《数据库设计》实验指导

编写人：李滨 许速

说明：本实验指导书主要为了学生更好地学习与掌握《数据库原理及应用》中数据库设计相关理论和方法。重点讲述一种数据库设计方法学，希望读者通过学习能够理解数据库设计方法学的基本框架，在实践中自觉遵循方法学中的各个步骤，以提高数据库设计的效率，并为整个应用系统构建一个健壮的数据库模型。

1 数据库设计概述

数据库应用系统在我们的日常生活中发挥着举足轻重的作用，如超市购物系统、图书馆管理系统、铁路售票系统、Internet 等等。尽管我们熟知并且几乎每天都要接触其中的某些应用，但是，在其背后隐藏的高深的技术对很多人来说可能还很陌生，这种技术的核心就是数据库本身。对于某一应用系统来说，为了最大限度地支持其最终用户，有必要对数据库进行结构化处理，这个过程就是所谓的数据库设计。

在实践中，很多数据库应用系统开发者并没有给予数据库设计足够的重视，主要原因是：他们太迷信数据库管理系统（Database Management System, DBMS），认为购入一个功能强大的 DBMS 后数据库设计就不困难、不重要了。一些国内外的数据库教材常常是在为 DBMS 的开发厂商做宣传，而很少站在数据库用户角度，从数据库应用系统出发介绍数据库设计方法。结果往往使读者搞不清书中介绍的是数据库管理程序的设计思想，还是应用这种 DBMS 进行数据库设计的思想。其实，DBMS 只是给用户为已采用的数据库提供一个舞台，而是否使用这个舞台上的道具以及唱什么戏，则完全取决于用户的戏剧脚本和导演(开发者)的安排。

由于数据库工程所必需的时间和资源往往被低估，使得所开发的数据库并不充分、效率低、文档有限、维护困难。实际上，一个系统能否被用户接受，数据库设计的质量是至关重要的。一个设计不良的数据库会产生很多错误，这些错误可能导致错误的决策，会对企业造成巨大的损失，而一个设计良好的数据库可以为企业领导提供很好的决策支持。

目前，关系数据库管理系统（Relational Database Management System, RDBMS）是市场上的主流，它代表了第二代数据库管理系统，它基于 E.F.Codd 博士 1970 年的“A Relational Model of Data for Large Shared Data Banks”（大型共享数据银行的关系数据模型）论文中所提出的关系数据模型。本实验指导书要讨论的正是关系数据库设计。

数据库应用系统的开发生命周期如图 1 所示[1]，大致包括数据库规划、系统定义、需求的收集与分析、设计（包括数据库设计和应用程序设计）、构建原型、实现、数据转换与加载、测试、操作性维护。当然，我们感兴趣的是数据库设计这个步骤。读者可参阅相关资料来获取对其他各个阶段的详细信息。

从图 1 中我们可以得出下列结论：

1） 在数据库系统开发生命周期中，数据库和应用程序设计是平行的活动，并且基于相同的用户需求分析结果。一般情况下，在数据库本身的设计完成之前，我们不能完成应用程序的设计；另一方面，数据库本身就是支持应用程序的。因而，在二者之间一定有信息流存在。

2） 这不是瀑布模型，各个步骤也不是严格按顺序进行的，而是存在一定的反馈和迭代。

3） 在设计过程的不同阶段，我们可以选择是完全实现数据库系统的设计还是构建一个原型。

2 数据库设计方法学

2.1 数据库设计方法

一般数据库设计方法有两种，即属性主导型和实体主导型。属性主导型从归纳数据库应

用的属性出发，在归并属性集合(实体)时维持属性间的函数依赖关系，属于自下而上的数据库设计方法。实体主导型则先从寻找对数据库应用有意义的实体入手，然后通过定义属性来定义实体，属于自上而下的方法。一般现实世界的实体数在属性数 1/10 以下时，宜使用实体主导型设计方法。本书讲述的是面向对象的软件工程，而数据库设计是软件工程的一个环节，因此我们采用实体主导型数据库设计方法来展开本指导书的其他部分的讨论。

