第 3 章

数据库设计和 E-R 模型

设计数据库之前，需要充分了解业务场景的关系模型，对数据对象进行规范化的设计，合理应用数据之间的关系和约束。本章介绍了关系代数中的各种运算，并结合实际的 SQL 例子来帮助理解对数据集的操作；结合 E-R 模型分析实体对象之间的联系，规范化对象之间的依赖、范式等，给出了通用的数据库设计流程；同时介绍了数据实体集中的数据元素之间的约束。通过本章内容的学习，用户能够去分析、构建一个完善的数据库系统。

3.1 关系代数

关系模型是数据库查询语言的基础。因此在了解数据库之前，需要首先了解数据库领域常用的关系代数运算符的概念。下面首先介绍关系代数的历史，然后逐一介绍常用的关系代数运算符的概念并结合具体示例介绍其用法，最后结合具体的 SQL 给出相关的查询语法。

3.1.1 关系代数的由来

在用户获取数据时，需要借助查询语言。查询语言通常分为两种：一种是过程性的，需要用户指定查询计划，计算机严格遵循查询计划获取数据；另一种是非过程性的，不需要用户指定查询计划即可直接返回最终结果，这种查询语言也可以理解为用户定义“做什么”而不是“如何做”。表 3-1 描述了两者的区别。

关系代数通过对关系的运算来表达查询。简单来说，关系代数的运算通过输入有限数量的关系进行运算，运算结果仍为关系，过程可以按照代数运算符的运算顺序进行计算，从而得到最终计算结果。关系演算与关系代数相反，是一种典型的非过程性

表 3-1　查询语言对比

分　类	查询语言	
	过程性(Procedural)	非过程性(Non-Procedural)
语言	3GL(Third-Generation Language)	4GL(Fourth-Generation Language)
需求： 显示 fileA 文件所有名称信息	Open fileA Do while . not. eof() ? name,total skip enddo	Open fileA List name,total

数据库查询语言，提供了查询数据库的申明性方式，其包括元组关系演算(Tuple Relational Calculus，TRC)和域关系演算(Domain Relational Calculus，DRC)两个部分。元组演算以元组为对象进行操作，取出关系的每一个元组进行运算，一个元组变量可能需要一个循环，从而使元组演算运算次数不可控，有安全风险。域关系演算以域变量为对象，取出域的每一个变量确定是否满足所需条件。总体来看，三者体现了三种不同的思维，可以根据不同的用途选取适合的语言。本节将详细介绍关系代数相关概念，包括关系代数运算符的原理及在 SQL 上的应用。

3.1.2　关系代数运算符

关系(一张表)本质上是元组(表中的每行，即数据库中每条记录)的集合，因此很多关系代数运算符跟集合常用操作非常相似。关系代数最基本的运算符有 7 个：选择(Selection)、投影(Projection)、笛卡儿积(Cartesian Product)、连接(Join)、除(Division)、关系并(Union)和关系差(Difference)。事实上这 7 个基本关系代数运算符可以满足关系完备性的要求。此外，还有一些扩展的关系运算符，如重命名(Rename)等。这些代数运算符的功能可以用前面介绍的基础运算符的组合来表示，但是因为这些运算符经常被用到，因而也同样被抽象成一个完整的代数运算符来使用。

为了更加形象地说明各关系代数运算符的含义，下面罗列了关系表示例，后续的关系操作将基于如下关系表示例展开。表 3-2 和表 3-3 为两个玩具的关系清单，表 3-4 为生产商当天全部产品处理折扣，以下是其中各列的说明：

