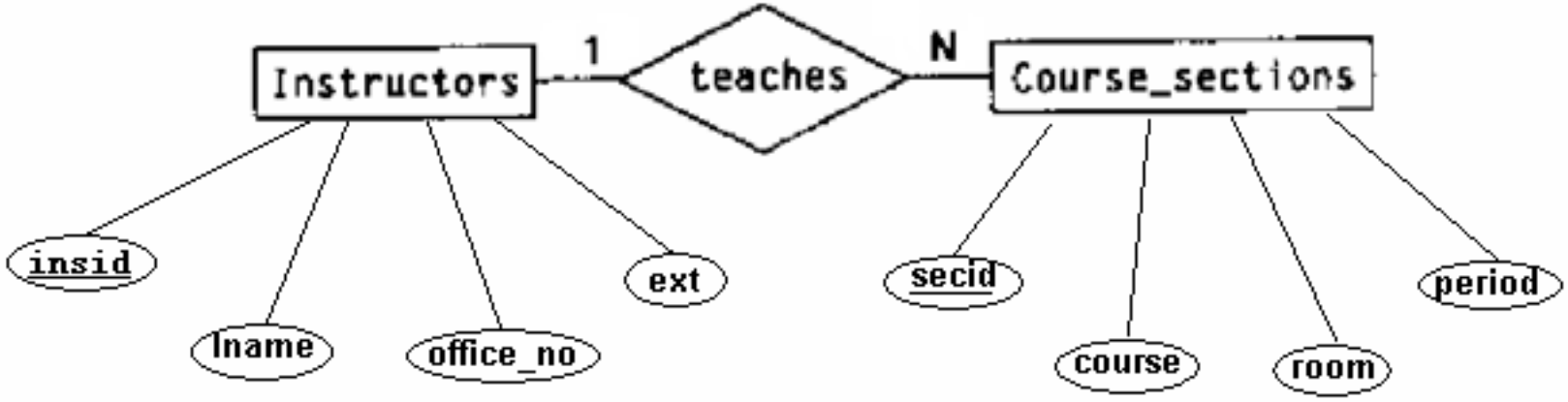
prid	proj_name	due_date
p11	Phoenix	3/31/99
p13	Excelsior	9/31/99
p21	White Mouse	6/30/00

E-R 图→关系表

- 转换规则4：
- 一个1:n联系可以转换为一个独立的关系模式，也可以与n端对应的关系模式合并。
  - 与n端对应的关系模式合并

    如果实体E 和 F 的 N-1关系R 不转换为单独的表，那么代表多方的实体F所转换的表T应该包括：
    - 实体F的标识符→表T的主键
    - 实体E的标识符→表T的外键
  - 转换为一个独立的关系模式
    - 实体E 和 F的主键及关系的属性→表T的列
    - 代表多方的实体F的主键→表T的主键

Example



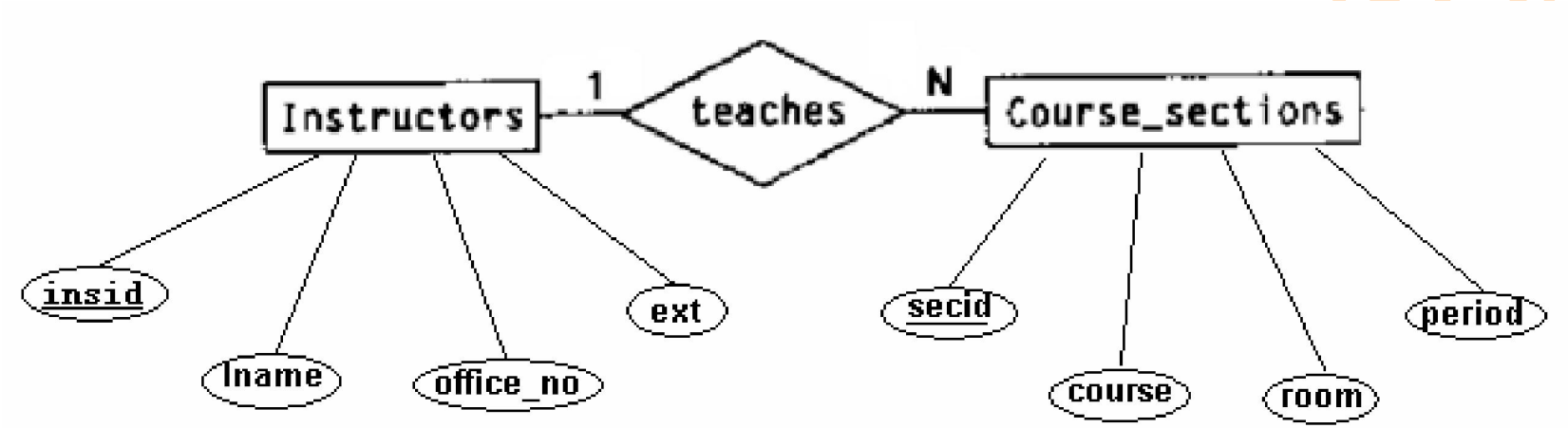
instructors

insid	lname	office_no	ext
309	O'Neil	s-3-223	78543
123	Bergen	s-3-547	78413
113	Smith	s-3-115	78455
…	…	…	…

course_sections

secid	insid	course	room	period
120	309	cs240	m-1-213	mw6
940	309	cs630	m-1-214	mw7
453	123	cs632	m-2-614	tth6
…	…	…	…	…

Example



instructors

insid	lname	office_no	ext
309	O'Neil	s-3-223	78543
123	Bergen	s-3-547	78413
113	Smith	s-3-115	78455
…	…	…	…

course sections

secid	course	room	period
120	cs240	m-1-213	mw6
940	cs630	m-1-214	mw7
453	cs632	m-2-614	tth6
…	…	…	…

Teach

secid	insid
120	309
940	309
453	123
…	…

E-R 图→关系表

- 转换规则5：
- 一个1:1联系可以转换为一个独立的关系模式，也可以与任意一端对应的关系模式合并

①转换为一个独立的关系模式

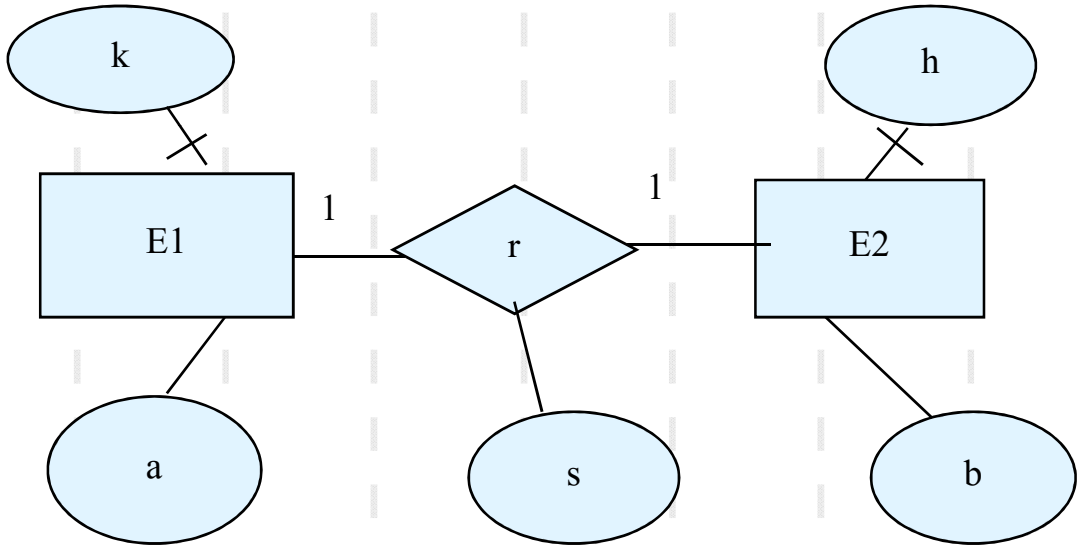
- 关系的属性：与该联系相连的各实体的码以及联系本身的属性
- 关系的候选码：每个实体的码均是该关系的候选码