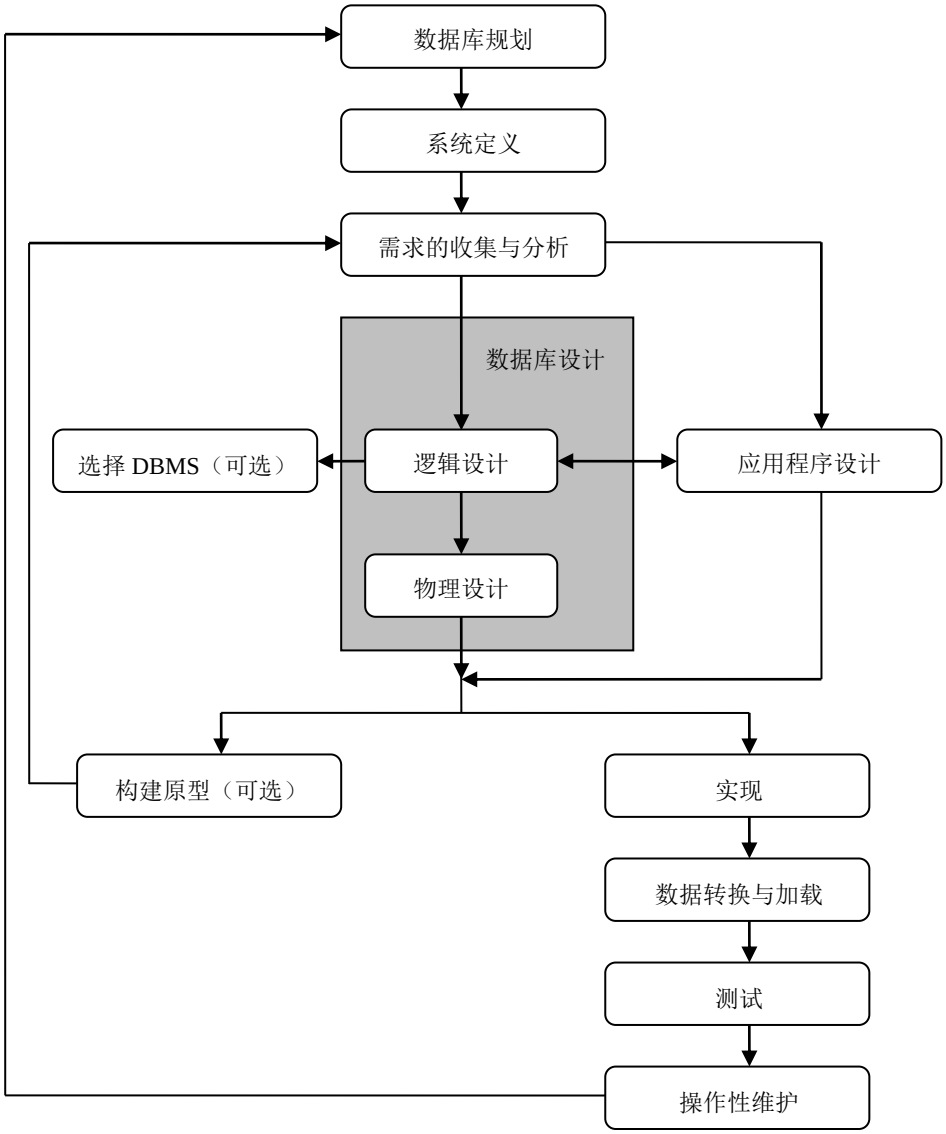


图 1 数据库应用系统生命周期的主要步骤

2.2 数据库设计方法学

当面临的应用系统所需要的数据库非常复杂时，为了使数据库既能满足用户需求又能满足性能需求（例如响应时间），就需要一种系统化的方法来设计和构建数据库。这种系统化的方法就是数据库设计方法学。

数据库设计方法学由一系列步骤组成，这些步骤在工程的每个阶段引导设计者使用合适的技术，这些阶段还帮助设计者规划、管理、控制和评价数据库开发过程。此外，该方法是一个结构化的方法，用于以标准化的和有组织的方式分析和建立数据库需求模型。

本实验指导书采用一种目前比较成熟的数据库设计方法[1]，把数据库设计分成两个主要阶段：逻辑数据库设计和物理数据库设计。某些文献中认为在逻辑数据库设计阶段之前还应该有概念数据库设计，它完全不考虑所使用的数据模型（例如，关系数据模型或对象数据模

型等）和其他物理实现细节。正如本书 1 节所说，我们在这里只考虑关系数据库的设计，所以可以将概念数据库设计合并到逻辑数据库设计阶段中。

各个阶段及其步骤如下：

逻辑数据库设计

步骤 1：创建并检查 ER 模型

- 1.1 标识实体
- 1.2 标识关系
- 1.3 标识实体或关系的相关属性
- 1.4 确定属性域
- 1.5 确定候选键、主键和备用键属性
- 1.6 特化/泛化实体（可选步骤）
- 1.7 检查模型的数据冗余
- 1.8 检查模型是否支持用户事务
- 1.9 与用户一起检查模型

步骤 2：将 ER 模型映射为表

- 2.1 创建表
- 2.2 用规范化方法检查表结构
- 2.3 检查表是否支持用户事务
- 2.4 检查业务规则
- 2.5 与用户讨论逻辑数据库设计

物理数据库设计

步骤 3：为目标 DBMS 转换全局逻辑数据模型

- 3.1 设计基本表
- 3.2 设计派生数据的表示
- 3.3 设计其他业务规则

步骤 4：选择文件组织方式和索引

- 4.1 分析事务
- 4.2 选择文件组织方式
- 4.3 选择索引

步骤 5：设计用户视图

步骤 6：设计安全性机制

步骤 7：引入受控冗余的考虑

步骤 8：监视并调整操作系统

图 2 数据库设计方法学中的各阶段

说明：数据库设计是一个迭代过程，开始以后就要不要不断精化。尽管该数据库设计方法学是一个过程化的过程，并不表示一定要以过程化的方式执行，而把反馈和迭代考虑进来是值得推荐的。这种方法只是作为开发人员进行高效数据库设计的指导框架。

在以下各部分，我们将分别介绍上述各步骤。

3 创建并检查 ER 模型

一旦完成了需求收集和分析阶段的工作，并且把数据库需求整理成了文档，就可以开始进行数据库设计了，如图 1 所示。为了设计人员、程序员和最终用户之间能够高效地交流，需要一个和技术实现无关且无二义性的通信模型，ER（Entity-Relationship，实体-关系）模型就是一种常常被用来表示数据库逻辑模型的图形化表示。

目前，ER 模型没有标准的表示法，比较常见的两种传统的表示方法分别是：

- Chen 氏表示方法，它由代表实体的矩形和代表关系的菱形以及连接矩形和菱形的连线组成。
- Crow 的 Foot（脚注）表示法，它由代表实体的矩形以及代表实体间关系的连线组成。连线一端的脚注代表一对多的关系。

其中，第一种方法是很多教材（包括本课程教材）采用的方法，但是，这两种方法在应用时比较麻烦而且难以解释。在本指导书中，我们采用 UML 的类图表示法，以便扩展学生知识面。这里假定，读者有一定的 UML 基础。

3.1 标识实体

1．实体及其图形化表示

定义用户感兴趣的主要对象或有必要存在的对象，这些对象就是模型中的实体。主要方法就是仔细研究用户的需求说明，从中找到提到的名词或名词短语（例如，教师、学生、课程等）。在模型图中，每个实体用矩形表示，在矩形框中写上实体的名字（通常是名词或名词短语），如图 3 所示。

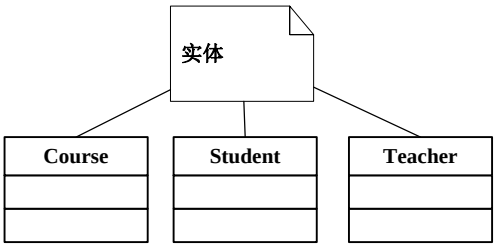


图 3 实体的图形化表示

某一个对象是否是实体、关系或属性并不是很容易区分的，并且主观性比较大，不同的设计者可能会做出不同地处理。在某种程度上来说，完成这些工作依赖于设计者的判断力和经验，数据库设计者必须根据现实世界进行选择，并根据实际情况对事物进行分类。因此，从用户需求说明提取的实体可能不是唯一的，但是必须要做到的一点是，通过多次迭代分析过程来获取完成系统需求所需要的足够多的实体。