- ToysName 表示玩具名；
- Price 表示玩具价格，单位为 $；

- Material 表示材质；
- Supplier 表示生产商；
- Discount 表示折扣。

表 3-2　关系：Toys1

ToysName	Price	Material
Bear	12	Rag
Tiger	7	Plastic
Fox	7	Plastic

表 3-3　关系：Toys2

ToysName	Price	Material
Rabbit	10	Rag
Dog	8	Plastic

表 3-4　关系：Supplier

Supplier	ToysName	Discount
JinFu	Rabbit	8
Dingsheng	Dog	8
Funrui	Bear	9

关系代数运算符如表 3-5 所示，下面逐一介绍各关系代数运算符的原理。

表 3-5　关系代数运算符

关系代数运算符类别	运　算　符	符号
基本关系	选择	σ
	投影	π
	笛卡儿积	$\times$
	连接	$\bowtie$
	除	$\div$
	关系并	$\cup$
	关系差	$-$
扩展关系	关系交	$\cap$
	重命名	ρ
	聚集	G

1. 选择

选择代数运算符在关系代数中通常用符号 σ 表示，用于在关系 R 中选择满足给定条件 $F(t)$ 的元组，公式表达为：

$$\sigma_F(R)=\{t \mid t \in R \cap F(t)='\text{true}'\}$$

F 是一个或多个逻辑表达式的组合。选择运算实际上是从关系 R 中选择满足条件 F 的元组。当关系 R 中所有元组均满足选择条件 F 时，返回值跟关系 R 完全相同；当关系 R 中所有元组均不满足选择条件 F 时，返回结果关系中的元组数为 0。

逻辑表达式 F 的基本形式为 $X\theta Y$。其中 X、Y 表示属性名，可以是常量、简单函数或者属性(关系中的列)名，也可以用其序号表示。θ 代表比较运算符，它可以是>，<，<=，>=，=或<>等基本运算符，同样也可以使用与(&&)、或(||)、非(!)等逻辑运算符进行逻辑运算。表 3-6 为选择示例。

表 3-6　示例：$\sigma_{\text{Price}<10}$(Toys1)

ToysName	Price	Material
Tiger	7	Plastic
Fox	7	Plastic

注：查询表达式为 Price<10，因此只有后两个元素满足条件。

2. 投影

投影代数运算符用符号 π 表示，符号 π 也是一元代数运算符(只需要一个操作数的运算符)。投影是对关系 R 的一种垂直切割，从关系 R 中取出一列或者多列，F 可以是一个属性列或者多个属性列组成的元组，取出的内容由 F 指定，投影返回的结果由 F 中所有属性组成，公式表达为：

$$\pi_F(R)=\{t[F] \mid t \in R\}$$

需要注意的是，投影之后不仅取消了原始关系中某些列，而且还可能取消掉一些元组，原因是取消某些关系列后可能出现重复行，违反了关系的定义。因此必须检查并去除结果关系中重复的元组。表 3-7 为投影示例。

表 3-7　示例：π_{Material}(Toys1)

Material
Rag
Plastic

注：投影属性列表由 Material 组成，不包括 Toys1 中其他属性，但是拥有行 Rag 和 Plastic，去重后得到上述结果。

3. 笛卡儿积

笛卡儿积代数运算符用符号×表示，又称叉积，表示两个操作关系 R 和 S 的元组之间所有可能的连接。笛卡儿积运算会将两个原始元组连接生成一个新的元组。另外，同一个关系中不应该存在相同的属性，否则结果关系中会出现相同的属性，违反了关系唯一性定义，计算结果属性唯一。公式定义如下：

$$R \times S = \{t_r t_S \mid t_r \in R \cap t_S \in S\}$$

表 3-8 为笛卡儿积示例。

表 3-8　示例：Toys1×Supplier

Toys1. Toys Name	Toys1. Price	Toys1. Material	Supplier. Supplier	Supplier. ToysName	Supplier. Discount
Bear	12	Rag	JinFu	Rabbit	8
Bear	12	Rag	Dingsheng	Dog	8
Bear	12	Rag	Funrui	Bear	9
Tiger	7	Plastic	JinFu	Rabbit	8
Tiger	7	Plastic	Dingsheng	Dog	8
Tiger	7	Plastic	Funrui	Bear	9
Fox	7	Plastic	JinFu	Rabbit	8
Fox	7	Plastic	Dingsheng	Dog	8
Fox	7	Plastic	Funrui	Bear	9