②与某一端实体对应的关系模式合并

- 合并后关系的属性：加入对应关系的码和联系本身的属性
- 合并后关系的码：不变

E-R 图→关系表

- 转换规则5：
  - 与一端对应的关系模式合并
    - R1(k, a,h,s)
    - R2(h,b)
  - 独立的关系模式
    - R1(k, a,)
    - R2(h,b)
    - R3(k, h, s)



E-R图向关系模型的转换（续）

转换规则6：

三个或三个以上实体间的一个多元联系转换为一个关系模式。

例，“讲授”联系是一个三元联系，可以将它转换为如下关系模式，其中课程号、职工号和书号为关系的组合码：

讲授（课程号，职工号，书号）

E-R图向关系模型的转换（续）

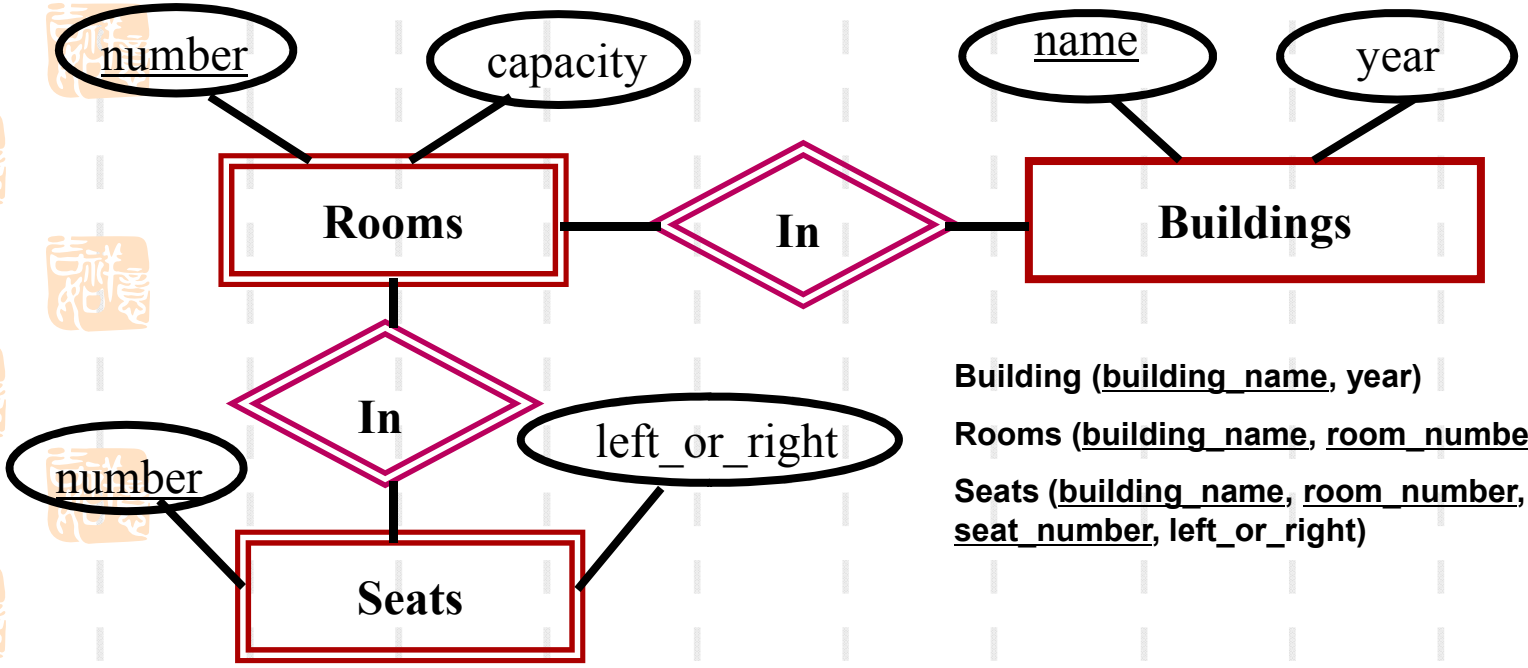
转换规则7

具有相同码的关系模式可合并

- 目的：减少系统中的关系个数
- 合并方法：将其中一个关系模式的全部属性加入到另一个关系模式中，然后去掉其中的同义属性（可能同名也可能不同名），并适当调整属性的次序

弱实体转换

- 转换规则8
  - Remember the ‘borrowed’ key attributes
  - Watch out for attribute name conflicts

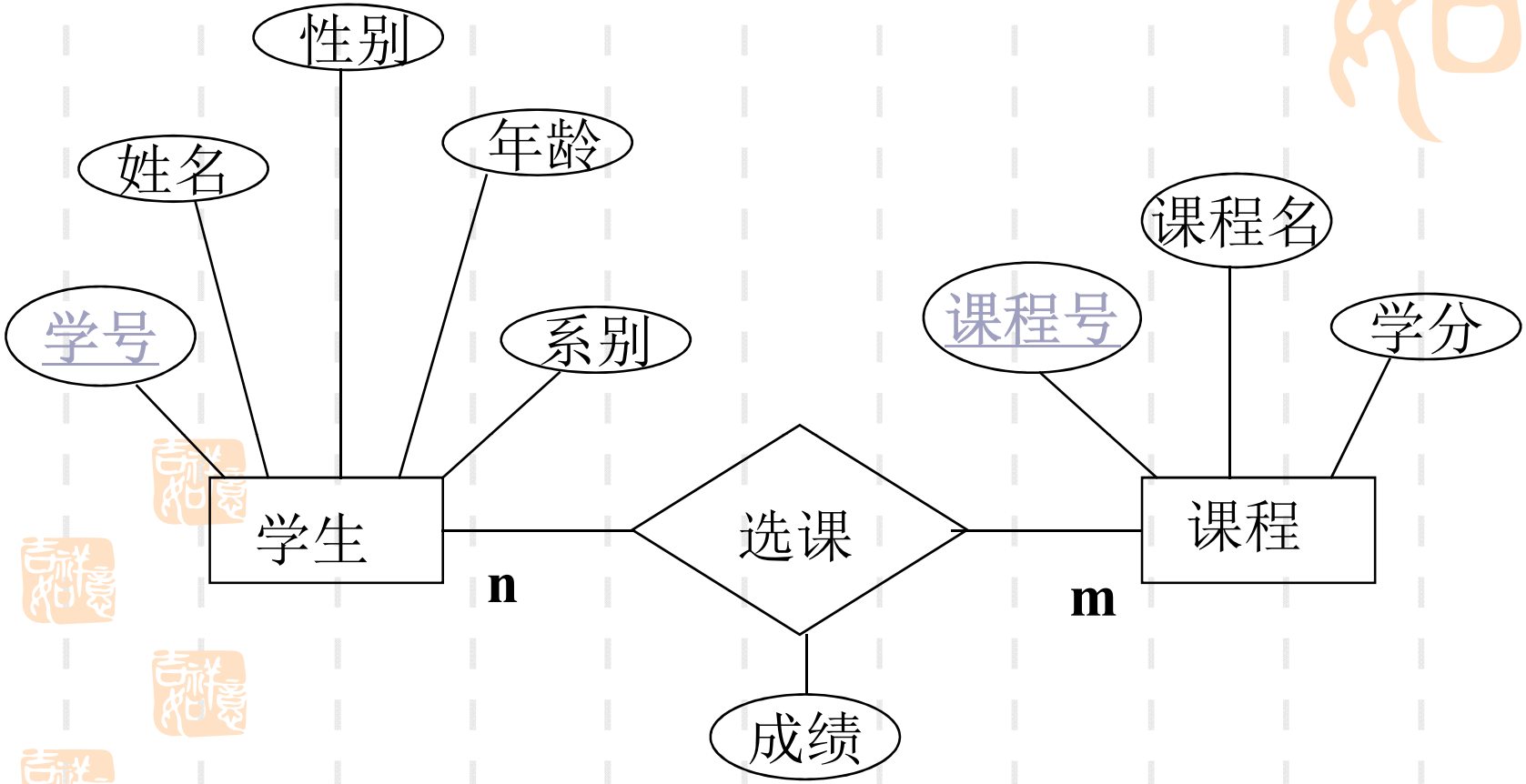


Building (building_name, year)