3.2 标识关系

1．关系及其图形化表示

在标识完实体之后，接下来就是要找出这些实体间的所有关系。类似地，我们也可以借助于用户需求来完成，关系一般由动词或动词短语表示。例如：

- 系 *拥有* 教工
- 学生 *选修* 课程
- 教师 *管理* 学生（任班主任）

一个关系是特定实体之间对一个关联。每个关系也有一个名字，它描述关系的功能。在模型图中，每个关系用连接关联实体的一条线表示，并且用关系的名字标记。在模型中，关系名应该尽可能唯一。一个关系仅在一个方向标记，表示关系仅在一个方向上起作用，一旦选择了关系名，就要在名字旁边加上箭头指示正确的方向。如图 4 所示。

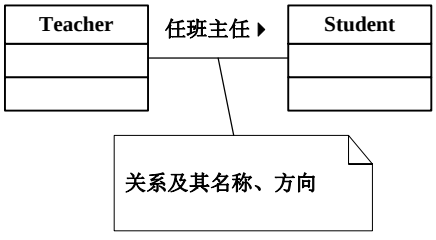


图 4 关系及其名称、方向的图形化表示

在确定了关系之后，接下来就要考虑关系的多样性约束。多样性（Multiplicity）是指一个实体中可能和相关实体的一个存在关联的实体事件的数目。多样性约束代表了用户或者企业建立的策略，被称为业务规则，这些规则确保所有合适的事务规则都被标识并且被表达。

在所有关系中，最常见的关系是度为 2 的二元关系，二元关系上的多样性约束一般包括一对一（1：1）、一对多（1：*）或多对多（*：*）。其次，就是递归关系，其多样性约束包括一对一（1：1）和一对多（1：*）两种情况。在 UML 图中，“一”被表示为“0..1”或“1..1”，“多”被表示为“1..*”或者有确定上下限的范围值（如“3..7”），分别标注在关系的受约束实体的一端。如图 5 所示。

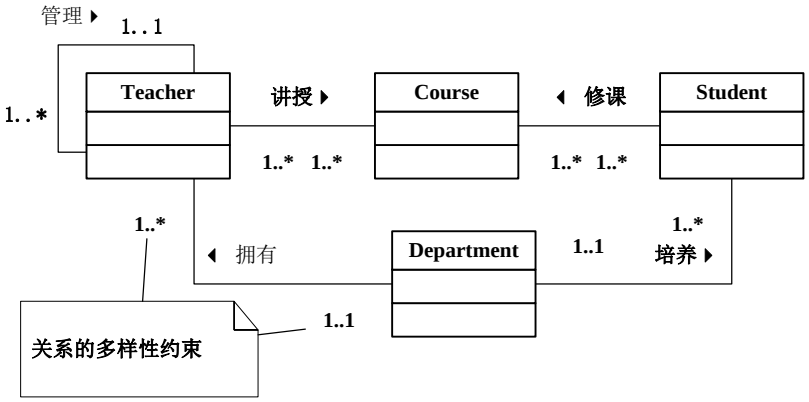


图 5 关系多样性约束的图形化表示

注意：在标识了关系及其多样性约束之后，还应该检查每个关系所描述的是否确实是所需要的，同时，还要检查是否产生了设计陷阱（主要包括扇形陷阱和深坑陷阱）。有关设计陷阱的深入讨论请参阅参考文献[1]。

3.3 标识实体或关系的相关属性

1．属性的类型

接下来的主要任务就是标识数据库中的实体和关系的属性。与标识实体类似，需要在用户需求说明中寻找名词或名词短语，它们是特性、标志或前面定义的实体或关系的特征。常见的属性类型有：

- 简单/复合属性。

根据属性是由单个还是多个元素构成的，可分为简单属性和复合属性。复合元素有简单属性构成。例如，学生籍贯可以是“XX 省 XX 市”（简单属性），或者是由“XX 省”和“XX 市”分开描述（复合属性）。具体的选择方案是由用户需求或业务规则决定的。例如，如果用户需要按省份和城市来统计学生情况，就应该把籍贯属性描述为由单值属性构成的复合属性。

- 单值/多值属性

根据属性的取值是否唯一，可分为单值属性和多值属性。例如，一个电影的类别属性可能是多个，如喜剧、动作、暴力等，而其出品时间就是单值属性。

- 派生属性

派生属性是指属性值可以由相关属性或属性集派生出来。例如，年龄属性可以从生日属性中派生而来。派生属性在相同实体中不是必须的，是否保存在数据库中要根据用户需求或数据库性能调整的需要而定（详见本指导书第 9 小节）。

2．属性的图形化表示

如果属性要显示在 E-R 图中，则把属性放在表示实体的矩形的第二部分（第一部分显

示实体名），如图 6 所示。

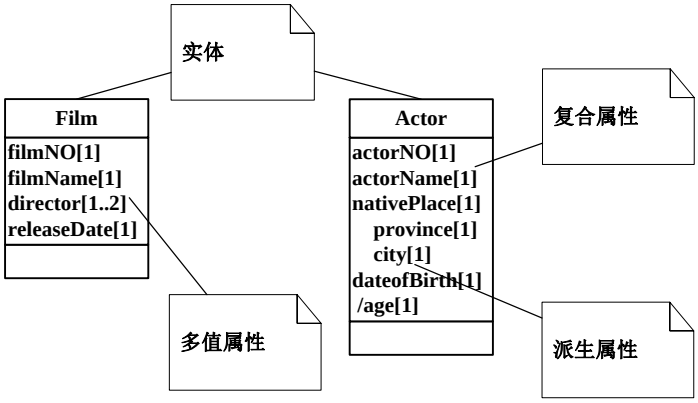


图 6 属性的图形化表示

3.4 确定属性域

域是一组值的集合，一个或多个属性可以从中选择它们的值。例如图书馆系统的属性域包括：

- 合法的读者编号属性是 zt20050001
- 其他

3.5 确定候选键、主键和备用键属性

这步确定实体的候选键，如果有多个候选键，则从中选择一个作为主键，其他的作为备用键。从候选键中选择主键时可以考虑如下建议：

- 具有最少的一组属性的候选建；
- 值很少变化的候选键；
- 在未来不会失去唯一性的候选键；
- 字符最少的候选键（对文本属性来说）；
- 最大值最小的候选键（对数值属性来说）；
- 从用户观点来看易于使用的候选键。