注：Toys1 和 Supplier 的笛卡儿积为两者所有可能排列的组合，因为关系中不应该存在相同的属性，即关系表不可以同属性名，对于相同的属性值 ToysName 使用其对应的关系和属性名的组合代替。

4. 连接

连接运算是从两个关系的笛卡儿积中选取属性间满足一定条件的元组形成一个新的关系。连接代数运算符是关系代数中很有用的关系代数运算符，有四种常用的连接，分别为等值连接(Equijoin)、自然连接(Natural Join)、半连接(Semi Join)跟反连接(Anti Join)。

连接条件用 θ 表示，连接公式定义如下：

$$R \underset{A\theta B}{\bowtie} S = \{\widehat{t_r t_s} \mid t_r \in R \cap t_S \in S \cap t_r[A]\theta t_S[B]\}$$

表 3-9 为连接示例。

表 3-9　θ 连接示例：Toys2 $\bowtie_{(Price<Discount)}$ Supplier

Toys2. ToysName	Toys2. Price	Toys2. Material	Supplier. Supplier	Toys2. ToysName	Supplier. Discount
Dog	8	Plastic	Funrui	Bear	9

上述连接也称为 θ 连接，这个条件为 $A\theta B$。

等值连接中 θ 为“=”，它是从关系 R 与关系 S 的笛卡儿积中选取 A、B 属性值相同的元组。也可以理解为如下关系式：

$$R \overset{\bowtie}{}_{A=B} S = \{\widehat{t_r t_s} \mid t_r \in R \cap t_S \in S \cap t_r[A] = t_S[B]\}$$

表 3-10 为等值连接示例。

表 3-10　等值连接示例：Toys2 $\bowtie_{(Price=Discount)}$ Supplier

Toys2. ToysName	Toys2. Price	Toys2. Material	Supplier. Supplier	Toys2. ToysName	Supplier. Discount
Dog	8	Plastic	JinFu	Rabbit	8
Dog	8	Plastic	Dingsheng	Dog	8

自然连接代数运算符用$\bowtie$表示，它是一种特殊的等值连接。它要求两个关系中进行比较的分量必须是同名的属性组，并在结果中去除重复的元素。即若 R 与 S 中具有相同的属性组 B，U 为 R 跟 S 的全体属性集合，则自然连接可以记作：

$$R \bowtie S = \{\widehat{t_r t_s}[U-B] \mid t_r \in R \cap t_S \in S \cap t_r[B] = t_S[B]\}$$

表 3-11 为自然连接示例。

表 3-11　自然连接示例：Toys2 $\bowtie$ Supplier

Price	Material	ToysName	Supplier	Discount
10	Rag	Rabbit	JinFu	8
8	Plastic	Dog	Dingsheng	8

此外，还存在其他两种连接：半连接（运算符为$\ltimes$和$\rtimes$）及反连接（运算符为$\triangleright$）。半连接类似于自然连接，常用符号$\ltimes$和$\rtimes$表示，跟自然连接主要的区别是决定需要显示哪些列。反连接，可以用 $R \triangleright S$ 来表示，类似于半连接，区别是反连接结果只能是 R 中有、S 中没有的元组，它们的公共属性相同。

表 3-12 为左半连接示例。

表 3-13 为右半连接示例。

表 3-14 为反连接示例。

表 3-12　左半连接示例：Toys1 ⋉ Supplier

ToysName	Price	Material
Bear	12	Rag

注：Toys1 和 Supplier 的左半连接类似于自然连接，只显示关系 Toys1 和关系 Supplier 属性列 ToysName 相同的组合，属性列只显示关系 Toys1 所包含的属性 ToysName、Price 及 Material。

