第3章 关系数据模型

CHAPTER

关系数据模型是目前数据库领域占主导地位的数据模型，它与以前的层次模型和网状模型最大的不同在于，关系模型有着坚实、严格的理论基础。基于关系模型的RDBMS，也是当前数据库市场上占据绝对地位的数据库产品。

因此，本章将对关系数据模型中的主要技术作详细介绍。由于关系模型上的完整性约束（或限制），在实际项目的设计与开发中应用普遍，为使读者能学以致用，掌握该实用技术，本章将结合目前数据库市场上较流行且易于上手的Microsoft SQL Server产品，详细讲解关系模型上的完整性约束，及其在SQL Server上的具体体现和使用。在数据库应用开发中，常常要将高级数据模型向关系模型转换，本章将详细重点地介绍实体联系模型和对象模型分别向关系模型的转换方法。之后，还将对关系数据模型的理论基础之一——关系代数和关系运算，结合SQL语言进行全面描述。

本章学习目的和要求：

（1）关系数据模型的基本概念；
（2）关系模型上的完整性约束；
（3）实体联系模型向关系模型的转换；
（4）对象模型向关系模型的转换；
（5）关系代数；
（6）关系运算。

3.1 SQL语言简介

由于本章涉及关系模型完整性约束的设定和使用，而完整性约束的设定需要用到关系数据库语言——SQL（structured query language，结构化查询语言），它是一种用得最为广泛的关系数据库语言，因此，本节将首先简要地对SQL语言及基本命令进行介绍。同时，本章中所使用的均是最基本的SQL命令，容易接受，这些SQL命令的使用，也将为下一章SQL语

言的学习打下良好的基础。

SQL语言是最早在IBM System-R RDBMS上使用的查询语言，由于其广泛的使用，出现了标准化需求，于是形成SQL标准。有了SQL标准，用户可评判厂家的SQL版本。

第一个SQL标准，由ANSI（American National Standards Institute，美国国家标准协会）于1986年制订，称为SQL-86。1989年作了些许改进，推出的版本称为SQL-89。1992年，由ANSI和ISO（International Organization for Standardization，国际标准化组织）合作，作了较大改动，推出的版本称为SQL-92，以前曾称作SQL2（表示SQL的第2个版本），这是目前大多数商用RDBMS支持的版本。相应地，SQL-86和SQL-89属于SQL的第1个版本，可记为SQL1。1999年提出SQL:1999，曾称作SQL3（表示SQL的第3个版本），是SQL-92的扩展，引入了对象关系特征及其他许多新的功能。2003年提出的SQL:2003，对SQL:1999标准做了细微的扩充。SQL:2006标准增加了几个与XML相关的特性，SQL:2008标准则引入了许多对SQL语言的扩展。

说明：SQL版本改进更方便于SQL的使用。作为一个有说服力的例子是，用SQL-89创建表，如果在表创建完后需要增加一列或多列，由于SQL-89没有提供此修改功能，因此要完成此项工作，就必须先删除要修改的表，然后再创建一张同名的加入了新列的表。而SQL-92则具有在表创建好后增加列的功能。正是由于此原因，有人将SQL-89称作是“刚性”的，而SQL-92则较“柔性”。

SQL语言一般分成3个子语言，即数据定义语言（data definition language，DDL）、数据操纵语言（data manipulation language，DML）和数据控制语言（data control language，DCL）。

DDL用来定义数据库、数据结构及完整性约束等；DML用来操纵数据，主要包括增加、删除、修改和查询数据；而DCL则用来对数据库的事务、安全性等进行控制。

由此可见，SQL尽管称为查询语言，但其功能远远不只是查询，还可定义数据结构、定义完整性约束、对数据进行增加、删除和修改、实施事务控制、安全控制等。之所以称其为查询语言，可能是因为用户在数据的管理活动中，查询操作所占的比重比较大。

关系模型中的关系（relation）利用支持SQL-92标准的SQL命令来定义和操纵。SQL-92标准中，用表（table）来代表关系模型中的关系。SQL中，用于创建（其命令关键字为create）、删除（drop）和修改（alter）表结构的部分，属DDL；而对表中数据进行增加（其命令关键字为insert）、删除（delete）、修改（update）和查询（其命令关键字为select）的部分，属DML。关于DCL的命令，将在第5章涉及，主要包含安全授权中的GRANT和REVOKE，以及事务控制中的BEGIN TRAN、COMMIT和ROLLBACK，详细内容请参见第5章。

说明：

① 在DDL和DML中，均有修改和删除命令，但各自的命令关键字不同，应该注意。

② 本章将在相应的部分使用性地介绍SQL的定义与操纵子语言中最常用的几个命令的最基本的语法，以使读者对SQL命令先有一个感性认识，详细的命令语法将在第4章具体介绍。

③ 再次强调，关系模型的学习仍然应抓住数据模型的3个要素，即数据结构（主要是数据和联系的描述）、数据操作（增、删、改、查询）和约束。本章将主要介绍关系模型的数

据结构和约束，数据操作将在第 4 章通过 SQL 操纵语言介绍。

3.2　关系数据模型的数据结构

关系数据模型 RM(relational model)，于 1970 年首次由 E. F. Codd 提出，而当时大多数数据库系统 DBS(database system)，均基于层次或网状模型。关系模型带来了数据库领域的革命，并大量替换了以前的模型。20 世纪 70 年代中期，最早的 RDBMS(relational database management system，关系型数据库管理系统)，在 IBM 和加州大学伯克利分校开发，不久即有一些厂家提供关系数据库产品。目前，RM 已成为决定地位的数据模型，并成为目前最流行的 RDBMS 的基础。

任何一个数据模型，其内容均会包括 3 个方面，即静态数据结构、动态数据操作和完整性约束(integrity constraints，ICs)。在学习过程中，应始终抓住这三点，才不至于因内容的繁多而迷失方向。

在静态数据结构方面，主要是介绍，该数据模型如何表达数据本身(借用第 2 章的概念，即为实体)和数据间的联系；在动态数据操作方面，一般只会涉及 4 个操作，即对数据模型中的数据的增加、删除、修改和查询，俗称“增删改查询”；在完整性约束方面，则会因具体的数据模型的不同而不同。