注意，如果可以为一个实体赋予一个主键，则这个实体被看成强实体，反之，如果不能为一个实体标识主键，则这个实体就是弱实体。

3.6 特化/泛化实体（可选步骤）

对于较复杂的数据库应用来说，传统的 ER 模型就显得过于有限，因此，需要在此基础上附加一定的语义，这就是增强的实体-关系模型（Enhanced Entity-Relationship, EER）。与 EER 模型有关的最有用的概念就是特化/泛化，其实质就是在原有模型基础上抽象出超类和子类实体。此步骤是可选的，有兴趣的读者可以参见参考文献[1]。

3.7 检查模型的数据冗余

这个步骤的主要目标是检查模型中是否有冗余存在，并删除存在的冗余。在这一步中有三个活动：

1．重新检查一对一关系

在前面标识实体时，可能标识了两个实体，但它们表示同一个对象，只是因为在不同的使用场合称呼不一样，那么就应该将这样的实体合并。如果主键不同，那么选择其中一个作为主键，其他的作为备用键。

2．删除冗余关系

在模型中如果一个关系可以通过其他关系获取相同的信息，那么它就是冗余关系。在建模过程中，冗余关系是没有必要存在的，因此，应该删除这些关系，以获取最小化的模型。在实际应用模型中，冗余关系往往不是显而易见的，因为存在于两个实体间的不同路径可能

代表着不同的关系。

3.8 检查模型是否支持用户事务

至此，已经得到了一个描述公司数据需求的 ER 模型，这个步骤的目标就是通过检查 ER 模型这个 ER 模型能否支持所需要的事务。使用 ER 模型和数据字典，尝试手工完成操作。如果可以以这个方式解决所有事务，就完成了 ER 模型是否支持用户事务的检查。否则，数据模型一定会有问题存在，而且这个问题是必须要解决的。这种情况下，很可能是数据模型这丢失了一个实体、关系或者属性。

通常，可以采用下面两种方法来确保 ER 模型支持所需的事务。

1．描述事务

2．使用事务路径

注意：这一步非常麻烦，但是最好不要为了省事略去该步骤，因为，实践证明，在开发的后期发现问题解决起来要付出更大的代价。

3.9 与用户一起检查模型

在步骤 1 的最后，应该与用户一起检查 ER 模型，以确保此模型是“真实”的表达了企业的需求。并且，如果发现了模型中的异常，必须做出相应的修改，可能还要重复以前的一些步骤，直到得到用户的肯定为止。

4 将 ER 模型映射为表

这个步骤的主要目标就是为步骤 1 中建立的 ER 模型产生表的描述。这组表应该代表逻辑数据模型中的实体、关系、属性和约束。然后检查每个表的结构，确保建表过程中没有产生错误。如果表中有错误，则表明在建表过程中或在 ER 模型中仍有没发现的错误。

4.1 创建表

在此步骤中，为 ER 模型创建表来表达实体、关系、属性和约束。

1．如何表达实体

对 ER 模型中的每个实体，创建一个包括实体的所有简单属性的表。对于复合属性，仅包含表中组成复合属性的简单属性。例如，对于复合属性“籍贯（nativePlace）”，应该包括它的简单属性省份(province)和城市(city)。如果有可能，标识出每个表中组成主键的列。

2．如何表达关系

一个实体与另一个实体间的关系由主键/外键机制表达。为了决定将外键属性放在哪里，首先必须标识关系中包含的“父”实体和“子”实体。父实体指的是把自己的主键拷贝到代表子实体的表中作为外键的实体。接下来，我们具体需要考虑下列几种情况下的父/子实体的标识。

（1）一对多的二元关系

对于 1：*二元关系，关系“一”端的实体被设计为父实体，“多”端的实体被设计为子实体。为了描述这种关系，父实体主键的拷贝，被放置在子实体的表中，作为外键。

说明：在 1：*关系有一个或多个属性的情况下，这些属性也应该随着主键加到子表中。

（2）一对多的递归关系

对于 1：*递归关系，其表示方法与上面情况类似，不同的是，需要将实体中的主键列多拷贝一份，并重新命名，作为外键。

（3）一对一的二元关系

在表示 1：1 关系时会复杂一些，因为不能利用元组的数目来标识一个关系中的父实体和子实体，而是需要使用参与（Participation）过程来决定是把实体结合为一个表合适，还是建两个表由外键来表示关系好。

注：参与决定是否所有的或只有部分实体在关系中发生参与，前者称为是强制参与，后者被称为可选参与。

需要考虑下列几种情况：

- 1：1 关系的两边都是强制参与

这种情况下，应该将两个实体合并为一个表，并选择初试实体中的一个主键作为新表的主键，其他的主键作为备用键。

- 1：1 关系的一边是强制参与

这种情况下，关系中的可选参与的实体被设计为父实体，关系中的强制参与的实体被设计为子实体。如果关系中有一个或多个属性，这些属性也应该被加到子表中。

- 1：1 关系的两边都是可选参与

这种情况下，父实体和子实体之间的设计限制较小，可根据具体情况具体处理。

（4）一对一的递归关系

这种情况与（3）描述的类似，不同的是，这里的 1：1 关系两边的实体是相同的。

（5）多对多的二元关系

对于每个*：*二元关系，首先创建一个包含关系的任何属性（如果有）的表，并将参与关系的实体的主键拷贝到新表中作为外键。然后，将其中一个外键或全部外键（可能还有关系的一些属性）作为主键。