Rooms (building_name, room_number, capacity)

Seats (building_name, room_number, seat_number, left_or_right)

E-R→TBALE

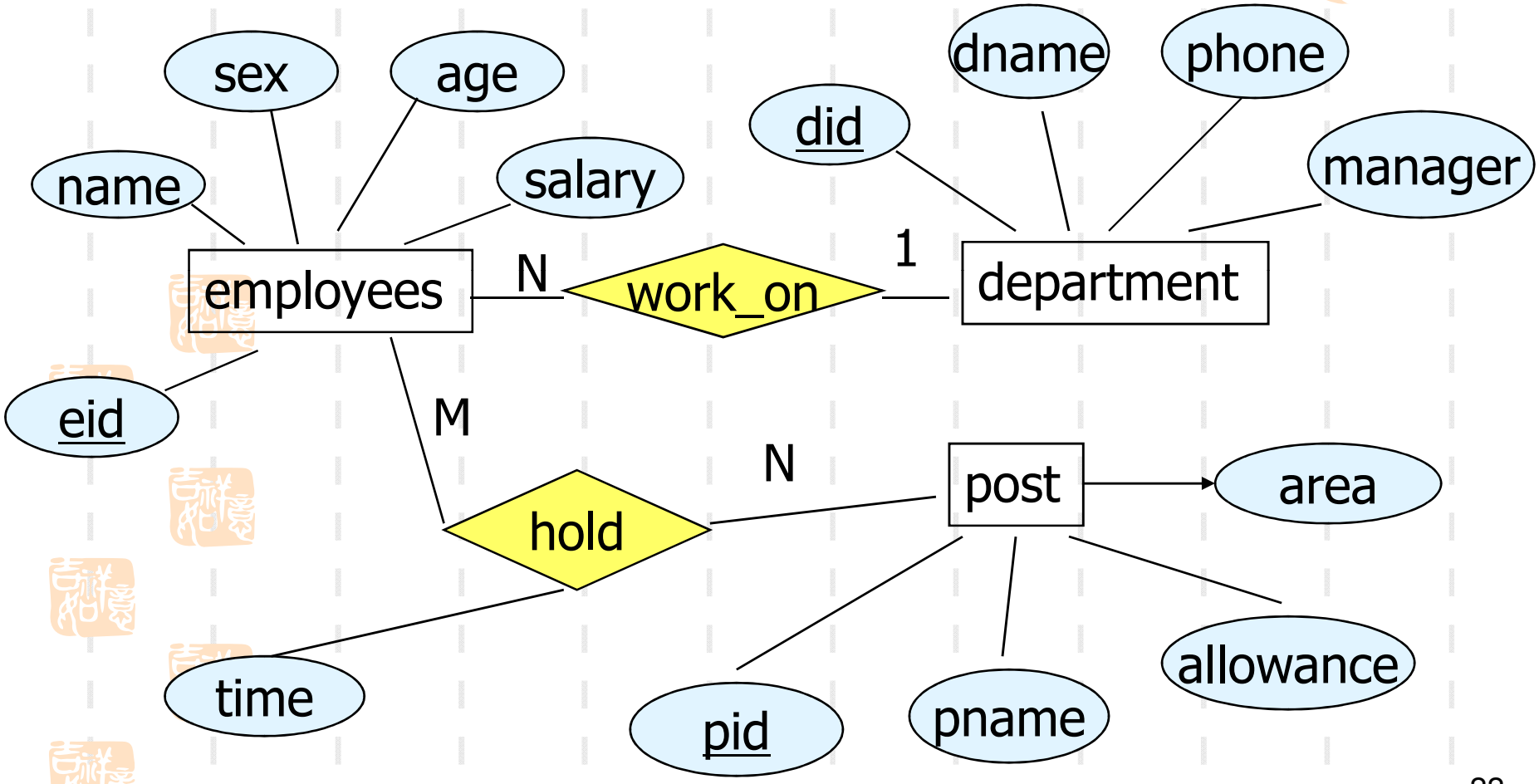


学生（学号，姓名，性别，年龄，系别）

课程（课程号，课程名，学分）

选课（学号，课程号，成绩）

Example

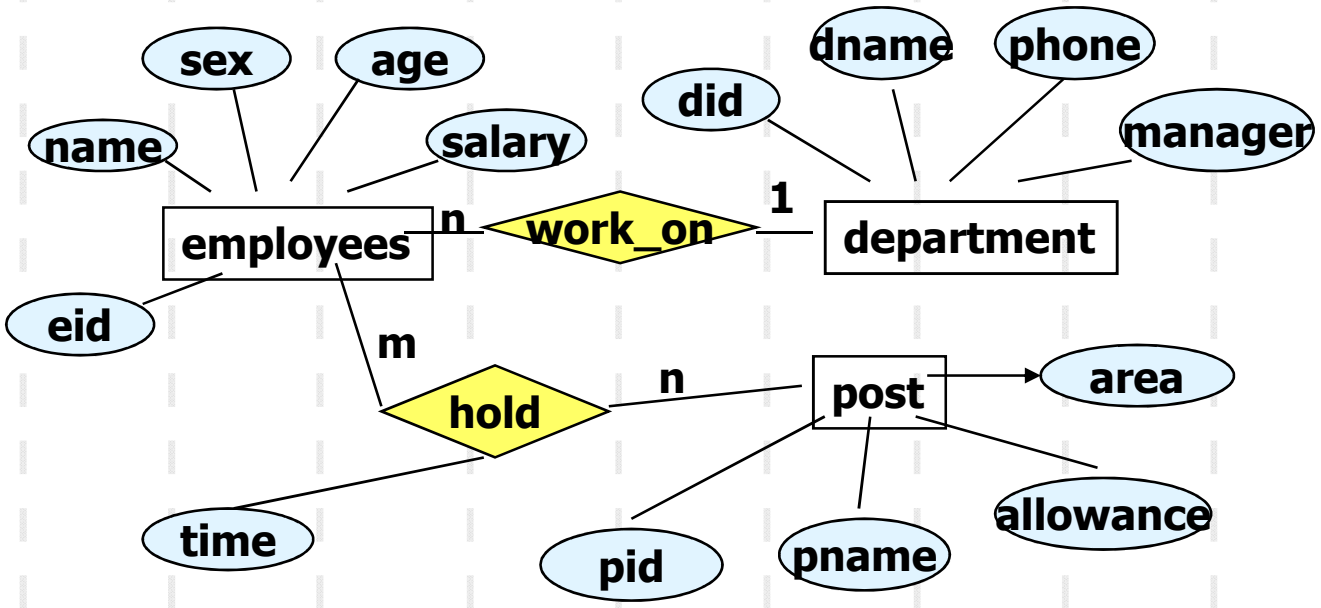

sex
age
name
salary
employees
N
work_on
1
department
dname
phone
did
manager
eid
M
hold
N
post
area
time
pid
pname
allowance

Entities:

- **Employees(eid,name,sex,age,salary,did)**
- **Departments(did,dname,phone,manager)**
- **Posts(pid,pname,allowance,area)**

Relationship:

- **Hold(eid,pid,time)**



案例学习(1)