表 3-13　右半连接示例：Toys1 ⋊ Supplier

Supplier	ToysName	Discount
Funrui	Bear	9

注：Toys1 和 Supplier 的右半连接类似于左半连接，区别为属性列只显示关系 Supplier 所包含的属性 Supplier、ToysName 及 Discount。

表 3-14　反连接示例：Toys1 ▷ Supplier

ToysName	Price	Material
Tiger	7	Plastic
Fox	7	Plastic

注：反连接，属性列只显示关系 Toys1 所包含的属性 ToysName、Price 及 Material，属性值只显示存在于关系 Toys1 中的且关系 Supplier 中没有的元组。

5. 除

假设存在两个关系 $R(X,Y)$ 和 $S(Y)$，其中 X，Y 为属性组。R 中的 Y 与 S 中的 Y 可以有不同的属性名，但出自相同集合。R 与 S 除运算得到新的关系，新关系中的元组是 R 中满足下列条件的元组在 X 属性列上的投影：元组在 X 分量值 x 的像集 Y_x 包含 S 在 Y 上投影的集合，其中 $x=t_i[X]$。可以理解为如下关系式：

$$R \div S = \{t_i[X] \mid t_i \in R \ \wedge \ \pi_Y(S) \subseteq Y_x\}$$

$R \div S$ 的关系如图 3-1 所示。

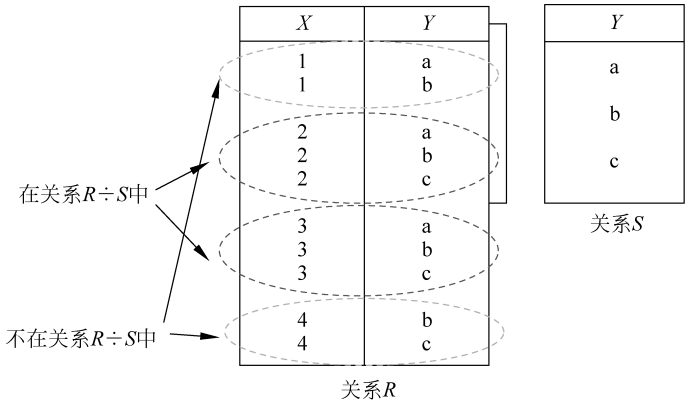

图 3-1　$R \div S$ 除运算关系图

$R \div S$ 示例如表 3-15 所示。

表 3-15　示例：$R \div S$

X
2
3

注：除运算，元素 2 与 3 在 X 列的像集包含关系 S 在 Y 列上投影的集合，而元素 1 与元素 4 不满足这个关系，因而不在新的关系列中。

6. 关系并代数运算符

关系并操作要求两个关系相互兼容，也可以说是具有相同的属性。R 跟 S 相互兼容时，并操作定义如下：

$$R \cup S = \{t \mid t \in R \cup t \in S\}$$

结果关系是由关系 R 与关系 S 共同组成。在 R 跟 S 没有元组重复的情况下，结果元组数的最大值为 R 与 S 元组数之和。重复结果需要进行去重。表 3-16 为关系并操作示例。

表 3-16　示例：Toys1∪Toys2

ToysName	Price	Material
Bear	12	Rag
Tiger	7	Plastic
Fox	7	Plastic
Rabbit	10	Rag
Dog	8	Plastic