（6）多值属性

对于每个与实体有关的多值属性，应该遵守上述 1：*关系中所描述的规则，同时，创建一个表达多值属性的表，并将父实体的主键属性复制到该表中作为外键。

4.2 用规范化方法检查表结构

这一步骤的主要目的是通过检查每个表中的列的组成，用规范化方法来重构这些表，以便降低不必要的数据冗余，避免数据库的更新异常。通常，使所建的每个表至少满足第三范式（3NF）（有关范式的介绍见参考文献[2]）。

注：更新异常指的是由于数据表列的重复而导致数据库在执行插入、更改和删除操作时引起的数据不一致的现象，相应地，更新异常可以分为插入异常、更改异常和删除异常。

4.3 检查表是否支持用户事务

这一步骤的目标是检查 4.1 所建的表是否与用户需求说明中所要求的一致，即是否支持用户所需事务。一种方法是检查是否支持事务的数据需求，以确保数据在一个或多个表中存在，同时，如果事务所需求的数据在多个表中，则应该检查这些表是否能够通过主键/外键机制连接起来。

说明：虽然在 3.8 小节中已经进行过这样的检查，但那是确保局部逻辑数据模型支持所需求的事务，而本步骤是检查根据 ER 模型所建的表是否也支持用户事务，以此确保 ER 模型映射到表的过程中没有发生错误。

4.4 检查业务规则

这一步骤的目标是检查逻辑数据库设计中表达的业务规则，即用于防止数据库不完整、不准确或不一致的约束。在这个阶段，只需要关系高级设计，即确定需要什么样的数据库完整性约束，而不管实现方式。

应该考虑的完整性约束有下列类型：

1．需要的数据

某些列必须要包含数据，即不允许有空值。这在数据字典中应该标记出来。

2．列的值约束

每个列都有一个值域（即一组合法的值）。如图书类型、员工职务、年龄、联系电话等在取值时都是有一定限制的。

3．实体完整性

实体的主键不能为空，并且必须是唯一的。

4．多样性

多样性表达了数据库中数据间的关系的约束。例如，图书馆的每个部门必须有员工。

5．参照完整性

父/子表连接是通过外键关联起来的，因此，子表中的外键值在父表中必须存在（是否允许为空值要根据用户需求而定），否则就违反了参照完整性约束。

6．其他业务规则

主要指企业的商业规则，如每个借阅证最多允许的图书数量、最长借阅时间。

4.5 与用户讨论逻辑数据库设计

至此，逻辑数据库设计即将完成，在结束本步骤之前，与用户一起研究设计结果还是值得的。

如果设计只有一个用户视图的数据库，或已经使用集中化方法合并了多个用户视图，那么就可以开始物理数据库的设计了。

5 为目标 DBMS 转换全局逻辑数据模型

逻辑数据库结构（即实体、属性、关系和约束）必须最终转换为目标 DBMS 可以实现的物理数据库设计。逻辑数据库设计最大限度地独立于实现细节，例如目标 DBMS 的具体功能、应用程序、编程语言或任何其他的物理考虑。因此，对数据库中的某些部分在特定目标 DBMS 中可能会有不止一种实现方法。为了选择合适的实现方案，在进行物理数据库设计时有必要对目标 DBMS 的功能有个较为全面的认识，并且提供多种不同的可选实现方案。

逻辑数据库设计关心的是“什么”，而物理数据库设计关系的是“怎么”。但是，物理数据库设计并不是独立的行为，首先它需要以逻辑数据库设计的逻辑模型作为输入，其次，在逻辑、物理和应用程序设计之间经常会有反复的。例如，在物理数据库设计期间为了系统性能优化而进行的决策，如合并表，可能会影响逻辑数据模型（如图 1 所示）。

本节的主要内容是介绍怎样从逻辑数据模型产生基本的工作关系数据库。

5.1 设计基本表

这个步骤的目标是确定如何在目标 DBMS 中描述在逻辑数据模型标识的基本表，这些表的必要的信息可以从数据字典中获得，包括：

- 表名；
- 列名；
- 主键以及备用键和外键；
- 任何标识出的参照完整性约束。

对于每个列，应该有如下定义：

- 域，包括数据类型、长度和域上的约束；
- 可选的默认值；
- 是否可以为空；
- 是否是派生列，如果是，怎么计算。

1．在 MS SQL Server 2000 中实现基本表

SQL Server 2000 提供了两种创建空表的方法：

- 使用 Enterprise Manager（企业管理器）进行可视化的操作，方便易用且功能强大，适合初学者使用。
- 使用 SQL Query Analyzer（查询分析器）工具，通过编写和执行 SQL 脚本来完成基本表的创建工作，适合有一定的 SQL 编程和调试经验的开发者使用。

不论采用哪知方法，完成的功能都是一样的，即指定表名、列名及其数据类型（包括长度、精度）、列是否允许为空、是否为标识列、列的默认值、各种约束等等。需要指出的是，每个列的域可以通过创建新的自定义数据类型来指定，同时可以附加事先创建的规则。

2．在 MS SQL Server 2000 中创建两个表之间的关系

要创建两个表之间的关系，前提是创建子表时父表已经存在，或者先创建两个基本表，然后指定二者之间的主键/外键关系。这里也有两种方法，即分别使用企业管理器和查询分析器来完成。限于篇幅，不再赘述。

5.2 设计派生数据的表示

本步骤目标是设计派生数据在数据库中的表示。派生列作为文档出现在数据字典中，但不一定要出现在 ER 模型中，如果在 ER 模型中显示，则在其名字前边用一个“/”标记。

从物理数据库设计的角度看，派生列是存储在数据库中还是每当需要时再进行计算，没有统一的规定，需要根据实际情况做出权衡，应该考虑下面几个因素：

- 存储派生数据的代价以及维护它与派生它的数据的一致性的代价；
- 每次计算它需要的代价；
- 针对派生数据查询操作的执行频率、对响应时间的要求；
- 针对派生它的列的更新操作的执行频率、对响应时间的要求。