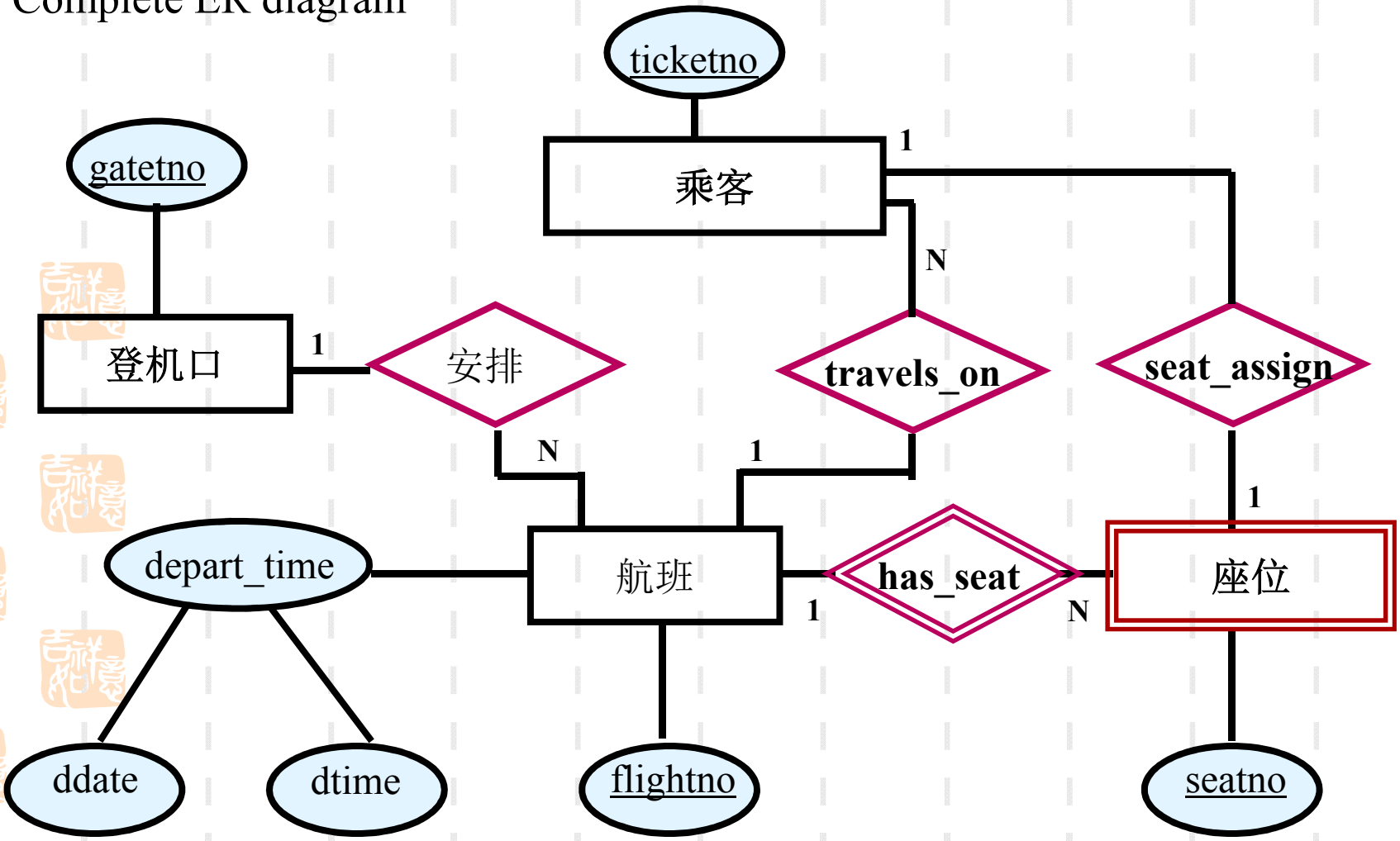
- 一个简单的航空公司预订数据库:乘客,航班,登机口, 座位分配.
- 每一个航班有一个 标识符 **flightno**, 和属性起飞时间 **depart_time** (由日期和时间组成)
- 每一个乘客有一个 标识符 **ticketno**.
- 每一个座位,有一个 标识符 (**seatno**), 只对一个特定的航班有效– 弱实体.
- 座位分配**Seat assignment**是乘客和座位 的联系
- 每一个登机口被安排给多个航班(在不同的时间),它们之间是“安排”联系

案例学习(2)

- 联系”安排”（1：N）
  - 每一个登机口可以被安排给多个航班（不同时间）
  - 每一个航班只对应一个登机口
- 联系 ”travels_on” （1：N）
  - 每一个航班可以有多个乘客
  - 每一个乘客必须而且只能在一个航班上
- 联系 has_seat （1：N）
  - 每一个航班可以有多个座位
  - 每一个座位只对应一个航班.
- 联系 seat_assign （1：1）
  - 每一个乘客必须而且只能有一个座位
  - 每一个座位最多一个乘客，可以是空的.

案例学习(3)

Complete ER diagram



案例学习(4)

- Translating entities :
  - 乘客*(ticketno)*
  - 登机口(*gateno*)
  - 航班(*flightno,ddate,dtime)*
  - 座位*(seatno, flightno)* –座位 is a weak entity
- Translating relationship *seat_assign*:
  - 乘客*(ticketno,seatno,flightno)*
- Translating relationship *marshals*：
  - 航班*(flightno,gateno,ddate,dtime)*
- How to do with the relationship *travels_on* and *has_seat* ?

E-R图向关系模型的转换（续）

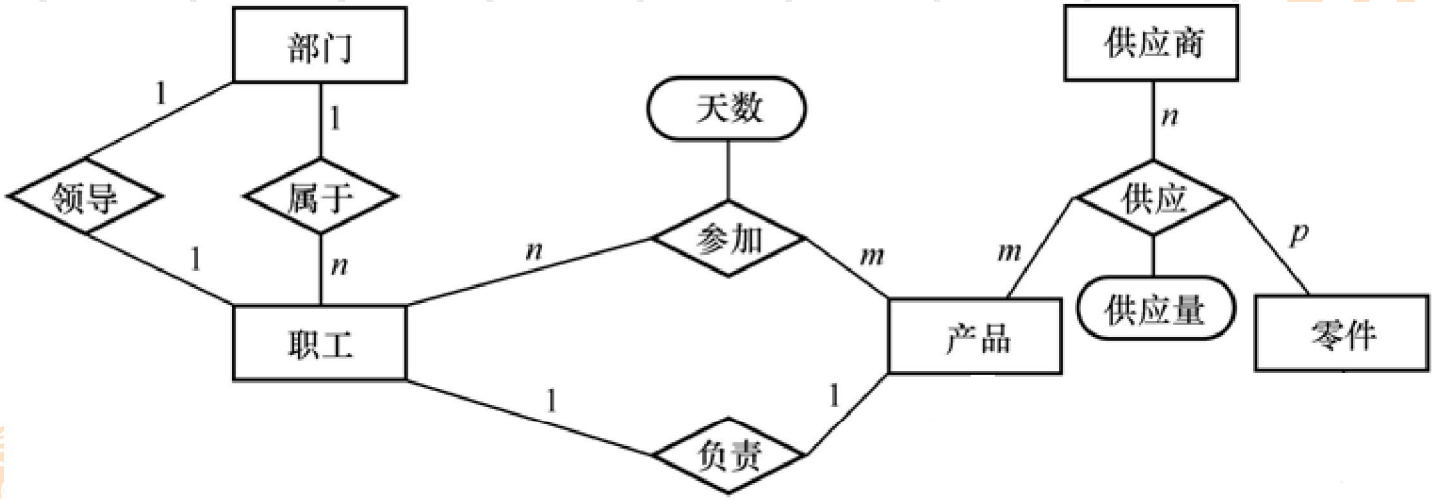
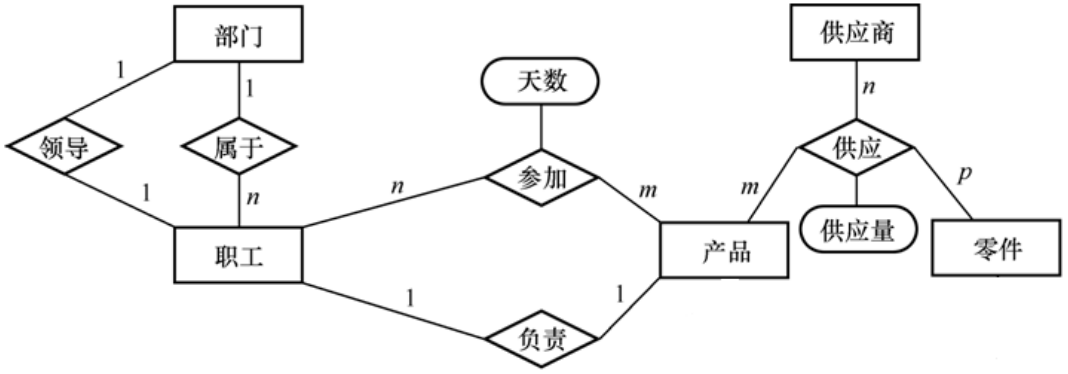


