简介

章节纲要

数据和数据库管理 2

数据库生命周期 3

概念性数据建模 9

小结 10

数据库专业人员的提示和见解 10

文献摘要 11

自从关系型数据库系统兴起并最终占据主导地位以来，数据库技术在过去的三十年中发展迅速。虽然许多专门的数据库系统（面向对象、空间、多媒体等）在科学和工程领域找到了次要的用户群体，但关系型系统仍然是商业企业的主导数据库技术。

关系型数据库设计已经从一门艺术发展到一门科学，它已经可以部分实现为一套软件设计辅助工具。许多这样的设计辅助工具已经作为计算机辅助软件工程（CASE）工具的数据库组件出现，其中许多提供了使用简化数据建模方法的交互式建模能力。逻辑设计--即基本数据关系的结构和它们在特定数据库系统中的定义--主要是应用设计师的领域的这些设数据库管理员（DBA）。今天的DBA有各种供应商提供的工具来帮助设计最高效的数据库。本书致力于介绍当今最流行的关系型数据库的逻辑设计方法和工具。物理设计方法和工具将在另一本书中介绍。

在本章中，我们回顾了数据库管理的基本概念，并介绍了数据建模和数据库设计在数据库生命周期中的作用。

# 数据和数据库管理

文件系统中文件的基本组成部分是数据项，它是在现实世界中有意义的最小的命名数据单位--例如，姓、名、街道地址、身份证号码和政党。一组被应用程序视为一个单位的相关数据项被称为记录。记录类型的例子有订单、销售人员、客户、产品和部门。文件是一个单一类型的记录的集合。数据库系统在这些定义的基础上进行了扩展。在关系型数据库中，一个数据项被称为列或属性，一个记录被称为行或元组，一个文件被称为表。

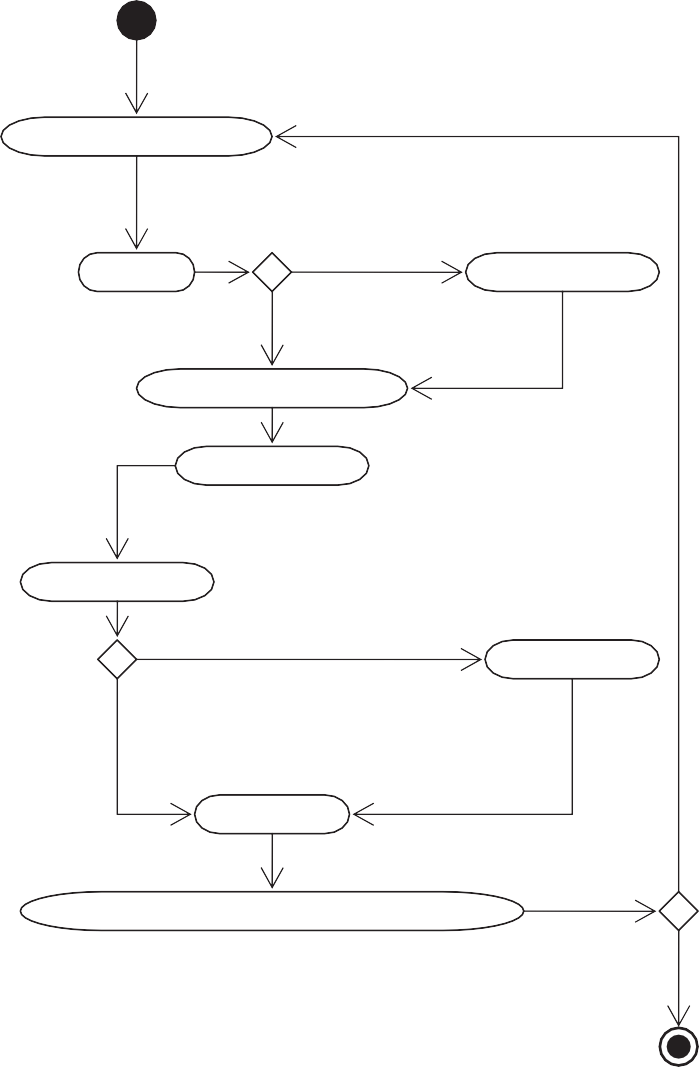
数据库是一个更复杂的对象；它是一个相互关联的存储数据的集合，为一个或多个组织内的多个用户的需要服务--也就是说，是许多不同类型的表的相互关联的集合。使用数据库而不是文件的动机是为不同的用户提供更多的服务，整合数据以方便复杂交易的访问和更新，并减少数据的冗余。