例如，如果涉及派生列的查询比较频繁或/和派生列计算复杂但对响应时间要求比较严格，可考虑把派生列存储在数据库中，因为扩充存储设备所需要的成本还是很容易承受的。如果相对派生列的查处而言，涉及派生它的列的更新比较频繁/和对响应时间要求比较严格，那么最好不要存储派生列。

5.3 设计其他业务规则

对表的更改操作可能会受到其所表达的现实世界的事务业务规则的约束。有关域约束以及关系完整性约束前面已经叙述过，这里主要关注的是与用户商业逻辑相关的其他业务规则。如图书馆系统中规定，每个学生用户最多允许借阅的书籍数量是 5 本。对于这样的规则，可以在创建表时使用 CHECK 约束或者使用触发器（Trigger）来实现。如果 DBMS 不支持某些或全部的业务规则，就有必要在应用程序中设计相应的规则。甚至，在实际应用系统设计时，为了开发和维护的方便，专门把业务规则抽取出来形成业务逻辑层，如典型的三层体系结构。

注意:触发器的功能通常可以用其他方式实现,并且在调试程序时触发器可能成为干扰。因此，设计中应该尽量避免使用触发器，假如确实需要采用触发器，最好集中对它文档化。

6 选择文件组织方式和索引

与逻辑设计相同，物理设计也必须遵循数据的特性以及用途，尤其是，必须理解数据库要支持的典型工作量。在分析阶段，用户可能提出了这样的需求，如要求某些事务必须要运行多快，或者每秒必须要处理多少个事务。本节的目的就是确定最佳文件组织方式来存储基本表并且实现所要求性能的索引，而上述用户需求信息将是讨论的基础。

可以获得什么样的文件组织方式依赖于目标 DBMS，因此，有必要了解可用的结构，并且知道目标系统怎样使用这些结构的。同时，还必须了解要支持的事务的细节，以便做出有意义的物理设计选择。

6.1 分析事务

要进行有效的物理数据库设计，很好地理解运行在数据库中的事务是非常有必要的。需要说明的是，要分析所有的事务在许多情况下是不可取的，但是，至少应该分析其中最重要的事务，建议使用 80-20 规则（用户查询的最活跃的 20%占总的数据访问的 80%）。

为了辅助分析，可以使用事务/表交叉引用矩阵，它显示了每个事务所访问的表。一般的处理方法是：

1）将所有事务路径映射到表中；

2）确定哪些表最常被事务访问；

3）分析选出的包含了这些表的事务。

6.2 选择文件组织方式

为数据选择有效的存储方式也是物理数据库设计的目标之一，因此，如果目标 DBMS 允许，就为每个表选择最佳的文件组织方式，如堆、哈希、索引顺序访问方法（ISAM）和 B+树等。但是，尽管某些关系 DBMS 允许创建索引，却不允许选择文件组织方式，如 SQL Server 和 Access 等。

有关选择文件组织方式的技巧，详见参考文献[1]，这里不再赘述。

6.3 选择索引

索引是从数据库中获取数据的最高效方式之一，95%的数据库性能问题都可以采用索引技术得到解决。虽然为数据表添加合适的索引能有效提高数据库访问性能，但是，也应该避免索引泛滥，因为不合适的索引不仅不能提高查询效率，还会增加维护索引的开销，并且降低数据更新的效率。

SQL Server 中，索引分为聚集索引（Clustered Index）和非聚集索引（Unclustered Index），二者的存储结构都是 B 树，但不同的是前者使数据行按索引键排序，而后者不是。

1．索引设计原则

SQL Server 对最终用户和 T-SQL 开发人员通常是透明的。查询中通常不必指定索引，除非用户强制要求优化器使用特定索引（不推荐使用）。一般情况下，当 MS SQL Server 执行查询时，基于成本的查询优化器会对可用的数据检索方法（如顺序扫描、索引扫描或索引查询）的成本进行评估，从中选用 I/O 成本最低的方法。因此，为了能使关键的查询采用最优的查询计划，在考虑是否为一个列创建索引时，应考虑被索引的列是否以及如何用于查询中，并掌握索引设计的常规准则。

1） 索引设计的一般准则

- 对复合索引，应把选择性较高的列放在索引左边

  为了保证索引的使用率最大化，第一排序的列应该为查询中最常用的列。

- 保证在连接中使用索引列

  如果在连接中指定的列没有索引，则连接处理的效率很低。注意，主键/外键连接是最常见的情况，但是外键约束（Foreign Key Constraints）并不自动对列生成索引，而主键约束可以自动生成索引。因此，如果确实需要在查询中使用主键/外键连接，则必须对外键列添加索引。

- 对最关键的查询和事务调整索引

  不能对表中运行的每个查询生成索引。因此，标识出最关键的、最常执行的查询，并对其设计索引，对改善查询性能是很帮助的。SQL Profiler 是个很好的工具，它能够确定最常用的查询，也可以确定运行较慢的查询，通过设计索引来改进。

- 避免对选择性差的列生成索引

  因为优化器通常用不到这样的索引，相反，它会占用空间，并且在插入、删除和更新数据时增加不必要的开销。索引涵盖是个例外，了解更多索引涵盖相关知识可查阅参考文献[3]。

- 小心选择聚集索引和非聚集索引

2）聚集索引选择指南

在执行查询时，通过聚集索引比通过非聚集索引有很高的执行效率，但是，一个表只能生成一个聚集索引，因此，在设计聚集索引时要认真考虑。默认情况下，表中的主键定义为唯一聚集（Unique Clustered）索引，但这不一定总是最佳选择。相反，对主键生成唯一非聚集（Unique Unclustered）索引并选择其他索引作为聚集索引倒是一个比较好的选择。在选择聚集索引的候选索引时可以考虑下列准则：

- 有几个重复值要经常查找的列

例如：WHERE department = ‘computer’。由于数据值是物理排序的，因此所有重复值放在一起，针对该索引的任何查询都可以用最小量的 I/O 取得所有值。