在关系数据模型的数据结构方面，数据本身和数据之间的联系都是利用关系来表达的。

关系：用于描述数据本身、数据之间的联系，俗称表。

构成：由行(row)和列(column)组成。

列：有时也称字段(field)，与属性(attribute)、数据项、成员(member)同义。

行：有时也称元组(tuple)、记录(record)。

说明：与第 2 章的实体联系模型相比，关系模型在表达数据及其联系方面相对要简单些。因为在 ERM 中，数据本身和数据间的联系是分别用实体和联系这两个概念来表达，并且可以图示，所以 ERM 的表达更直观、更易于理解。而在关系模型中，则只用关系这一个概念，既可表达数据本身(即实体)，又可表达数据间的联系(即联系)。关系模型这种表达上的简单，导致对其理解就比 ERM 较困难些。当然，这种表达上的简单更便于计算机的实现。

例 3-1　关系示例。

“学生信息表”这个关系，表达的是“学生”这个实体数据本身，而“学生选课表”这个关系，表达的是“学生”实体与“课程”实体间的联系。尽管一个是实体，而另一个是联系，但它们都可用表(即关系)来表达。

域(domain)：指列(或属性)的取值范围。

关系模式(schema)：是对关系的描述，由关系名及各个列构成。其描述的一般形式为：$R(A_1, A_2, \cdots, A_n)$，其中，R 为关系名，$A_i(i=1, 2, \cdots, n)$ 为关系 R 的属性。

例如，学生模式可表示如下：学生(学号，姓名，性别，生日，所属学院)。

说明：关系模型与关系模式之间的关系，对应数据模型与数据模式之间的关系。

关系实例(instance)：记录或元组的集合。如用 t 表示元组，则 $t=<a_1, a_2, \cdots, a_n>$，其中，$a_i(i=1, 2, \cdots, n)$ 为对应属性 A_i 的值，则关系实例可描述为 $\{<a_1, a_2, \cdots, a_n>\}$。

一个学生关系的实例为{<0001，张三，男，1987.10.12，计算机学院>，<0002，李四，

男，1986.05.28，理学院＞，…｝。

在不引起混淆的情况下，关系实例经常简称为关系。一般，列的顺序要比行的顺序重要些，在后续的关系代数中会看到这一点。因为在关系代数中，有时用列顺序号来代表列名。每个元组的字段必须对应关系模式中的字段。

候选键(candidate key)：能唯一识别关系实例元组的最小字段集。一个关系可能有多个候选键。它与E-R模型中候选键的概念相同。

主键(primary key，PK)：一个唯一识别关系实例元组的最小字段集合。可从关系的候选键中，指定其中一个作为关系的主键。一个关系最多只能指定一个主键。它也与ERM中主键的概念相同。

说明：事实上，关系模型中的很多概念，如主键、候选键、属性和域等，与实体联系模型中的对应概念意义基本相同。

以上所介绍的都是围绕一张表来进行的。在关系模式中，对于表与表之间的关系有一个重要的概念，此即外键，它是表间联系的重要纽带。这是ERM中所没有涉及的。

外键(foreign key，FK)：一张表中的某个属性(组)是另一张表中的候选键或主键，则称该属性(组)为此张表的外键。

例如，图3-1的主键、外键示意图中，若指定“班号”为“班级表”中的主键，而它又出现在“学生表”中，则称其为“学生表”的外键。

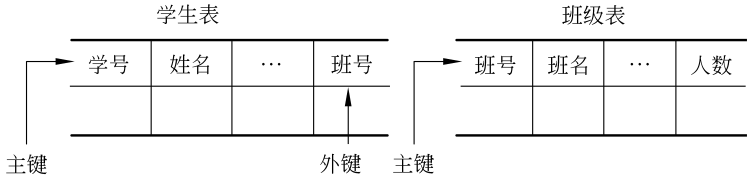


图3-1 主键、外键示意图

说明：要使某张表中的某个属性(组)，成为另一张表的外键，必须指定该属性(组)为该张表的主键或候选键。如何指定？这就是“关系模型上的完整性约束”所要介绍的。

至此，关系模型中关于静态数据结构的描述即告结束。可以看出，这一部分相当简单。由于关系模型的动态数据操作只是增、删、改、查询4个操作，非常简单，在此就不再介绍。在第4章的SQL数据操纵语言部分，会看到与其对应的4个命令(即INSERT、DELETE、UPDATE和SELECT)的具体语法，而且本章会用到这4个数据操纵命令最基本的使用方法。

3.3 关系模型上的完整性约束

下面，将重点介绍关系模型的完整性约束部分。

3.3.1 完整性约束简介

利用完整性约束(或限制)，DBMS可帮助用户阻止非法(invalid)数据的输入。完整性约束ICs实际上指出了要求存入DB的数据应满足的约束条件。

SQL-92 标准规定，关系模型中可被指定的完整性约束，包括域约束（domain constraint）、主键约束（primary key constraint）、唯一约束（unique constraint）、外键约束（foreign key constraint）、一般性约束（general constraint）等。

为什么要用完整性约束？其实，用一个简单的例子即可说明。

例如，前面谈到的主键，对它的描述是“一个唯一识别关系实例元组的最小字段集合”。由于关系模型中的完整性约束，不仅用来描述现实系统中的一些规定，关键是要拿来用（使这些约束发生作用），而不像 ERM 基本上只是用来描述，因此，应说到做到。要做到唯一识别关系实例元组，必须要求主键的值既不能重复，又不能为空值。但如何在实现中保证主键的值不重复且不为空的条件呢？当然，可以由程序员在程序中对主键值进行判断，以使其满足这一条件，但这样会增加程序员的工作量及程序的代码量。

另一种方法是由 RDBMS 来帮助判断主键值，使其满足指定的条件，这样做还可减少程序员的工作量和程序的代码量。RDBMS 通过给定主键约束来实现这一功能，用户只要将某一表中的主键设定主键约束，即可让 RDBMS 帮助判断该主键的值是否重复和为空。因此，主键约束是 RDBMS 中保证主键完整性的一种措施。

指定完整性约束的时间：可在定义一个关系模式的同时定义完整性约束，或在定义完关系模式后再追加约束的定义。

完整性约束的实施：完整性约束一旦指定，当 DB 应用程序运行时，DBMS 检查插入或修改的数据，看其是否满足相应完整性约束指定的条件。如满足，则接受该插入或修改的数据；否则，拒绝接受，并返回出错信息。