图7.28中虚线上部的E-R图转换为关系模型。

关系的码用下横线标出。

E-R图向关系模型的转换（续）

- 部门（部门号，部门名，经理的职工号，…）
- 职工（职工号、部门号，职工名，职务，…）
- 产品（产品号，产品名，产品组长的职工号，…）
- 供应商（供应商号，姓名，…）
- 零件（零件号，零件名，…）
- 职工工作（职工号，产品号，工作天数，…）
- 供应（产品号，供应商号，零件号，供应量）



7.4 逻辑结构设计

7.4.1 E-R图向关系模型的转换

7.4.2 数据模型的优化

7.4.3 设计用户子模式

7.4.2 数据模型的优化

- 得到初步数据模型后，还应该适当地修改、调整数据模型的结构，以进一步提高数据库应用系统的性能，这就是数据模型的优化
- 关系数据模型的优化通常以规范化理论为指导

数据模型的优化（续）

优化数据模型的方法

1. 确定数据依赖

   按需求分析阶段所得到的语义，分别写出每个关系模式内部各属性之间的数据依赖以及不同关系模式属性之间数据依赖

2. 消除 冗余的联系

   对于各个关系模式之间的数据依赖进行极小化处理，消除冗余的联系。

3. 确定所属范式

   - 按照数据依赖的理论对关系模式逐一进行分析
   - 考查是否存在部分函数依赖、传递函数依赖、多值依赖等
   - 确定各关系模式分别属于第几范式

数据模型的优化（续）

4. 按照需求分析阶段得到的各种应用对数据处理的要求，分析对于这样的应用环境这些模式是否合适，确定是否要对它们进行合并或分解。

注意：并不是规范化程度越高的关系就越优，一般说来，第三范式就足够了

- 当查询经常涉及两个或多个关系模式的属性时，系统必须经常地进行连接运算
- 连接运算的代价是相当高的
- 因此在这种情况下，第二范式甚至第一范式也许是适合的。

数据模型的优化（续）

例：在关系模式

学生成绩单(学号,英语,数学,语文,平均成绩) 中存在下列函数依赖：

学号→英语

学号→数学

学号→语文

学号→平均成绩

(英语, 数学, 语文)→平均成绩

数据模型的优化（续）

显然有：

学号→(英语,数学,语文)

因此该关系模式中存在传递函数信赖，是2NF关系

虽然平均成绩可以由其他属性推算出来，但如果应用中需要经常查询学生的平均成绩，为提高效率，仍然可保留该冗余数据，对关系模式不再做进一步分解

数据模型的优化（续）

5. 按照需求分析阶段得到的各种应用对数据处理的要求，对关系模式进行必要的分解，以提高数据操作的效率和存储空间的利用率

- 常用分解方法
  - 水平分解
  - 垂直分解

数据模型的优化（续）

- 水平分解
  - 什么是水平分解
    - 把(基本)关系的元组分为若干子集合，定义每个子集合为一个子关系，以提高系统的效率。
  - 如何分解
    - ✓满足“80/20原则”的应用，把经常被使用的数据（约20%）

      水平分解出来，形成一个子关系。
    - ✓水平分解为若干子关系，使每个事务存取的数据对应一个子关系。

数据模型的优化（续）

- 垂直分解
  - 什么是垂直分解
    - 把关系模式R的属性分解为若干子集合，形成若干子关系模式。
  - 垂直分解的原则
    - 经常在一起使用的属性从R中分解出来形成一个子关系模式
  - 垂直分解的优点
    - 可以提高某些事务的效率
  - 垂直分解的缺点
    - 可能使另一些事务不得不执行连接操作，降低了效率

数据模型的优化（续）

- 垂直分解的适用范围
  - 取决于分解后R上的所有事务的总效率是否得到了提高
- 进行垂直分解的方法
  - 简单情况：直观分解
  - 复杂情况：用第6章中的模式分解算法
  - 垂直分解必须不损失关系模式的语义（保持无损连接性和保持函数依赖）

7.4 逻辑结构设计

7.4.1 E-R图向关系模型的转换

7.4.2 数据模型的优化

7.4.3 设计用户子模式

7.4.3 设计用户子模式

- 定义数据库模式主要是从系统的时间效率、空间效率、易维护等角度出发。
- 定义用户外模式时应该更注重考虑用户的习惯与方便。包括三个方面：

设计用户子模式（续）

（1）使用更符合用户习惯的别名

- 合并各分E-R图曾做了消除命名冲突的工作，以使数据库系统中同一关系和属性具有唯一的名字。这在设计数据库整体结构时是非常必要的。
- 用视图机制可以在设计用户视图时可以重新定义某些属性名，使其与用户习惯一致，以方便使用。

设计用户子模式（续）

（2）针对不同级别的用户定义不同的视图，以保证系统的安全性。

- 假设有关系模式产品（产品号，产品名，规格，单价，生产车间，生产负责人，产品成本，产品合格率，质量等级），可以在产品关系上建立两个视图：
  - 为一般顾客建立视图：
    产品1（产品号，产品名，规格，单价）
  - 为产品销售部门建立视图：
    产品2（产品号，产品名，规格，单价，车间，生产负责人）

设计用户子模式（续）

（3）简化用户对系统的使用

- 如果某些局部应用中经常要使用某些很复杂的查询，为了方便用户，可以将这些复杂查询定义为视图。

设计用户子模式（续）

[例] 关系模式产品（产品号，产品名，规格，单价，生产车间，生产负责人，产品成本，产品合格率，质量等级），可以在产品关系上建立两个视图：

为一般顾客建立视图：

产品1（产品号，产品名，规格，单价）

为产品销售部门建立视图：

产品2（产品号，产品名，规格，单价，车间，生产负责人）

- 顾客视图中只包含允许顾客查询的属性
- 销售部门视图中只包含允许销售部门查询的属性
- 生产领导部门则可以查询全部产品数据
- 可以防止用户非法访问不允许他们查询的数据，保证系统的安全性