- 通常在数值范围中查询的列

例如：WHERE orderDate BETWEEN ‘2005-12-1’ and ‘2005-12-31’ 或 WHERE orderDate >=‘2005-12-1’ and orderDate <= ‘2005-12-31’。由于表中的数据已经按顺序排列，因此，可以通过聚集索引找到符合查询范围的第一个匹配行，然后可以按顺序扫描，直到最后一个匹配行。当这个范围很大时，使用聚集索引扫描比通过非聚集索引的书签查找能够大大减少总逻辑 I/O 数。

- 经常在 order by 子句中指定的列

由于数据已经排序，当按索引键顺序读取数据时，SQL Server 不需要重新排序数据。唯一的例外是并行查询操作，了解更多并行查询策略相关知识可查阅参考文献[3]。

- 连接子句中经常使用的非主键列

聚集索引通常比非聚集索引小，每个查找所需的页 I/O 数通常比非聚集索引少，这在连接多个记录时非常有好处。比如，对于单行的读取，多一两个页读取无关紧要，但如果在 100 000 个连接迭代时增加这些页读取则要增加 100 000 到 200 000 次页读取。

3）其它

- 对小型表进行索引可能不会产生优化效果，因为 SQL Server 在遍历索引以搜索数据时，花费的时间可能会比简单的表扫描还长。
- 不要索引大型字段（有很多字符），这样作会让索引占用太多的存储空间。
- 应使用 SQL 事件探查器和索引优化向导帮助分析查询，确定要创建的索引。
- SQL Server 2000 允许在视图上定义聚集索引，生成索引视图。在索引视图上也可以非聚集索引，同一般索引一样，定义合适的索引能够提高查询性能，不恰当的索引也会适得其反。比如可以对基础表数据变化不大的视图生成索引。
- SQL Server 2000 中可以对表中的计算列建立聚集索引或非聚集索引，具体内容见参考文献[3]。

为数据库及其工作负荷选择正确的索引是非常复杂的，需要在查询速度和更新成本之间取得平衡。窄索引（搜索关键字中只有很少的列的索引）需要的磁盘空间和维护开销都更少。而另一方面，宽索引可以覆盖更多的查询。确定正确的索引集没有简便的规则。经验丰富的数据库管理员常常能够设计出很好的索引集，但是，即使对于不特别复杂的数据库和工作负荷来说，这项任务也十分复杂、费时和易于出错。

7 设计用户视图和安全性机制

数据库是一种公共资源，在尽可能满足用户之外，还有保证其安全性。本节的主要内容是研究如何设计在需求分析和收集阶段标识的用户视图以及安全性机制。同物理数据库设计的其他步骤一样，本节的内容也与目标 DBMS 有很大关系。

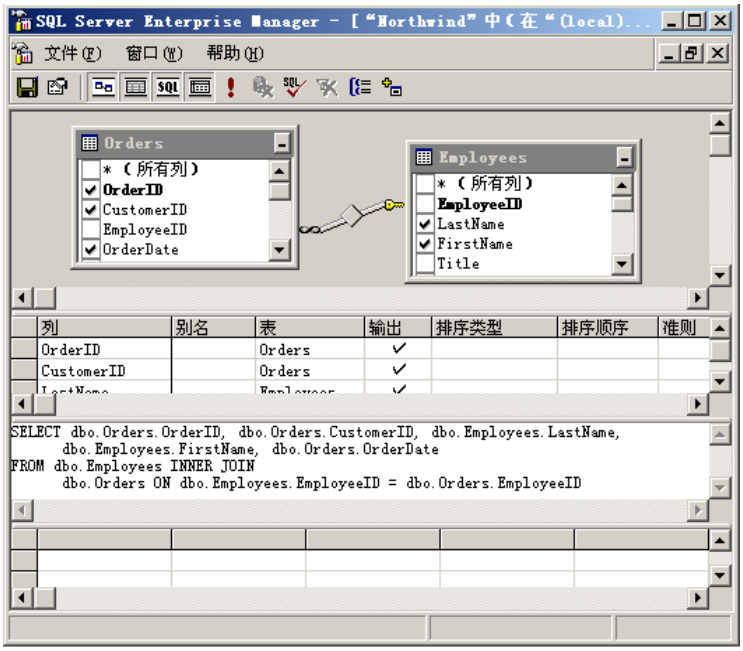
7.1 设计用户视图

回顾 3 节，数据库设计方法学是从根据用户视图构建 ER 模型开始的。

现在我们就开始设计前面标识出的所有用户视图。在 SQL Server 2000 中，创建视图有两种途径：

- 企业管理器
- Transaction-SQL

1．在企业管理器中创建视图



2．语法

CREATE VIEW [< database_name > .] [< owner > .] view_name [(column [,...n])]
[WITH < view_attribute > [,...n]]
AS
select_statement
[WITH CHECK OPTION]
< view_attribute > ::=
{ ENCRYPTION | SCHEMABINDING | VIEW_METADATA }

7.2 设计安全性机制

保证数据库的安全对企业来说是至关重要的，有关的安全性需求可能在需求收集和分析阶段整理清楚，本步骤的目的就是决定如何利用目标 DBMS 提供的功能来实现这些需求。

在数据库分析阶段，需要确定用户的类型，也就是谁将使用系统，以及为完成他们指定的任务所必需赋给他们的访问级别。关系 DBMS 提供两种类型的数据库安全：

- 系统安全：系统级的数据库访问和使用，例如用户名和密码。
- 数据安全：数据库对象级的访问和使用（例如表、视图、存储过程等）以及用户在这些对象上可以执行的操作。

SQL Server 安全采用两层模型，第一层是访问 SQL Server，涉及验证所连接人员的有效 SQL Server 账号，称为登录，这些登录可以是 Windows 账号、Windows 组或 SQL Server 登录（如果是混合方式）。第二层是访问数据库，由于 SQL Server 支持多个数据库，因此每个数据库都有自己的安全层，通过用户账户提供对数据库的访问。在生成登录之后，还要把它们映射到用户，以提供数据库访问。一个登录可以映射多个数据库中的用户。