数据库管理系统（DBMS）是一个用于操作数据库的通用软件系统。DBMS支持逻辑视图（模式、子模式）；物理视图（访问方法、数据集群）；数据定义语言；数据操作语言；以及重要的工具，如事务管理和并发控制、数据完整性、崩溃恢复和安全性。关系型数据库系统是成熟的商业数据库的主要系统类型，它比早期的分层和分层系统提供了更大程度的数据独立网络（CODASYL）数据库管理系统。数据独立性是指对数据库的逻辑或物理结构进行改变而不需要对应用程序重新编程的能力。它也使得数据库的转换和重组变得更加容易。关系型数据库管理系统提供了比以往系统更高的数据独立性；它们是我们讨论数据建模的重点。

# 数据库生命周期

数据库生命周期包含了设计逻辑数据库的全局模式、在计算机网络中分配数据、以及定义本地DBMS特定模式所涉及的基本步骤。一旦设计完成，生命周期将继续进行数据库的实施和维护。本章包含了对数据库生命周期的概述，如图1.1所示。在接下来的章节中，我们将关注数据库设计过程，从需求建模到逻辑设计（以下步骤I和II）。我们在图1.2中用一系列的图来说明生命周期中每一步的结果。每张图都显示了每一步输出的可能形式，这样读者就可以看到设计过程中从一个想法到实际数据库实现的进展。这些形式将在第2-6章中更详细地讨论。

1. *需求分析。*数据库的要求是通过采访数据的生产者和使用者，并利用这些信息产生一个正式的需求规格来确定。该规范包括处理所需的数据，自然的数据关系，以及数据库实施的软件平台。作为一个例子，图1.2（步骤一）显示了产品、客户、销售人员和订单的概念在采访过程中在终端用户的头脑中形成。
2. *逻辑设计*。全局模式，一个显示所有数据及其关系的概念性数据模型图，是使用实体-关系（ER）或UML等技术开发的。数据模型的结构必须最终转化为表。



|  |  |  |
| --- | --- | --- |
| 确定需求 | **信息要求** | |
|  | **逻辑设计**  [多视角]  [单视图]  转化为SQL表  正常化 | ModelIntegrate 视图 |
| 选择索引  [否则] | **物理设计**  [特殊要求] | 去正规化 |
|  | **实施**  实施  监测和检测不断变化的需求 | [否则]  [已停用] |

图1.1 数据库的生命周期。

1. 概念性数据建模。数据要求通过使用ER或UML dia- gram进行分析和建模，其中包括我们将在第二章和第三章研究的许多特征，例如，可选的关系、三元关系、超类型和子类型（类别）的语义。处理要求通常使用自然语言表达式或SQL命令以及出现的频率来指定。图 1.2（步骤二.a）显示了一个可能的ER

**步骤I 信息要求（现实）。**



产品

销售人员

订单

客户

**第二步 逻辑设计**

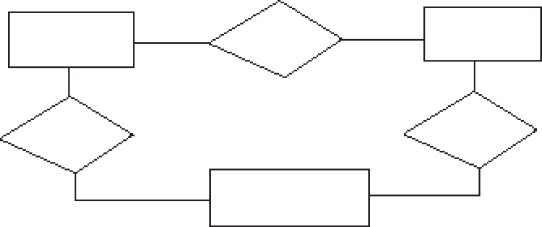
**步骤二.a 概念性数据建模**

**数据库生命周期**

客户

零售销售人员观 N

N 订单 N 产品

N

点

**步骤二.b 查看整合**

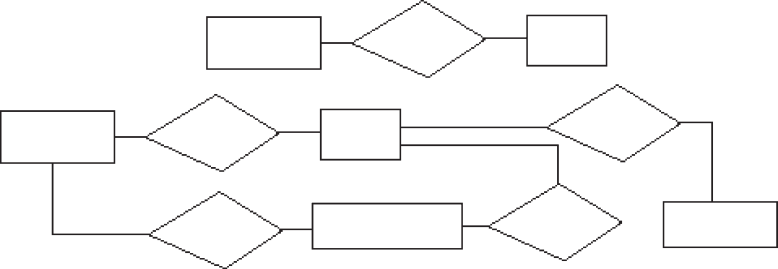
被服务的 售出时

1 销售人员 N

整合零售销售人员和顾客的意见

图1.2 生命周期的结果，一步一步来（在下页继续

）。



客户观点

客户

1

地方

N

秩序

客户

1

地方

N

N

秩序

为

N

N

N

被服务的

1

销售人员

1

填补了

产品

终端用户心目中的产品/客户数据库的模型表示。

1. *观*点整合。通常情况下，当设计规模较大，并且有超过一个人参与需求分析时，会出现对数据和关系的多种看法，由于分类学、背景或认知的不同而导致不一致。为了消除模型中的冗余和不一致，这些观点必须

**步骤二.c 将概念数据模型转换为SQL表**客户



监护人-编号

监护人姓名 ..........