下面，将分别介绍 SQL-92 标准规定的关系模型上的几种完整性约束。

3.3.2　域约束

域约束是指对列数据类型的约束，这是关系模型中最基本的约束。例如，某列的数据类型为“整数型”，那么，该列的值就不能为字符或字符串，否则就出错。

这种约束一般不需要显式指定，它是根据列数据类型的定义而自动起作用。

3.3.3　主键约束

主键约束主要是针对主键，以保证主键值的完整性。主键约束要求主键值必须满足两个条件：

（1）值唯一；

（2）不能为空值。

主键约束需要显式指定，但不需要指明约束的条件，因为 RDBMS 知道主键约束要求主键值应满足的条件。显式指定主键约束的方法，在后续的内容中有介绍。

说明：

① 指定了表中的主键约束，也即是指定了该表的主键。

② 一张表只能指定一个主键约束，因为一张表只允许有一个主键。

3.3.4 唯一约束

唯一约束主要是针对候选键，以保证候选键值的完整性。唯一约束要求候选键满足2个条件：

（1）值唯一；

（2）可有一个且仅有一个空值。

对于候选键，由于它也是一种键，也能唯一地识别关系实例元组。但其中只能有一个作为主键，该主键可用主键约束来保证其值的完整，其他的候选键也应有相应的约束来保证其值的唯一，这就是唯一约束。因此，表中的候选键可设定为唯一约束。反过来说，设定为唯一约束的属性（组）就是该表的候选键。

唯一约束也需要显式指定，但不需要指明约束的条件，因为RDBMS也知道唯一约束要求候选键应满足的条件。

说明：

① 指定了表中的唯一约束，也即是指定了该表的候选键。

② 一张表可以指定多个唯一约束，因为一张表允许有多个候选键。

3.3.5 外键约束

1. 简介

如果将关系模型上的完整性约束按属于“表本身”还是属于“表间”来分类，则以上的域约束、主键约束和唯一约束属“表本身”的完整性约束，而外键约束则属“表间”的完整性约束。

为描述外键约束的方便，首先定义两个概念，即主表和从表。从表是指含有外键的表，或外键所在的表，而主表是指外键在另一张表中作为主键或候选键的表。

以图3-1为例，“班级表”为主表，而“学生表”则为从表。

外键约束的目的是维护表与表之间外键所对应属性（组）数据的一致性，即一张表这个属性（组）值的改动，可能要求另一张表对应属性（组）的值要作相应改动；或一张表这个属性（组）值的改动，应参照另一张表对应属性（组）的值，以保持两表中对应属性（组）数据的一致。为使RDBMS能帮助完成这样的功能，则应指定这种涉及两个表的完整性约束，此即外键约束，有时又称参照完整性约束（reference integrity constraints）。

为说明这种表间完整性的维护方法，首先应了解两个通过外键联系的表（即主表和从表），对它们分别进行外键所对应属性（组）的插入、删除和修改，这3种数据操作对完整性的影响。

2. 主表主键/候选键的操作对从表的影响

为叙述和理解的方便，特以图3-2中的两张关系实例为例进行说明。其中，学生表中的学号为主键、班号为外键，班级表中的班号为主键。

对班级表（即主表）的班号进行如下操作。

（1）插入（INSERT）。插入一新班级＜计0009，计算机09班，…，70＞，从图3-2可以看出，该班级信息的插入符合该表主键约束要求，并且对“学生表”（从表）中的记录不产生

任何影响。

学生表

学　号	姓　名	…	班　号
0101	张三	…	计 0001
0102	赵一	…	计 0001
0201	李四	…	计 0002
0601	王五	…	计 0006
⋮	⋮	⋮	⋮

班级表

班　号	班　名	…	人　数
计 0001	计算机 01 班	…	60
计 0002	计算机 02 班	…	65
计 0006	计算机 06 班	…	66
⋮	⋮	⋮	⋮

图 3-2　学生表和班级表实例

因此，主表中主键值的插入，不会影响从表中的外键值。

(2) 修改(UPDATE)。图 3-2 班级表中，班号是按序号编排的，假定想将班号按“计 xxyy”样式进行修改，其中，xx 表示“年”的后两位，yy 表示序号，则除了要修改班级表中的所有班号外，还要修改学生表中对应的班号。例如，两张表中的班号“计 0001”、“计 0002”、“计 0006”，分别改为“计 9801”、“计 9802”和“计 9901”。

因此，如果从表中的外键值与主表中的主键值一样的话，主表中的主键值的修改要影响从表中的外键值。那么，在这种情况下，如何保证两表数据的一致性呢？

事实上，SQL-92 标准提供了此种情况下可选用的 4 种行动策略：其一，改变从表中所有对应的外键值，使之与主表中的主键值一致，此即级联修改，所用的命令为“CASCADE”；其二，将受到影响的外键值修改为空值(NULL)，所用命令为“SET NULL”；其三，将受到影响的外键值修改为该属性的默认值(DEFAULT)，所用命令为“SET DEFAULT”；其四，当从表中存在相应的外键值时，不允许修改主表中的主键值，此即“禁止”修改，所用命令为“NO ACTION”。不过，SQL Server 和 Sybase 只实现了其中的两种，即“CASCADE”和“NO ACTION”。

(3) 删除(DELETE)。假如要删除班级表中的“计 0006”班级的信息，则由于该班级在学生表中已分配有学生，如“王五”，因此，为保证表间数据一致性，需要删除学生表中所有外键值“计 0006”对应的元组，或采取其他能保证一致性的行动。

因此，主表中主键值的删除，可能会对从表中的外键值产生影响，除非主表中的主键值没有在从表中的外键值中出现。

同样，SQL-92 标准提供了此种情况下可选用的 4 种行动策略：其一，删除从表中、所有与主表主键值相同的外键值对应的元组，此即级联删除，所用的命令为“CASCADE”；其二，将受到影响的外键值修改为空值，所用命令为“SET NULL”；其三，将受到影响的外键值修改为默认值，所用命令为“SET DEFAULT”；其四，当从表中存在相应的外键值时，不允许删除主表中的主键值对应的元组，此即“禁止”删除，所用命令为“NO ACTION”。不过，SQL Server 和 Sybase 也只实现了其中的两种，即“CASCADE”和“NO ACTION”。

3. 从表外键的操作对完整性的影响

如果对学生表(即从表)的班号做如下操作,看看会发生什么情况:

(1) 插入(INSERT)。假定要向学生表插入一学生<0901,高九,…,计0009>,由于要插入的外键值"计0009"不在班级表主键值的范围之内,因此,要插入的外键值"计0009"是非法数据,应拒绝此类插入。而如果插入的是<0901,高九,…,计0006>,则由于要插入的外键值"计0006"在班级表主键值的范围之内,因此,应接受此类插入。

所以插入从表的外键值时,要求插入的外键值应"参照"(REFERENCE)主表中的主键值。

(2) 修改(UPDATE)。假定要将学生"王五"的班号"计0006"改为"计0005",由于要修改的外键值"计0005"不在班级表主键值的范围之内,因此,要修改的外键值"计0005"是非法数据,应拒绝此类修改。而如果是改为"计0002",则由于该外键值"计0002"在班级表主键值的范围之内,因此,应接受此类修改。

所以修改从表的外键值时,要求修改的外键值应"参照"主表中的主键值。

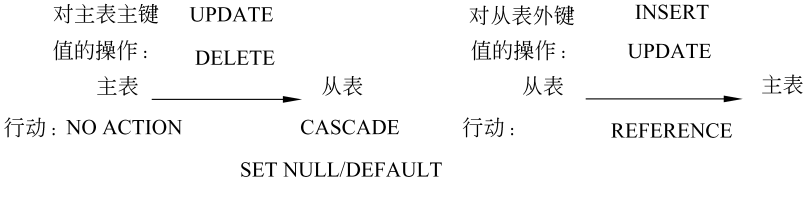
(3) 删除(DELETE)。如果要删除学生表中"王五"的信息,不需要参照主表中的主键值。

因此,从表中元组的删除不需要参照主表中的主键值。

综上所述,维护表间的完整性实际上是从以下两个方向完成的。

① 主表→从表。表示主表中的主键值在修改和删除时,从表中与该主键值相同的外键值可级联(CASCADE)修改和删除,或改为空值(SET NULL)或默认值(SET DEFAULT),或禁止(NO ACTION)主表主键值的修改和删除,如图3-3(a)所示。

② 从表→主表。表示从表中的外键值在插入和修改时,其值应参照(REFERENCE)主表中的主键值,如图3-3(b)所示。



(a) 主表→从表方向的数据完整性　　(b) 从表→主表方向的数据完整性

图3-3　表间完整性的维护

实现表间数据完整性的维护,可有以下两种方式。

① 利用外键约束定义,即在从表上定义外键约束,来完成主表和从表间两个方向的数据完整性。一般,现有的商用RDBMS都支持这种方式。

② 利用触发器(TRIGGER)完成两表间数据完整性的维护,即主表的触发器维护主表到从表方向的数据完整性,而从表的触发器维护从表到主表方向的参照完整性。关于触发器的使用,将在后续内容介绍。

说明：

① 两种实现方式，可任选其一。且不论哪种方式，均应按图 3-3 所示作两个方向的完整性维护。

② 在表间数据完整性维护方面，SQL-92 标准只推荐外键约束方式，未定义触发器方式。因此，在决定使用触发器方式时，应查阅所用 RDBMS 技术资料，看其是否支持利用触发器来实现表间数据完整性。

③ 正是由于 SQL-92 未定义触发器的标准语法，因此许多实现了触发器的 RDBMS，其触发器的语法可能相差比较大，在学习和使用各个 RDBMS 的触发器时应注意这一点。

外键约束需要显式定义。具体定义方法，在后面介绍。

说明：

① 要定义外键约束的一个或多个列，必须在另一张表中定义成主键约束或唯一约束，以符合外键定义的条件。

② 应在从表上定义外键约束。

3.3.6　一般性约束

域约束、主键约束、唯一约束和外键约束是关系数据模型中最基本的约束，也是 SQL-92 标准推荐的几种完整性约束，大多数商用 RDBMS 产品均支持这 4 种约束。除此之外，还有更一般的约束，如检查约束(check constraint)和断言(assertion)。

说明：如果说主键约束、唯一约束和外键约束分别是保证表中 3 个(或组)特殊列(主键、候选键和外键)的完整性，那么一般性约束则主要是保证表中其他一般性列的完整性。

检查约束，有时亦称表约束(table constraint)或表限制，可用来检查：

(1) 表中某一列的值是否在某一取值范围之内；

(2) 表中某几列之间是否满足指定的条件。

断言可检查表中个别列、整个表、或表与表之间是否满足指定的条件。

一般，表约束用于单个表的检查，而断言用于对多个表的检查。

说明：

① 表约束和断言，均应显式定义。

② 不像主键约束和唯一约束的定义，定义表约束和断言时，条件应显式指出，因为 RDBMS 预先不知道它们的条件。

③ 断言是 SQL-92 标准推荐的，但不是所有的商用 RDBMS 都支持，例如 SQL Server 和 Sybase 就不支持断言。因此，在决定使用断言之前，应查阅所用 RDBMS 技术资料，看其是否支持断言。

④ 由③可知，SQL-92 标准推荐的标准，商用产品不一定支持；而有时商用产品中具有的功能，在 SQL-92 标准中也不一定有。例如，SQL Server、Oracle 等支持触发器，而 SQL-92 标准中没有触发器定义。不过，在 SQL 1999 标准中增加了对触发器的定义。

3.3.7　完整性约束的实施

关系创建并指定了完整性约束后，当关系更新(包括插入、删除和修改，以后凡是谈到更新操作时，均是指这 3 种操作，除非特别说明)时，由 RDBMS 对该关系实施完整性约束

的检查。由于各个完整性约束对关系中数据影响的不同，以及条件复杂性程度的不一样，因此，RDBMS对各完整性约束的实施也有所不同。

1. 对域约束、主键约束和唯一约束的实施

这3种完整性约束，由于影响直接、条件简单，故只要插入、删除和修改的SQL命令，在执行过程中违背了这些约束，就会立即被RDBMS拒绝执行，并返回被违背的约束名称及其他信息。

2. 外键约束的实施

从前面的介绍可知，外键约束包含两个方向的内容，影响较复杂。因此，RDBMS对外键约束的检查，也是从两个方向分别进行的，但不是同时进行。

尽管外键约束只是在从表上定义，根据前面对外键约束的讨论可知，主表中的主键值在被删除和修改时，可能会影响到从表中的外键值。因此，当主表中的主键值在被删除和修改时，RDBMS就会检查这种主键值的变化是否影响到了从表中的外键值。如果是，则会根据预先设定的行动策略（如NO ACTION或CASCADE），进行相应的动作；否则，RDBMS将接受主表中主键值的变化。