7. 关系差

关系差是传统运算之一，用符号－表示。关系差操作跟关系并操作类似，要求关系兼容。计算结果为属于关系 R 而不属于关系 S 的属性元组。公式表示如下：

$$R - S = \{t \mid t \in R \cap t \notin S\}$$

表 3-17 为关系差操作示例。

表 3-17　示例：Toys1－Toys2

ToysName	Price	Material
Bear	12	Rag
Tiger	7	Plastic
Fox	7	Plastic

注：关系 Toys1 与关系 Toys2 无交集，因而两者关系差为 Toys1 本身。

8. 关系交代数运算符

关系交操作与关系并操作类似，同样需要关系具备相同的属性。其公式表示如下：

$$R \cap S = \{t \mid t \in R \cap t \in S\}$$

关系交操作可以通过关系差运算表示：$R \cap S = R - (R - S)$。表 3-18 为关系交操作示例。

表 3-18　示例：Toys1 ∩ Toys2

ToysName	Price	Material

注：由于 Toys1 跟 Toys2 无共同的元素，所以两者相交值为空。

9. 重命名代数运算符

重命名代数运算符用符号 ρ 表示，常常起辅助性作用。当需要连接新的关系 S 时，它又与原始关系组 R 的属性名完全相同，即使含义不同，也会在连接时造成一定困扰。此时可以通过重命名代数运算符进行属性名修改。公式表示如下：

$$\rho_Q(y_1, y_2, \cdots)(S)$$

表达式含义为，将关系 S 重命名为关系 Q，其对应属性名称修改为 $y_1, y_2, \cdots$ 等新名称。表 3-19 为重命名操作示例。

表 3-19　示例：ρ_{Toys3}(NewName, NewPrice, NewMaterial)(Toys2)，新表名称为 Toys3

NewName	NewPrice	NewMaterial
Rabbit	10	Rag
Dog	8	Plastic

10. 聚集代数运算符

聚集操作通常可以采用以下公式表达：

$$G_1, \cdots, G_i F_1(X_1), \cdots, F_i(X_i)(R)$$

关系 R 中数据被分为 i 组，分组属性分别为 $G_1, \cdots, G_i$，X_i 为一个属性名。其中 $F_1(X_1), \cdots, F_i(X_i)$ 为属性分组对应的关系表达式。划分规则为：同一组中所有元组在 $G_1, G_2, \cdots, G_n$ 上的值相同；不同组中元素在 $G_1, G_2, \cdots, G_n$ 上的值不同。表 3-20 为聚集操作示例。

表 3-20　示例：Toys2. ToysName，Material sum(Price)(Toys2×Supplier)

Toys2. ToysName	Material	sum(Price)
Rabbit	Rag	30
Dog	Plastic	24

注：对 Toys2×Supplier 的关系表做聚集运算，对于属性列 Toys2. ToysName、Material 相同的部分元素求取 Price 的和，生成一个新的关系表。

3.1.3　关系代数与 SQL 的转换

SQL 允许用户在上层数据结构工作，不需要了解底层实现及数据存放方式。另外，SQL 语句允许一条 SQL 语句的输出作为另一条语句的输入，这赋予它很大的灵活性和强大的功能。不过需要注意的是，在关系代数集合中，每一个元组都是唯一确定的，不允许出现重复情况。因此，在计算一个关系代数表达式时，必须对查询结果进行去重工作。

下面列出几个关系代数操作转换为 SQL 语句的等价形式。为方便后续对 SQL 扩展语句的介绍，图 3-2 为 SQL 简单关系表示例。

x	y
a	3
c	2
b	5
a	1

(a) Relation1

x	y
a	3
b	2

(b) Relation2

y	z
3	5
2	6

(c) Relation3

图 3-2　SQL 操作关系表示例

1. 选择运算

格式如下：

$$\sigma_F(R)$$

选择运算的 SQL 语句如下：

```
SELECT * FROM(R) WHERE condition
```

SQL 常用的查询条件有：比较运算符(>、>=、<、<=、=、<>或!=)；范围查询(BETWEEN…AND)；集合查询(IN)；空值查询(IS NULL)；字符串匹配查询