逻辑结构设计小结

- 任务
  - 将概念结构转化为具体的数据模型
- 逻辑结构设计的步骤
  - 将概念结构转化为关系模型
  - 对数据模型进行优化
  - 设计用户子模式

第七章 数据库设计

7.5 数据库的物理设计

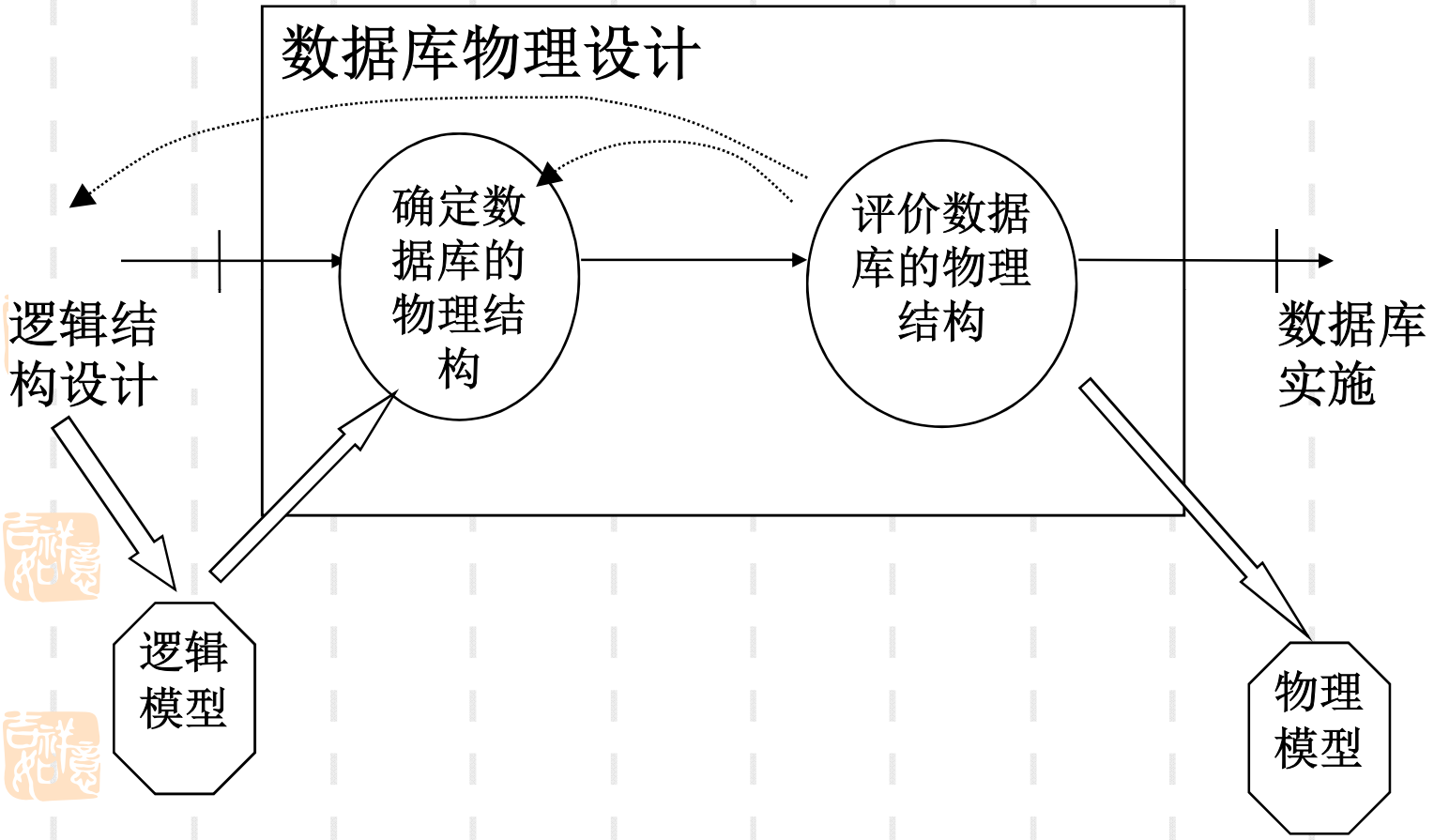
- 数据库的物理设计
  - 数据库在物理设备上的存储结构与存取方法称为数据库的物理结构，它依赖于选定的数据库管理系统
  - 为一个给定的逻辑数据模型选取一个最适合应用环境的物理结构的过程，就是数据库的物理设计

数据库的物理设计(续)

- 数据库物理设计的步骤
  - 确定数据库的物理结构，在关系数据库中主要指存取方法和存储结构
  - 对物理结构进行评价，评价的重点是时间和空间效率

  如果评价结果满足原设计要求，则可进入到物理实施阶段，否则，就需要重新设计或修改物理结构，有时甚至要返回逻辑设计阶段修改数据模型

数据库的物理设计(续)



7.5.1 数据库物理设计的内容和方法

- 设计物理数据库结构的准备工作
  - 充分了解应用环境，详细分析要运行的事务，以获得选择物理数据库设计所需参数
  - 充分了解所用RDBMS的内部特征，特别是系统提供的存取方法和存储结构

数据库的物理设计的内容和方法（续）

- 选择物理数据库设计所需参数
  - 数据库查询事务
    - 查询的关系
    - 查询条件所涉及的属性
    - 连接条件所涉及的属性
    - 查询的投影属性

数据库的物理设计的内容和方法（续）

- 选择物理数据库设计所需参数(续)
  - 数据更新事务
    - 被更新的关系
    - 每个关系上的更新操作条件所涉及的属性
    - 修改操作要改变的属性值
  - 每个事务在各关系上运行的频率和性能要求

数据库的物理设计的内容和方法（续）

- 关系数据库物理设计的内容
  - 为关系模式选择存取方法(建立存取路径)
  - 设计关系、索引等数据库文件的物理存储结构

7.5.2 关系模式存取方法选择

- 数据库系统是多用户共享的系统，对同一个关系要建立多条存取路径才能满足多用户的多种应用要求
- 物理设计的任务之一就是要确定选择哪些存取方法，即建立哪些存取路径

关系模式存取方法选择（续）

- 数据库管理系统常用存取方法
  1. B+树索引存取方法
  2. Hash索引存取方法
  3. 聚簇存取方法

1. B+树索引存取方法的选择

- 选择索引存取方法的主要内容
  - 根据应用要求确定
    - 对哪些属性列建立索引
    - 对哪些属性列建立组合索引
    - 对哪些索引要设计为唯一索引

B+树索引存取方法的选择（续）

- 选择索引存取方法的一般规则
  - 如果一个（或一组）属性经常在查询条件中出现，则考虑在这个（或这组）属性上建立索引（或组合索引）
  - 如果一个属性经常作为最大值和最小值等聚集函数的参数，则考虑在这个属性上建立索引
  - 如果一个（或一组）属性经常在连接操作的连接条件中 出现，则考虑在这个（或这组）属性上建立索引

B+树索引存取方法的选择（续）

- 关系上定义的索引数过多会带来较多的额外开销
  - 维护索引的开销
  - 查找索引的开销

2. HASH存取方法的选择

- 选择Hash存取方法的规则
  - 如果一个关系的属性主要出现在等值连接条件中或主要出现在等值比较选择条件中，而且满足下列两个条件之一
    - 该关系的大小可预知，而且不变；
    - 该关系的大小动态改变，但所选用的数据库管理系统提供了动态Hash存取方法。

3. 聚簇存取方法的选择

- 什么是聚簇
  - 为了提高某个属性（或属性组）的查询速度，把这个或这些属性（称为聚簇码）上具有相同值的元组集中存放在连续的物理块中称为聚簇。
  - 该属性（或属性组）称为聚簇码（cluster key）
  - 许多关系型数据库管理系统都提供了聚簇功能
  - 聚簇存放与聚簇索引的区别

聚簇存取方法的选择（续）

- 聚簇索引
  - 建立聚簇索引后，基表中数据也需要按指定的聚簇属性值的升序或降序存放。也即聚簇索引的索引项顺序与表中元组的物理顺序一致。
  - 在一个基本表上最多只能建立一个聚簇索引
- 聚簇索引的适用条件
  - 很少对基表进行增删操作
  - 很少对其中的变长列进行修改操作

聚簇存取方法的选择（续）

- 聚簇的用途
  - 对于某些类型的查询，可以提高查询效率

    1. 大大提高按聚簇属性进行查询的效率

[例] 假设学生关系按所在系建有索引，现在要查询信息系的所有学生名单。

- 计算机系的500名学生分布在500个不同的物理块上时，至少要执行500次I/O操作。
- 如果将同一系的学生元组集中存放，则每读一个物理块可得到多个满足查询条件的元组，从而显著地减少了访问磁盘的次数。