而对从表，RDBMS会在从表中的外键值进行插入或修改时，检查该外键值的变化是否在主表的主键值范围之内。如果是，RDBMS将接受从表中外键值的变化；否则，将拒绝对应的SQL语句的执行，并返回该外键约束的名称及相关信息。

说明：

① 外键约束一旦设定，与该约束有关的主表和从表，都会成为RDBMS检查的目标。

② 对从表，外键值的删除不会触发RDBMS进行表间完整性检查，但插入和修改会。

③ 对主表，主键值的插入不会触发RDBMS进行表间完整性检查，但删除和修改会。

3. 一般性约束的实施

对一般性约束的实施通常是在每个SQL语句之后，由RDBMS根据预先设定的条件，对条件中指定的数据进行检查。

3.4 SQL Server和Sybase支持的完整性约束及其设定

前面介绍的是SQL-92中推荐的关系模型上的一些完整性约束。本节将以MS SQL Server及Sybase产品为例，来介绍这两个具体的数据库产品中所支持的完整性约束及其设定方法。

3.4.1 SQL Server和Sybase支持的完整性约束

SQL Server和Sybase数据库产品支持的完整性约束相同，如表3-1所示。

下面，将分别就表3-1中的7种完整性约束的设定方式进行介绍。其中，会逐渐涉及一些简单实用的SQL命令，读者可在实验环境下实际运行，体验其实际效果，以加深对关系模型完整性约束的理解，并掌握其使用方法。

表 3-1　SQL Server 和 Sybase 支持的完整性约束

完整性分类	实 现 方 式	含　义	备　　注
表本身的完整性	DEFAULT	默认	指定列的默认值
	RULE	规则	指定列的取值范围
	CHECK CONSTRAINT	检查约束	均有列级(即只涉及一列)和表级(涉及表中多列)两种写法
	PRIMARY KEY	主键约束	
	UNIQUE	唯一约束	
表间的完整性	FOREIGN KEY	外键约束	外键约束亦称参照完整性约束
	TRIGGER	触发器	可利用触发器来维护表间数据完整性

3.4.2　DEFAULT 的设定

DEFAULT 主要是用来为列或用户自定义数据类型指定默认值。每一个列或自定义类型只能有一个默认值。

作用及效果：当用户没有给指定有默认值的列输入数据时，RDBMS 自动用该默认值代替。

对于默认，有两种设定方式，分别是表定义时设定方式和创建默认的设定方式。

1. 表定义时设定 DEFAULT

该方式是在创建表的同时，对需要默认值的列设定其相应的默认值。

例 3-2　表定义时的默认设定示例。

```
CREATE TABLE publishers
( pub_id          char(4)          NOT NULL,
  pub_name        varchar(40)      NULL,
  city            varchar(20)      DEFAULT 'Pasadena',
  state           char(2)          DEFAULT 'CA')
```

例 3-2 为表 publishers 创建了 4 个列，分别是 pub_id、pub_name、city 和 state。其中，pub_id 和 state 是长度分别为 4 和 2 的字符型，pub_name 和 city 是长度分别为 40 和 20 的可变长度字符型，关于 SQL Server 中可用的系统数据类型请参见第 4.4.2 小节。另外，pub_id 的值必须为非空(NOT NULL)，而 pub_name 的值可为空值(NULL)。该 SQL 语句将 city 和 state 列的默认值分别设定为“Pasadena”和“CA”，其中的 DEFAULT 是命令关键字，不能少。

说明：

① CREATE TABLE 是 CREATE 中最常用的命令之一，用来创建表的结构，包括各个属性的名称、数据类型、数据长度及其他特性，如各种完整性约束等。

② 在 SQL Server 中，对字符型数据须用单引号(' ')括起。

③ publishers 是 SQL Server 样例库 pubs 中的表，关于 SQL Server 样例库表结构请参见附录。

验证：读者可利用例3-3中的2条SQL语句，来验证默认的作用与效果。

例3-3 默认作用的验证示例。

```
INSERT publishers (pub_id) VALUES ('0001')
SELECT * FROM publishers
```

其中，第一条语句用来向表publishers，插入一条记录，且只给出了pub_id的值为0001，其他的列均未赋值。第二条语句用来查询publishers表中现在有哪些记录，查询的结果中应该有＜0001，null，Pasadena，CA＞。如果是，则说明指定的两个默认值都起作用了。

说明：

① INSERT是SQL数据操纵中的插入命令，VALUES为关键字，其后的部分是给对应的各列赋值。如果在其前面没有列出要赋值的列，则其后的部分应按表中列的顺序，给所有的列赋值。

② SELECT是SQL中的数据查询命令，主要由一些子句（clause）构成，各子句由一关键字及其相应的表达式组成，例如，SELECT子句由SELECT及其后的各个属性构成（如用*，则表示表中所有列），FROM子句由FROM关键字及其后的一个或多个表构成，还有WHERE条件子句，它是由WHERE关键字及其后的条件表达式构成。这3个子句构成最基本的SELECT查询命令。

2. 创建DEFAULT

当多个表中的列其默认值相同时，这种设定方式很有用。其创建命令如下：

```
CREATE DEFAULT 默认名 AS '默认值'
```

要使该默认值起作用，必须将其绑定到表中相应的列上，绑定方法如下：

```
sp_bindefault '默认名', '表名.列名'
```

其中，sp_bindefault为SQL Server中的系统存储过程（stored procedure），表示将默认名所代表的默认值绑定（bind）到表中某个列上。这类存储过程可以直接在交互方式下使用。关于存储过程，请参见第4.4.6小节。

因此，这种设定方式下的默认设定，需要以上两步来完成。

例3-4 创建默认示例。先按例3-2所示，创建表publishers，但不为state列指定默认值。

```
CREATE DEFAULT dft_state AS 'CA'
sp_bindefault 'dft_state', 'publishers.state'
```