每个登录和数据库用户都应该具备一定的权限才能访问数据库对象。在 SQL Server 2000 中，权限是通过角色（包括固定服务器角色、固定数据库角色和用户定义角色）来指定的。

8 引入受控冗余的考虑

在 4.2 小节中，我们提到了利用规范化思想来降低数据库冗余，节省存储空间。但是，规范化的数据库设计可能不能提供最大的处理效率，并且在很多情况下，应用系统对系统的响应时间比存储空间的要求更严格，因此，考虑通过降低数据表的范式级别（即“反规范化”）

来换取性能上的提升还是值得的。这就是本节要讨论的主要内容。

但是，这不能说明可以把规范化从逻辑数据库设计中省略，因为反规范化不是在任何情况下都是必需的，它只是在数据库不能满足性能要求，并且表的更新率较低，查询率较高的前提下才是被提倡的。因为反规范化在带来性能上的改善的同时，也存在下列弊端：

- 使实现更加复杂；
- 会牺牲灵活性；
- 可能加快检索速度，但会降低更新速度。

但是，何时进行反规范化没有固定的规则。接下来的部分，就是对一些比较常见的反规范化的情况进行讨论。

8.1 合并一对一关系

在合并后，表中可能会出现大量的空值，造成空间的浪费。因此，需要衡量合并所带来的性能的提升是否值得。

8.2 复制一对多关系中的非键列来减少连接

这种处理也会带来一些问题，例如，首先由于复制列的存在会占用额外的空间，并且，如果修改了父表中的复制数据，子表中的相应数据也必须进行更新，这也需要花费额外的处理时间。

8.3 复制一对多关系中的外键列来减少连接

注意：表之间的关系是 1：*还是*：*，不能随意复制。

8.4 复制多对多关系中的列来减少连接

在步骤 2.1 中，将每个*：*关系映射为三张表：两个从原始实体派生的表和一个表达二者关系的表。现在，如果要从*：*关系中产生信息，必须要连接着三个表。不过，有时候在中间表中复制原始实体中的列可以减少连接。

8.5 引入重复组

为了规范化的考虑，应该从逻辑数据模型中把重复组分离出来，形成新表，与其父表形成 1：*的关系。有时，再次引入重复组是提高系统性能的很有效的方法。

8.6 创建提取表

如果不得不在每天峰值的时候来统计复杂报表，并且报表所需数据并不要求是最新的情况，则可以考虑专门创建一张基于报表所需要的表的提取表，并且允许用户直接访问提取表代替访问基本表。而在系统使用最少的时候来生成提取表是常见的做法。

8.7 分区表

随着表中存储数据的快速增长，在整个表中执行某个查询可能是相当耗时的。因此，可以考虑对超大数据库（VLDB, Very Large Database）进行数据分区以减少在执行操作时的时间和性能问题，分为垂直分区和水平分区。

1．垂直分区

垂直分区就是将宽表的多列或大列分解为两个或多个表，以提高性能和可管理性。这种情况适合于包含文本或图形数据的表或有些列很少访问的表。有时候，分区是为了弥补最初没有正确规范化的数据。

将表垂直分区为多个表时，可以用文件组将表分配到多个磁盘中，提高 I/O 性能并允许按文件组备份。当将表垂直分区到多个服务器中时，可以伸缩数据库，提高性能和存储容量。这种情况下，表不是放在一个数据库的多个文件组中，而是放在两个或多个服务器中。这些服务器常被配置成链接服务器，修改链接服务器中的数据时，要执行分布式事务。

2．水平分区

水平分区是对表中数据按数值分配到多个表中。这里分解的是长表而不是宽表，例如，可以将包含一年数据的表分解为四个表，每个表包含一季度的记录，或者将企业数据按子公

司进行分解，每个表包含本子公司的记录。

与垂直分区一样，分解过的表可以分布在多个文件组、数据库或服务器中。

注意：在反规范化时，应该考虑它对前面的步骤的影响。例如，可能需要重新考虑已进行反规范化的表中的索引的选择，以检查现存的索引是否应该被删除，或添加新的索引。另外，需要考虑如何维护数据完整性，如使用事务来维护，但不提倡采用触发器（可能会引发其他的性能问题）。

9 监视并调整操作系统

本步骤的目标就是监视操作系统并改善系统的性能以改正不正确的设计决策，或者反映变化的需求。

不应该将原始的物理数据库看出是静止不变的，而是要估计操作系统会怎样实现。许多 DBMS 提供了让数据库管理员（DBA）监视系统的操作并调整系统的功能。为了保证系统运行正常，还有必要不断地监视与调制操作系统，以便获取系统的持久地成功。

另外，一旦系统投入运行，作为用户反馈和更改需求的结果，有些更改是需要的。有时，这些变化只不过是表面的，有时就需要修改数据库结构。这时，就必须重新进行一些逻辑和无理设计中步骤，以确保正确的设计和实现这些更改。

例如，需要在网站上发布图书的封面和内容介绍，因而需要增加新的字段。而这种变化可能引入的主要问题是潜在地扩大了所需磁盘空间，以便存储大量的图像文件和文本资料。

10 小结

本实验指导书对数据库设计方法学中的各个步骤作了概要性的介绍。对于成功的数据库设计，下面这些指导建议是很重要的：

- 尽可能多地与用户交流；
- 在整个数据建模过程中使用一种结构化方法学；
- 使用数据驱动方法；
- 在数据模型中加入结构化和完整性考虑；
- 将规范化和事务有些性急世界合金方法学中；
- 尽可能多地使用图形去表示数据模型；
- 构建数据字典补充数据模型图；
- 乐于重复以上步骤。

这些建议都被融入方法学中，在本指导书各节中都有所体现，希望读者能够认真体会，在实践中灵活运用。

参考文献：

1. 数据库设计教程（第二版）.Thomas M Connolly, Carolyn E.Begg 著.何玉洁，黄婷儿 等译.机械工业出版社.2005.1
2. 数据库系统概论（第四版）.王珊，萨师煊.高等教育出版社.2006.5
3. SQL Server2000 实用全书. 邱仲潘 编著.电子工业出版社.2002