聚簇存取方法的选择（续）

2. 节省存储空间

- 聚簇以后，聚簇码相同的元组集中在一起了，因而聚簇码值不必在每个元组中重复存储，只要在一组中存一次就行了。

聚簇存取方法的选择（续）

- 聚簇的局限性
  - 聚簇只能提高某些特定应用的性能
  - 建立与维护聚簇的开销相当大
    - 对已有关系建立聚簇，将导致关系中元组的物理存储位置移动，并使此关系上原有的索引无效，必须重建。
    - 当一个元组的聚簇码改变时，该元组的存储位置也要做相应改变。

聚簇存取方法的选择（续）

- 聚簇的适用范围
  - 1. 既适用于单个关系独立聚簇，也适用于多个关系组合聚簇

例：假设用户经常要按系别查询学生成绩单，这一查询涉及学生关系和选修关系的连接操作，即需要按学号连接这两个关系，为提高连接操作的效率，可以把具有相同学号值的学生元组和选修元组在物理上聚簇在一起。这就相当于把多个关系按“预连接”的形式存放，从而大大提高连接操作的效率。

聚簇存取方法的选择（续）

- 2. 当通过聚簇码进行访问或连接是该关系的主要应用，与聚簇码无关的其他访问很少或者是次要的时，可以使用聚簇。
  - 尤其当SQL语句中包含有与聚簇码有关的ORDER BY，GROUP BY，UNION，DISTINCT等子句或短语时，使用聚簇特别有利，可以省去对结果集的排序操作

聚簇存取方法的选择（续）

- 选择聚簇存取方法
  - 设计候选聚簇

（1）常在一起进行连接操作的关系可以建立组合聚簇

（2）如果一个关系的一组属性经常出现在相等比较条件中，则该单个关系可建立聚簇；

（3）如果一个关系的一个（或一组）属性上的值重复率很高，则此单个关系可建立聚簇。

聚簇存取方法的选择（续）

- 检查候选聚簇中的关系，取消其中不必要的关系
  - （1）从聚簇中删除经常进行全表扫描的关系
  - （2）从聚簇中删除更新操作远多于连接操作的关系
  - （3）从聚簇中删除重复出现的关系

当一个关系同时加入多个聚簇时，必须从这多个聚簇方案（包括不建立聚簇）中选择一个较优的，即在这个聚簇上运行各种事务的总代价最小。

7.5.3 确定数据库的存储结构

- 确定数据库物理结构主要指确定数据的存放位置和存储结构，包括：确定关系、索引、聚簇、日志、备份等的存储安排和存储结构，确定系统配置等。

- 确定数据的存放位置和存储结构要综合考虑存取时间、存储空间利用率和维护代价3个方面的因素。

确定数据库的存储结构（续）

- 影响数据存放位置和存储结构的因素
  - 硬件环境
  - 应用需求
    - 存取时间
    - 存储空间利用率
    - 维护代价

这三个方面常常是相互矛盾的

例：消除一切冗余数据虽能够节约存储空间和减少维护代价，但往往会导致检索代价的增加

必须进行权衡，选择一个折中方案

1. 确定数据的存放位置

- 基本原则
  - 根据应用情况将
    - 易变部分与稳定部分分开存放
    - 经常存取部分与存取频率较低部分分开存放
- [例]
  - 可以将比较大的表分别放在两个磁盘上，以加快存取速度，这在多用户环境下特别有效。
  - 可以将日志文件与数据库对象（表、索引等）放在不同的磁盘以改进系统的性能。

2. 确定系统配置

- 数据库管理系统一般都提供了一些存储分配参数
  - 同时使用数据库的用户数
  - 同时打开的数据库对象数
  - 内存分配参数
  - 缓冲区分配参数（使用的缓冲区长度、个数）
  - 存储分配参数
  - 物理块的大小
  - 物理块装填因子
  - 时间片大小
  - 数据库的大小
  - 锁的数目等

确定系统配置（续）

- 系统都为这些变量赋予了合理的缺省值。

  在进行物理设计时需要根据应用环境确定这些参数值，以使系统性能最优。

- 在物理设计时对系统配置变量的调整只是初步的，要根据系统实际运行情况做进一步的调整，以切实改进系统性能。

7.5.4 评价物理结构

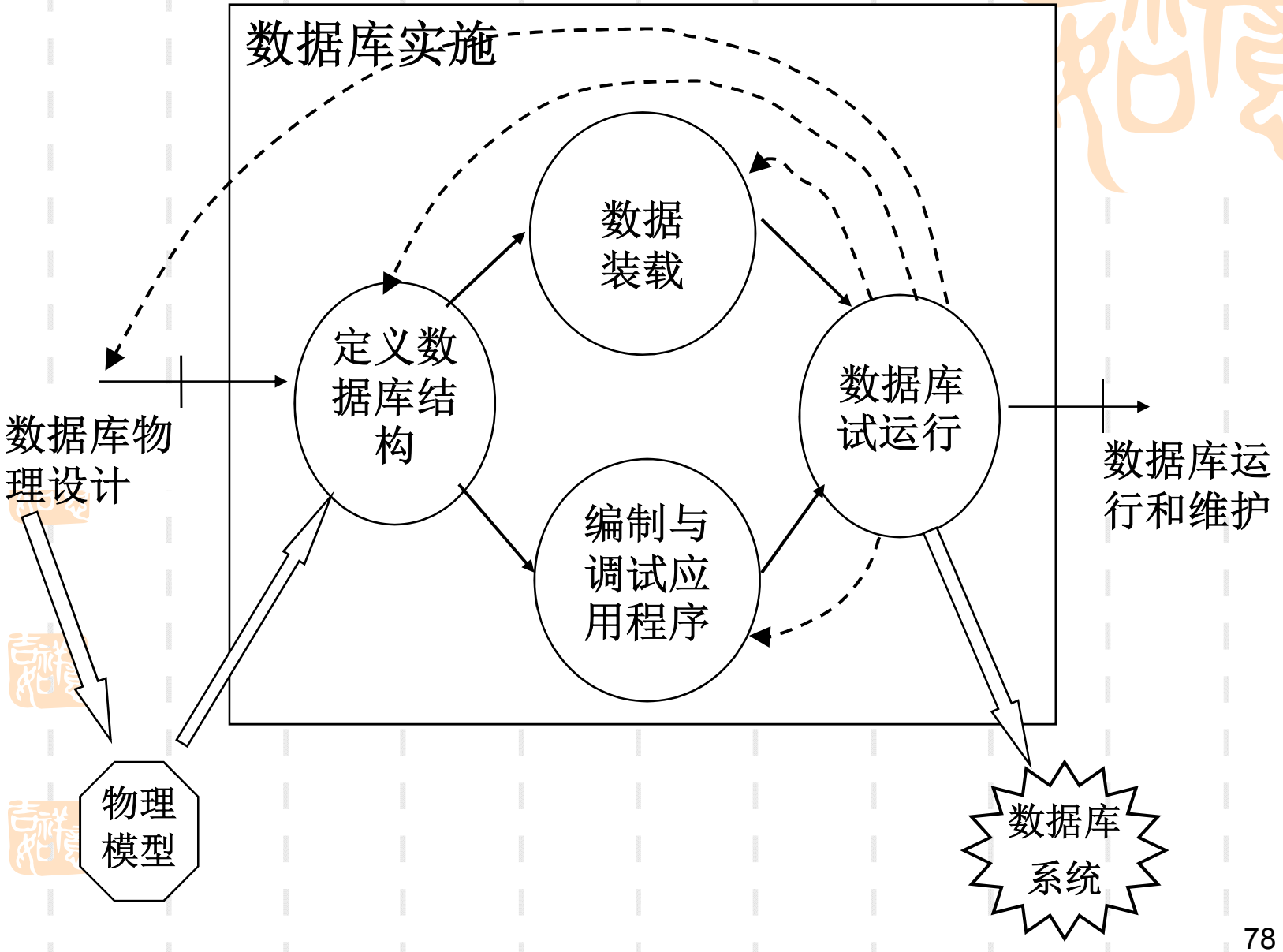
- 评价内容
  - 对数据库物理设计过程中产生的多种方案进行细致的评价，从中选择一个较优的方案作为数据库的物理结构