读者可自行按例3-3的方法，来验证此种方式下设定的默认。

说明：

① 绑定时，要求绑定的列名数据类型与默认值相同；

② 绑定了默认值后，不会对绑定默认值之前表中已存在的值产生影响，而只对绑定之后的值产生影响。

要取消某列的默认，可用 sp_unbindefault 解除绑定：

```
sp_unbindefault '表名.列名'
```

要删除默认，可用如下命令：

```
DROP DEFAULT 默认名
```

说明：应保证该默认已从所有绑定的列上摘除，否则删除不会成功。

3.4.3　RULE 的设定

RULE 主要是针对表中的某一列，指明该列的取值范围。

作用及效果：当该列值变化时，RDBMS 将检查变化的值是否在该规则规定的范围内。如是，则接受新列值；否则，拒绝该列值的改变，并返回该列值违反的规则名称及相关信息。

对于规则，只有一种设定方式，即创建规则的设定方式。其创建命令如下：

```
CREATE RULE 规则名 AS 规则
```

例 3-5　创建规则示例。

```
CREATE RULE state_rule AS @state IN ('CA', 'CO', 'WA')
```

说明：

① 规则可用 IN(…)、BETWEEN…AND…，关系式＜、＞、＜＝、＞＝、＜＞、＝、!＝、!＞、!＜和 LIKE 等操作符描述，各操作符的具体内容参见第 4 章；

② 创建规则时，应注意 AS 后有一个以@开头的局部变量，如例 3-5 中的@state 即为局部变量。

要使该规则起作用，必须将其绑定到表中相应的列上，绑定方法如下：

```
sp_bindrule  规则名, '表名.列名'
```

其中，sp_bindrule 也是 SQL Server 中的系统存储过程，表示将规则名所代表的规则绑定到表中某个列上。

例 3-6　绑定规则示例。先按例 3-2 所示，创建表 publishers，但不为 state 列指定默认值。再按例 3-5 创建规则 state_rule，然后用如下方式绑定规则。

```
sp_bindrule state_rule, 'publishers.state'
```

验证：读者可利用例 3-7 中的 SQL 语句，来验证规则的作用与效果。

例 3-7　规则作用的验证示例。语句如下：

```
INSERT publishers VALUES ('0002', 'WWWU Press', 'San Francisco', 'IL')
```

由于该 SQL 语句向表 publishers 插入的记录中，state 列的值“IL”，不在该列所绑定规则规定的范围之内，因此，SQL Server 会返回错误信息，拒绝本记录的插入。如果将例 3-7 中的“IL”改为“CA”，则该语句可成功执行，该条记录即可出现于 publishers 表中。

要取消某列的规则，可用 sp_unbindrule 解除绑定：

```
sp_unbindrule '表名.列名'
```

要删除创建的规则，可用：

```
DROP RULE 规则名
```

扩展用法：将创建好的默认和规则绑定到用户自定义类型上，方法分别如下：

```
sp_bindrule       规则名   用户自定义类型
sp_bindefault     默认名   用户自定义类型
```

要求：先应创建好用户自定义类型，然后再绑定。关于“用户自定义类型”的内容，请参见第 4.4.2 小节。

这样，只要使用该用户定义类型，其规则和默认即可发挥作用，从而简化了绑定规则和默认的操作。

要将默认和规则从用户定义类型摘除，则分别用如下存储过程：

```
sp_unbindefault   用户定义类型
sp_unbindrule     用户定义类型
```

查看创建规则和默认的过程，用如下存储过程：

```
sp_helptext   规则名或默认名
```

说明：

① 默认值必须满足规则的要求。

② 一般，在绑定一个新规则或默认时，应先摘除旧规则或旧默认。如没有摘除，则自动用新规则或新默认替换旧的。

3.4.4 检查约束的设定

检查约束类似于规则，要求用户对列或表中数据的更新应满足约束条件，条件也是用与规则类似的 IN、BETWEEN…AND 或 LIKE 等操作符来表达。

分类：分列级和表级两种定义方法。列级检查约束(column check constraint)针对表中一列，表级检查约束(table check constraint)则针对同一表中多列。

1. 列级检查约束

例 3-8 列级检查约束示例。

```
CREATE TABLE publishers1
(pub_id     char(4)       NOT NULL
 CONSTRAINT pub_id_constraint
 CHECK (pub_id IN ('234', '3344', '564') OR pub_id LIKE '43[0-9][0-9]'),
 city       varchar(20)   NULL,
 state      char(2)       NULL)
```

其中，pub_id_constraint 为检查约束名，CONSTRAINT 和 CHECK 为定义检查约束的命

令关键字。“CONSTRAINT pub_id_constraint”可省略，即用户如果不命名定义的检查约束，这时，SQL Server 将自动为该检查约束给定一个随机生成的名字。由于该名字是 DBMS 随机生成的，用户不容易记忆，故建议用户最好还是自己为定义的检查约束起一个利于记忆的名字。

说明：列级方式，表示在要定义约束的列本身定义完后，紧接其后定义其约束。

2. 表级检查约束

例 3-9　表级检查约束示例。

```
CREATE     TABLE   discounts1
( discounttype   varchar(40)          NOT NULL,
  store_id       char(4)              NULL,
  lowqty         smallint             NULL,
  highqty        smallint             NULL,
  discount       float                NOT NULL,
  CONSTRAINT low_high_check
  CHECK (lowqty <= highqty))
```

说明：

① 表级约束方式，表示在表中所有的列都定义完后，再来定义所要的约束。

② 表级和列级仅仅表示两种写法，没有其他含义。列级检查约束可用表级方式写，表级约束也能用列级方式写。

③ 默认值须满足检查约束要求。

3.4.5　主键约束的设定

主键约束要求主键值不能出现空值，且所有的值唯一。在定义了主键约束后，系统自动为该表生成一个聚簇索引(clustered index)。关于聚簇索引的内容，请参见第 4.2.5 小节。

分类：分列级和表级两种定义方式。列级针对表中一列，而表级则针对同一表中多列。

1. 列级主键约束

例 3-10　列级主键约束示例。

```
CREATE TABLE publishers2
( pub_id         char(4)              PRIMARY KEY,
  pub_name       char(30),
  city           varchar(20)          NULL,
  state          char(2)              NULL)
```

其中，PRIMARY KEY 为定义主键约束的命令关键字，此种方式下，系统会自动为该主键约束生成一个随机的名称，并生成一个基于该主键的聚簇索引。

2. 表级主键约束

例 3-11　表级主键约束示例。

```
CREATE TABLE sales1
( stor_id        char(4)              NOT NULL,
  date           datetime             NOT NULL,
  ord_num        varchar(20)          NOT NULL,
  CONSTRAINT     pk_sales_constr
  PRIMARY KEY NONCLUSTERED (stor_id, ord_num))
```