(LIKE)；逻辑查询(AND、OR、NOT)。可利用上述条件进行扩展查询。

表 3-21 为选择运算示例。

表 3-21　示例：SELECT * FROM Relation1 WHERE $y>2$

x	y
a	3
b	5

2. 投影运算

格式如下：

$$\pi_A(R)$$

投影运算的 SQL 语句如下：

```
SELECT A FROM R
```

3. 关系并运算

格式如下：

$$R \cup S$$

关系并运算的 SQL 语句如下：

```
SELECT * FROM R UNION SELECT * FROM S
```

SQL 语句中，UNION 代数运算符对应的常用语句有：UNION 运算用于去掉重复的元素，UNION ALL 运算不去重。其示例如表 3-22 和表 3-23 所示。

表 3-22　示例：SELECT * FROM Relation1 UNION ALL SELECT * FROM Relation2

x	y
a	3
c	2
b	5
a	3
b	2

表 3-23　示例：SELECT * FROM Relation1 UNION SELECT * FROM Relation2

x	y
a	3
c	2
b	5
b	2

4. 关系交运算

格式如下：

$$R \cap S$$

关系交运算的 SQL 语句如下：

```
SELECT * FROM R INTERSECT SELECT * FROM S
```

SQL 语句中 INTERSECT 跟 UNION 类似，同样存在 INTERSECT ALL 运算。INTERSECT 用于剔除重复行，INTERSECT ALL 运算不去重。

5. 关系差运算

格式如下：

$$R - S$$

关系差运算的 SQL 语句如下：

```
SELECT * FROM R EXCEPT SELECT * FROM S
```

6. 连接运算

格式如下：

$$R \bowtie S$$

连接运算的 SQL 语句如下：

```
SELECT * FROM R NATURAL JOIN S
```

SQL 连接代数运算符包括连接类型跟连接条件两部分。

连接类型分为内连接(Inner Join)和外连接(Outer Join)。内连接对应等值连接(Equijoin)，外连接分别对应左外连接(Left Outer Join)、右外连接(Right Outer Join)及完全外连接(Full Outer Join)等。此外，还存在半连接(Semi Join)及反连接(Anti Join)。

连接条件决定了哪些元组应该被匹配，决定了连接结果中出现哪些属性。连接条件放在连接类型右边，常用的连接条件有 NATURAL、ON、USING(A1，A2，…)。其示例如表 3-24 所示。

7. 笛卡儿积运算

格式如下：

$$R \times S$$

表 3-24　示例：SELECT Relation1. x Relation3. z AS z FROM Relation1 FULL JOIN Relation3 ON Relation1. y = Relation3. y

x	z
a	3
c	2
b	NULL
a	NULL

注：预期用 Relation1 中 x 列的值和 Relation3 中 z 列的值组成一个新的关系表。Relation3 中的 z 属性列在表中名称为 z，满足 Relation1. y = Relation3. y 条件的部分数据将填充在 z 属性列中。

笛卡儿积运算的 SQL 语句如下：

```
SELECT * FROM R,S
```

8. 重命名运算

格式如下：

$$\rho_R(\mathrm{A1},\mathrm{A2},\cdots)(S)$$

重命名运算的 SQL 语句如下：

```
SELECT * FROM S AS R(A1,A2,…)
```

9. 聚集操作

格式如下：

$$G_1 G_2,\cdots,G_n F_1(A_1),F_2(A_2),\cdots,F_m(A_m)(E)$$

聚集运算的 SQL 语句如下：

```
SELECT G1 G2,…,GnF1(A1),F2(A2),…,Fm(Am) FROM E GROUP BY G1 G2,…,Gn
```

SQL 聚集操作示例如表 3-25 所示。另外可以使用 HAVING 子句，删除不满足

HAVING 条件的部分数据。

表 3-25　示例：SELECT x,sum(y) FROM Relation1 GROUP BY x

x	sum
a	4
c	2
b	5

注：新表属性由 Relation1 中 x 列的值与 sum(y)共同组成，Relation1 表 x 列存在两个完全相同的元素 a，其 y 列 sum 和为 4，新生成元组与其他数据生成一个新的关系表。