产品



产品编号 产品名称库存数量

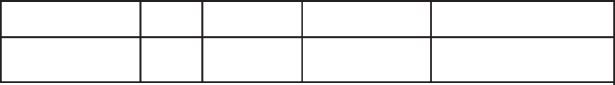
创建表**customer** (cust-no integer, cust-name char(15), cust-addr char(30),

sales-name char(15),

prod-no整数，主键（ cust-no）。

外键（sales-name）。推荐**销售人员**。

外键（prod-no）引用**产品**）。



销售人员的

地址

部门

假期-日子

姓名

职位级别



秩序

订单-编

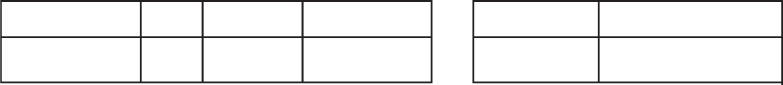
销售人员姓名：No

订单-产品

订单-编 产品-编

**步骤二.d SQL表的规范化**

分解表格和消除更新异常。销售人员 销售人员



销售名称

地址 部门

职位级

职位级

假期-日子

图1.2，续

进一步的生命周期结果

，一步步来。

**第三步 物理设计**

索引 聚类 分区 物化视图 反规范化

被 "合理化 "并整合为一个单一的全局视图。视图整合需要使用ER语义工具，如同义词的识别、聚合和概括。在图1.2（步骤II.b）中，产品/客户数据库的两个可能的视图被合并成一个基于客户和订单共同数据的全局视图。当应用程序必须被整合时，视图整合也很重要，每个应用程序都可能用自己的数据库视图来编写。

1. 将概念数据模型*转换为SQL*表。基于数据建模概念的分类和一组映射规则，每个关系和其相关的实体都被转化为一组DBMS特定的候选关系表。我们将在本章中展示这些在标准SQL中的转换。

5.在这个过程中，多余的表被消除了。在我们的例子中

，图1.2中步骤II.c的表格是步骤II.b中综合ER模型转换的结果。

1. 表的*规*范化。给定一个表（R），如果在每个瞬间，每个A值正好与一个B值相关，那么一组属性（B）在功能上依赖于另一组属性（A）。功能性依赖（FDs）是从概念数据模型图和需求分析中的数据关系的语义中得出的。它们代表了作为实体唯一标识符（键）的数据元素之间的依赖关系。额外的FDs，代表了实体内键和非键属性之间的依赖关系，可以从需求规范中导出。与所有派生的FDs相关的候选关系表被规范化（即，通过将表分解或拆分成更小的表来修改），使用标准的规范化技术。最后，在必须保持数据完整性的前提下，进一步分析在规范化的候选表中出现的数据冗余，以确定是否可以消除。图1.2显示了从步骤II.c到步骤II.d的一个将销

售人员表规范化为新的销售人员和SalesVacations表的例子。

我们在此注意到，数据库工具供应商倾向于使用

术语*逻辑*模型指的是概念数据模型，他们使用术语物理

模型指的是DBMS特定的实现模型（例如，SQL表）。我们还注意到，许多概念数据模型不是从零开始的，而是从现有的DBMS特定模式的逆向工程过程中得到的（ Silberschatz等人，2010）。

1. 物理*设计*。物理设计步骤包括索引（访问方法）的选择、分区和数据的聚类。步骤二中的逻辑设计方法通过减少需要分析的数据依赖关系的数量，简化了设计大型关系数据库的方法。这是通过在传统的关系型设计方法中插入概念性的数据建模和整合步骤（图1.2的步骤II.a和II.b）来实现的。这些步骤的目标是对现实的准确表述。当概念数据模型转化为关系模型时，通过对创建的候选表进行规范化处理来保持数据的完整性。物理设计的目的是为了优化性能。作为物理设计的一部分，如果在效率方面有明显的巨大收益，有时可以以有限的方式改进全局模式以反映处理（查询和事务）的要求。这就是所谓的去*规*范化。它包括根据高频率、高容量或明确的优先级来选择重要的进程；定义简单的表扩展，以提高查询性能；评估查询、更新和存储的总成本；并考虑副作用，如可能的完整性损失。这对在线分析处理（OLAP）应用尤其重要。
2. 数据*库*的*实*施、*监*控和修改。一旦设计完成，就可以通过使用DBMS的数据定义语言（DDL）实现正式的模式来创建数据库。然后，数据操作语言（DML）可以用来查询和更新数据库，以及设置索引和建立约束，如参考完整性。 SQL语言包含了DDL和DML的结构；例如，*创*建表命令代表了DDL，而*选择*命令代表了DML。