其中，pk_sales_constr 为该主键约束的名称，NONCLUSTERED 表示非聚簇索引（nonclustered index）。默认情况下，系统会自动为该表生成一个基于主键的聚簇索引，但用户可利用命令关键字 NONCLUSTERED，将该索引改为非聚簇索引。

例 3-11 将 sales1 表中的(stor_id, ord_num)定义为主键，这是一个复合键。

3.4.6 唯一约束的设定

唯一约束主要是针对候选键的约束。在定义了唯一约束后，系统自动为该表生成一个非聚簇索引，当然在定义时可改成聚簇索引。不过，要注意：一张表只能有一个聚簇索引。

与主键约束之区别：所有值唯一，最多只能有一个空值；默认索引为非聚簇索引。

分类：分列级和表级两种。列级针对表中一列，表级则针对同一表中多列。

1. 列级唯一约束

例 3-12 列级唯一约束示例。

```
CREATE TABLE publishers3
( pub_id         char(4)              UNIQUE,
  pub_name       char(30))
```

其中，UNIQUE 为定义唯一约束的命令关键字。此种方式下，系统会自动为该唯一约束生成一个随机的名称，并生成一个基于该候选键的非聚簇索引。

2. 表级唯一约束

例 3-13 表级唯一约束示例。

```
CREATE TABLE sales2
( stor_id        char(4)              NOT NULL,
  ord_num        varchar(20)          NOT NULL,
  date           datetime             NOT NULL,
  CONSTRAINT uq_sales_constr
  UNIQUE CLUSTERED (stor_id, ord_num))
```

其中，uq_sales_constr 为该唯一约束的名称，CLUSTERED 指示系统为该表生成一个基于候选键的聚簇索引。默认情况下，系统会自动为该表生成一个基于候选键的非聚簇索引，但用户可利用命令关键字 CLUSTERED，将该索引改为聚簇索引。

例 3-13 将 sales2 表中的(stor_id, ord_num)定义为候选键。

3.4.7 外键约束的设定

外键约束是用来维护通过外键联系的主表和从表间、两个方向的数据完整性。定义

外键约束的列，必须是另一个表中的主键或候选键。

分类：分列级和表级两种。列级针对表中一列，表级则针对同一表中多列。

1. 列级外键约束

例 3-14　列级外键约束示例。

```
CREATE TABLE titles
( title_id       tid                PRIMARY KEY,
  title          varchar(4)         NULL,
  pub_id         char(4)            NULL
  CONSTRAINT pub_id_const
  REFERENCES publishers3 (pub_id)
  ON DELETE CASCADE
  ON UPDATE CASCADE,
  notes          varchar(23)        NULL)
```

其中，pub_id_const 为该外键约束的名称，tid 是一种用户自定义类型，其定义是长度为 6 的不能为空值的字符型。关于用户自定义类型，请参见第 4.4.2 小节。

说明：

① 例 3-14 定义的外键约束中，publishers3 表中的 pub_id 在例 3-12 中被定义成候选键，则 titles 中的 pub_id 即为外键，故 publishers3 为主表，titles 为从表。

② 既然 titles 为从表，故外键约束应在其上定义。

③ REFERENCES publishers3(pub_id)用于定义“从表→主表”方向的参照完整性，表示 titles 表中的 pub_id 在插入和修改时应参照(REFERENCE)publishers3 表中的 pub_id。

④ ON DELETE CASCADE 和 ON UPDATE CASCADE 用于定义“主表→从表”方向的完整性，定义为维护“主表→从表”方向的完整性而采取的行动，表示当主表 publishers3 中的 pub_id 在删除(DELETE)或修改(UPDATE)时，如果影响到从表，则应级联(CASCADE)删除从表 titles 中具有相同 pub_id 值所在的元组，或修改从表 titles 中的pub_id。

⑤ 级联删除 titles 中的 pub_id，实际上是删除 pub_id 所在的行。

⑥ 如果不定义行动，即不要 ON DELETE CASCADE 和 ON UPDATE CASCADE，SQL Server 的默认行动是 NO ACTION。

2. 表级外键约束

例 3-15　表级外键约束示例。

```
CREATE TABLE salesdetail
( stor_id        char(4)            NOT NULL,
  ord_num        varchar(20)        NOT NULL,
  title_id       tid                NOT NULL,
  qty            smallint           NOT NULL,
  discount       float              NOT NULL,
  CONSTRAINT sales_constr
  FOREIGN KEY (stor_id, ord_num)
```

```
REFERENCES sales2 (stor_id, ord_num)
ON DELETE CASCADE
ON UPDATE CASCADE,

CONSTRAINT titles_constr
FOREIGN KEY (title_id)
REFERENCES titles (title_id)
ON DELETE CASCADE
ON UPDATE CASCADE)
```

例3-15中，定义了两个外键约束。由表salesdetail中的列，可以看出，(stor_id，ord_num)这两列在例3-13中被定义成表sales2的候选键，所以(stor_id，ord_num)为salesdetail的外键。另外，title_id在例3-14中被定义成表titles的主键，所以title_id也是salesdetail的外键。因此，salesdetail有两个外键，于是，可定义两个外键约束。

与例3-14中的列级外键约束定义不同的是，本例中多了FOREIGN KEY所在的行。这也算是列级与表级的另一区别。因为在例3-14的列级方式中，外键约束是紧接着pub_id写的，这也就表明pub_id是外键。而在例3-15的表级方式中，外键约束是在所有的列都定义完后才写，因此必须通过命令关键字FOREIGN KEY显式指出所定义的外键。

说明：CASCADE和NO ACTION两种策略是MS SQL Server及Sybase中支持的。事实上，在SQL-92标准中定义有4种策略，分别是CASCADE、NO ACTION、SET NULL及SET DEFAULT。由此也说明，具体的数据库产品不一定完全支持SQL标准。这是在学习和使用数据库时应特别注意的。

3.4.8 触发器的定义

触发器是一种特殊的过程，它不带参数，不被用户和程序调用，只能由用户对DB中表的操作(即插入、删除和修改3种操作)触发。也就是说，它是由操作激发的过程。正因如此，可以利用触发器，来维护表间的数据一致性。本小节将重点介绍如何定义触发器来维护表间数据一致性。