评价物理结构(续)

- 评价方法
  - 定量估算各种方案
    - 存储空间
    - 存取时间
    - 维护代价
  - 对估算结果进行权衡、比较，选择出一个较优的合理的物理结构
  - 如果该结构不符合用户需求，则需要修改设计

数据库的实施

- 数据库实施的工作内容
  - 用DDL定义数据库结构
  - 组织数据入库
  - 编制与调试应用程序
  - 数据库试运行


数据库实施
数据装载
定义数据库结构
数据库试运行
编制与调试应用程序
数据库物理设计
数据库运行和维护
物理模型
数据库系统

一、定义数据库结构

- 确定了数据库的逻辑结构与物理结构后，就可以用所选用的DBMS提供的数据定义语言（DDL）来严格描述数据库结构。

定义数据库结构（续）

例，可以用SQL语句定义如下表结构：

```
CREATE  TABLE  学生
    (学号  CHAR(8),
      ……
    );
CREATE  TABLE  课程
    (
      ……
    );
……
```

定义数据库结构（续）

接下来是在这些基本表上定义视图：

```
CREATE VIEW ....
  (
        ..............
  );
  ..............
```

如果需要使用簇集，在建基本表之前，应先用CREATE CLUSTER语句定义聚族。

二、数据装载

- 数据库结构建立好后，就可以向数据库中装载数据了。组织数据入库是数据库实施阶段最主要的工作。
- 步骤

1) **筛选数据**。需要装入数据库中的数据通常都分散在各个部门的数据文件或原始凭证中，所以首先必须把需要入库的数据筛选出来。

2) **转换数据格式**。筛选出来的需要入库的数据，其格式往往不符合数据库要求，还需要进行转换。这种转换有时可能很复杂。

3) **输入数据**。将转换好的数据输入计算机中。

4) **校验数据**。检查输入的数据是否有误。

三、编制与调试应用程序

- 数据库应用程序的设计应该与数据设计并行进行。

- 在数据库实施阶段，当数据库结构建立好后，就可以开始编制与调试数据库的应用程序。调试应用程序时由于数据入库尚未完成，可先使用模拟数据。

7.6.2 数据库的试运行

- 在原有系统的数据有一小部分已输入数据库后，就可以开始对数据库系统进行联合调试，称为数据库的试运行
- 数据库试运行主要工作包括：

1）**功能测试**

- 实际运行数据库应用程序，执行对数据库的各种操作，测试应用程序的功能是否满足设计要求
- 如果不满足，对应用程序部分则要修改、调整，直到达到设计要求

2）**性能测试**

- 测量系统的性能指标，分析是否达到设计目标
- 如果测试的结果与设计目标不符，则要返回物理设计阶段，重新调整物理结构，修改系统参数，某些情况下甚至要返回逻辑设计阶段，修改逻辑结构

数据库的试运行（续）

- 数据库性能指标的测量
  - 数据库物理设计阶段在评价数据库结构估算时间、空间指标时，作了许多简化和假设，忽略了许多次要因素，因此结果必然很粗糙。
  - 数据库试运行则是要实际测量系统的各种性能指标（不仅是时间、空间指标），如果结果不符合设计目标，则需要返回物理设计阶段，调整物理结构，修改参数；有时甚至需要返回逻辑设计阶段，调整逻辑结构。

数据库的试运行（续）

强调两点：

- 分期分批组织数据入库
  - 重新设计物理结构甚至逻辑结构，会导致数据重新入库。
  - 由于数据入库工作量实在太大，费时、费力，所以应分期分批地组织数据入库
  - 先输入小批量数据供调试用
    - 待试运行基本合格后再大批量输入数据
    - 逐步增加数据量，逐步完成运行评价

数据库的试运行（续）

- 数据库的转储和恢复
  - 在数据库试运行阶段，系统还不稳定，硬、软件故障随时都可能发生
  - 系统的操作人员对新系统还不熟悉，误操作也不可避免
  - 因此必须做好数据库的转储和恢复工作，尽量减少对数据库的破坏。

7.6.3 数据库的运行与维护

- 数据库试运行合格后，数据库即可投入正式运行。
- 数据库投入运行标志着开发任务的基本完成和维护工作的开始
- 对数据库设计进行评价、调整、修改等维护工作是一个长期的任务，也是设计工作的继续和提高。
  - 应用环境在不断变化
  - 数据库运行过程中物理存储会不断变化

数据库运行与维护（续）

- 在数据库运行阶段，对数据库经常性的维护工作主要是由DBA完成的，包括：

1.数据库的备份和恢复

- 转储和恢复是系统正式运行后最重要的维护工作之一。
- DBA要针对不同的应用要求制定不同的备份计划，定期对数据库和日志文件进行备份。
- 一旦发生介质故障，即利用数据库备份及日志文件备份，尽快将数据库恢复到某种一致性状态。

数据库运行与维护（续）

2.数据库的安全性、完整性控制

- DBA必须根据用户的实际需要授予不同的操作权限
- 在数据库运行过程中，由于应用环境的变化，对安全性的要求也会发生变化，DBA需要根据实际情况修改原有的安全性控制。
- 由于应用环境的变化，数据库的完整性约束条件也会变化，也需要DBA不断修正，以满足用户要求。

数据库运行与维护（续）

3.数据库性能的监督、分析和改进

- 在数据库运行过程中， DBA必须监督系统运行，对监测数据进行分析，找出改进系统性能的方法。
  - 利用监测工具获取系统运行过程中一系列性能参数的值
  - 通过仔细分析这些数据，判断当前系统是否处于最佳运行状态
  - 如果不是，则需要通过调整某些参数来进一步改进数据库性能

数据库运行与维护（续）

4.数据库的重组织和重构造

- 为什么要重组织数据库
  - 数据库运行一段时间后，由于记录的不断增、删、改，会使数据库的物理存储变坏，从而降低数据库存储空间的利用率和数据的存取效率，使数据库的性能下降。

数据库的运行与维护（续）

- 数据库的重组织和重构造
  - 重组织的形式
    - 全部重组织
    - 部分重组织
      - 只对频繁增、删的表进行重组织
  - 重组织的目标
    - 提高系统性能

数据库的运行与维护（续）

- 重组织的工作
  - 按原设计要求
    - 重新安排存储位置
    - 回收垃圾
    - 减少指针链
  - 数据库的重组织不会改变原设计的数据逻辑结构和物理结构

数据库运行与维护（续）

- 数据库重构造

  根据新环境调整数据库的模式和内模式

  - 增加新的数据项
  - 改变数据项的类型
  - 改变数据库的容量
  - 增加或删除索引
  - 修改完整性约束条件

7.7 小结

- 数据库的设计过程
  - 需求分析
  - 概念结构设计
  - 逻辑结构设计
  - 物理设计
  - 实施和维护

小结（续）

- 数据库各级模式的形成
  - 数据库的各级模式是在设计过程中逐步形成的
  - 需求分析阶段综合各个用户的应用需求（现实世界的需求）
  - 概念设计阶段形成独立于机器特点、独立于各个DBMS产品的概念模式（信息世界模型），用E-R图来描述

小结（续）

- 在逻辑设计阶段将E-R图转换成具体的数据库产品支持的数据模型如关系模型，形成数据库逻辑模式。然后根据用户处理的要求，安全性的考虑，在基本表的基础上再建立必要的视图（VIEW）形成数据的外模式
- 在物理设计阶段根据DBMS特点和处理的需要，进行物理存储安排，设计索引，形成数据库内模式

作业

- 1,7,10,11