当数据库开始运行时，监测表明性能要求是否被满足。如果没有得到满足，应该进行修改以提高性能。当需求改变或结束时，可能需要进行其他的修改

用户的期望随着良好的性能而增加。因此，生命周期继续进行着监测、重新设计和修改。在接下来的两章中，我们首先看一下基本的数据建模概念；然后，从第四章开始，我们将这些概念应用于数据库设计过程。

# 概念性数据建模

概念性数据建模是逻辑性数据库设计的驱动部分。让我们来看看这个重要的组成部分是如何产生的，以及为什么它很重要。模式图是由Charles Bachman在20世纪60年代正式提出的。他用矩形来表示记录类型，用从一个记录类型到另一个记录类型的定向箭头来表示这两种类型的记录实例之间的一对多关系。用于概念数据建模的*实*体关系（ER）方法是本书强调的两种方法之一，在第2章有详细描述，它是由Peter Chen在1976年首次提出的。陈氏ER模型的形式使用矩形来指定实体，这在某种程度上类似于记录。它还使用菱形对象来表示各种类型的关系，这些关系通过放置在连接菱形和矩形的线上的数字或字母来区分。

统一建模语言（UML）被引入

1997年由Grady Booch和James Rumbaugh提出，已经成

为指定和记录大规模软件系统的标准图形语言。UML（现

在的UML-2）的数据建模部分与ER模型有很大的相似性

，将在第三章详细介绍。我们将同时使用ER模型和UML来说明本书中的数据建模和逻辑数据库设计实例。

在概念数据建模中，最重要的是强调简单性和可读性。在 ER和UML方法最有用的地方，概念模式设计的目标是以一种简单而有意义的方式来捕捉现实世界的数据需求，使数据库设计者和终端用户都能理解。终端用户是指负责访问数据库通过使用DBMS软件来执行查询和更新，因此在数据库设计过程中具有既得利益。

# 摘要

数据建模和数据库设计技术的知识对于数据库从业者和应用开发者来说非常重要。数据库生命周期显示了设计数据库的方法需要哪些步骤，从独立于系统环境的逻辑设计，到基于数据库管理系统细节的物理设计，以实现数据库。在各种数据建模方法中，ER和UML数据模型可以说是目前使用最多的，因为它们的简单性和可读性。

# 为数据库专业人员提供的提示和洞察力

提示1。有条不紊地完成生命周期的各个步骤。每个步骤都有明确的定义，并且产生了一个结果，可以作为下一个步骤的有效输入。

提示2.尽快纠正设计错误，回到上一个步骤，尝试新的替代方案。你等得越晚，错误的代价就越大，修复的时间就越长。

提示3.把逻辑设计和物理设计完全分开，因为你要满足完全不同的目标。

*逻辑设计*。其目的是获得一个可行的解决方案，以满足所有已知和潜在的查询和更新。有许多可能的设计；没有必要找到一个 "最好的 "逻辑设计，只需要找到一个可行的。把优化的努力留给物理设计。

物理*设计*。其目的是优化已知和预测的查询和更新的性

能。

# 文献摘要

许多早期的数据建模工作是由Bachman（1969，1972）、 Chen（1976）、Senko等人（1973）等人完成的。坚持本章所述的关系数据库生命周期的重要部分的数据库设计教科书有Teorey 和Fry （ 1982 ）、Muller （ 1999 ）、Stephens 和 Plew（2000）、Silverston（2001）、Harrington（2002）

。Bagui（2003），Hernandez和Getz（2003），Simsion和 Witt （ 2004 ）， Powell （ 2005 ）， Ambler 和Sadalage （ 2006），Scamell和Umanath（2007），Halpin和Morgan（ 2008 ）， Mannino （ 2008 ）， Stephens （ 2008 ）， Churcher（2009），以及Hoberman（2009）。

Jenson和Snodgrass（1996）以及Snodgrass（2000）对时间（时间变化）数据库进行了定义和讨论。其他常用的概念数据建模方法包括IDEF1X（ Bruce， 1992 ； IDEF1X ， 2005 ）和Zachmann 框架的数据建模部分（ Zachmann ， 1987 ； Zachmann 框架推进研究所， 2005 ）。Harriman, Hodgetts, and Leo (2004)讨论了开发过程中模式的演变，这是一个经常发生的问题。