说明：触发器不仅可用来维护表间数据一致性，还可用来完成更复杂的功能。

触发器分AFTER触发器和INSTEAD OF触发器两种。

AFTER触发器表示其执行是在触发它的操作(INSERT、UPDATE和DELETE)之后，如果触发的操作失败，则此触发器不会执行。该触发器只有在表上建立。每个触发操作可定义多个AFTER触发器，可用sp_settriggerorder指定第一个和最后一个AFTER触发器的触发顺序，其他的则不确定。

INSTEAD OF触发器可在表或视图上定义，每个触发操作只能定义一个INSTEAD OF触发器。INSTEAD OF触发器主要是用来替换触发的操作(如INSERT、UPDATE或DELETE)，即不执行触发操作的语句，而是执行此触发器。

可利用AFTER触发器来维护表间的数据一致性。具体做法是：主表和从表应分别建立各自的触发器，主表的触发器维护主表到从表方向的数据完整性，而从表的触发器维护从表到主表方向的参照完整性。当然，这里的主表和从表，不需要定义外键，只要有共

同的列即可，只是为区分和与前面叙述对应起见，仍沿用关于外键约束中的“主表和从表”、“主键和外键”的说法。

应用示例：例 3-2 中创建的 publishers 表中的 pub_id，也出现在例 3-14 中创建的 titles 表中。借助外键约束中的说法，publishers 为主表，而 titles 为从表。要维护两表间的完整性，有两种方法可供选用，一种方法是利用外键约束，此方法需要首先定义 publishers 表中的 pub_id 为主键或候选键，使 pub_id 成为 titles 表的外键，然后在 titles 表中定义外键约束；另一种方法就是分别建立 publishers 和 titles 表的触发器，来维护它们间的完整性。

触发器创建的语法如下：

```
CREATE TRIGGER 触发器名
ON {表名|视图名}
{FOR|AFTER|INSTEAD OF} {[INSERT] [,] [UPDATE] [,] [DELETE]}
AS
    SQL 语句块
RETURN
```

其中，“表名”指要建立触发器的表。FOR 是为与早期版本兼容而保留，其默认类型即是 AFTER 触发器。FOR 后列出的是触发该触发器的操作，可只列一个操作，也可列两个或 3 个，但最多只能列 3 个操作。如果把 INSERT、UPDATE 和 DELETE 这 3 个操作都列上，表示 3 个操作都可触发该触发器。“SQL 语句块”表示用户编写的一段 SQL 语句，要求该触发器被触发后应做的事情。

说明：

① 一旦某操作触发了某个触发器，系统就会将该操作与该操作触发的触发器，作为一个事务提交或回退。关于事务、提交和回退的内容，请参见第 5.4 节和第 5.5 节。

② 系统为触发器提供有两张表，即 inserted 表和 deleted 表，其表结构与定义该触发器的表的结构一致，以便于程序员在编程时引用，但其中的数据不允许被修改。当触发的操作是“插入”时，新数据也会写入 inserted 表中；当触发的操作是“删除”时，删除的数据会保存在 deleted 表中；而当触发的操作是“修改”时，会同时用到这两张表，系统会采取先“删除”后“写入”的方式，先将删除的数据保存在 deleted 表中，而将修改的新数据写入 inserted 表中，即“新”数据会放到 inserted 表中，而“老”数据放到 deleted 表中。inserted 表和 deleted 表存储于内存，一旦触发器工作完成，它们即被删除。

根据前面关于表间完整性维护的讨论可知，对主表中的主键值进行删除和修改时，可能会影响从表中的外键值，但插入不会影响从表；而对从表的外键值进行插入和修改时，需要参照主表中的主键值，但删除不需要参照主表。

也就是说，主表只需建立两个触发器，即“删除”和“修改”触发器，就能维护主表到从表方向的数据完整性；而从表也只需建立两个触发器，即“插入”和“修改”触发器，就能维护从表到主表方向的参照完整性。下面将通过示例，分别建立主表和从表所需的几个触发器，读者在实际项目的开发中，可借鉴这 4 种触发器的写法。

例 3-16　由删除操作触发的主表删除触发器示例。

```
CREATE TRIGGER pub_del
ON publishers
AFTER DELETE
AS
    IF @@rowcount = 0 RETURN
    DELETE titles
    FROM titles t, deleted d
    WHERE t.pub_id = d.pub_id
RETURN
```

其中，@@rowcount 是 SQL Server 提供的系统变量（或称全局变量），以@@开头，其值表示表中有几行记录被删除了。

说明：

① 该触发器的功能表示，当删除了 publishers 表中的数据后，级联删除 titles 表中对应的行，实际上是模拟了外键约束中的 CASCADE 行动。如果要模拟 No Action 行动，则可用 Rollback Transaction 替换 pub_del 触发器中的 DELETE 语句。该触发器中 deleted 表的结构与 publishers 表结构一致。事实上，由于触发器中的行动是由用户定义的，因此利用触发器可以完成更加灵活的表间数据一致性维护。而这是外键约束无法做到的，因为外键约束的行动是被 DBMS 固定的，例如在 SQL Server 中只有两个行动（CASCADE 和 NO ACTION）可供选用。

② DELETE 是 SQL 数据操纵中的数据删除命令，DELETE 引出要操纵的表，FROM 指明要引用的表，WHERE 指出条件。只有满足该条件的元组才会被删除。一般在删除命令中都会带有条件，否则将删除表中所有的元组。

③ 例 3-16 中的 t 和 d，分别作为 titles 和 deleted 的别名，以便于简化较长表名的书写。别名的另一个用途是当引用的两张表是同一张表时，可用别名的形式来区分。

④ SQL Server 中也提供有类似高级语言中的流程控制语句，如 IF、WHILE 等，相关内容参见第 4.4 节。

例 3-17 由修改操作触发的主表修改触发器示例。

```
CREATE TRIGGER pub_update
ON publishers
AFTER UPDATE
AS
    DECLARE @num_rows INT
    SELECT @num_rows =@@rowcount
    IF @num_rows =0 RETURN
    IF UPDATE (pub_id)
    BEGIN
        IF @num_rows >1
        BEGIN
            RAISERROR 53333   '不支持多个 pub_id 值的修改'
            ROLLBACK TRANSACTION
```