3.2　数据库设计

数据库设计(Database Design)指基于某一具体的数据库管理系统(Database Management System，DBMS)设计并实现数据库实例内具体数据对象，例如表 table、视图 view、索引 index 的过程。DBMS 是位于用户和操作系统之间的一层系统级软件，主要负责数据的定义、组织、存储、管理、操纵及数据库相关业务功能的实现。

数据库设计本身是一项复杂且反复的过程，需要设计者对数据对象及数据对象间的关系进行优化设计和反复推敲。

3.2.1　数据库设计概述

随着大数据时代爆炸式的数据增长，软件、硬件设备快速更新迭代，高效且可靠的数据组织管理模式日趋重要。数据库作为有组织、低冗余、独立、易扩展且可共享的数据存储及管理系统，在大数据时代下的重要性愈发凸显。

数据库设计的完整定义包含：数据库逻辑模式和物理结构的设计、数据库和相应应用系统的建立以及数据的存储和管理。数据库设计的最终目标是为用户提供高效的数据存储效率以及管理模式。

3.2.2　数据库设计的特征

数据库设计的主要特征体现为以下四点：

(1) 技术复杂：需要软件设计、硬件设计与应用技术的综合性设计能力。

(2) 有效的管理模式：基于数据库建设项目的管理设计方式。

(3) 基于基础数据构建：基础数据结构与特征决定数据库设计导向。

(4) 与应用系统紧耦合：设计过程需紧密结合上层应用特性。

事实上，数据库设计不仅仅局限于计算机科学技术，它是一项涉及多种交叉学科的技术内容，设计者需要充分了解上层应用的学科知识。例如，设计一款面向天文科学大数据的数据库，需要同时对数据库软件设计理论、底层存储设备特性，以及天文科学大数据的采集、构建、存储特征，都有较好的认识和了解。

就管理模式而言，要设计出一个高效的数据库应用系统，有效的管理模式比开发技术更加重要。除了对数据库设计工程本身的项目管理，也包含了侧面影响数据库设计的业务部门管理。

基础数据的收集、整理、组织和不断更新是数据库设计中的重要环节，也是数据库建立初期最为烦琐、细致的工作。在数据库设计阶段，需要明确其面向的是何种意义、何种结构的数据，并进行数据收集和整理；在数据库运行阶段，需要组织和不断更新数据库中的数据，并基于基础数据对数据库的功能和性能进行测试；在数据库迭代更新、优化过程中，需要采集更多的基础数据，进行更为全面的功能和性能测试。由此可见，基础数据是数据库存在的核心原因，所以基础数据在数据库设计中具有核心地位。

数据库作为面向上层应用的数据组织管理单元，它的设计和上层应用系统的设计是密不可分的，具有紧耦合性。数据库设计过程中要将数据库结构设计和上层应用中的数据处理模式设计有效结合起来，这也是数据库设计的重要特征之一。

3.2.3　实体联系模型：E-R 模型

概念模型就是一种信息结构，用于现实世界到信息世界的抽象化描述。E-R 模型(Entry-Relationship Model)，即实体联系模型或实体关系模型，是用来描述现实世界概念模型的有力工具，它是由美籍华裔计算机科学家陈品山(P. P. S. Chen)提出的一种实体联系模型，通过实体、属性、联系三元组表达概念模型。在现实世界中，事物内部及事物之间都存在着一定的联系，实体内部的联系通常指组成实体的各属性之间的联系，实体之间的联系通常指不同实体的实体集间的联系。实体之间的联系中，把参与实体联系的实体数目称为联系的度，两个实体之间的联系度为 2，也称为二元联系；三个实体之间的联系度为 3，称为三元联系；N 个实体之间的联系度为 N，也称为 N 元联系。无论是不同实体间还是实体内部，联系都分为一对一联系、一对多联系、多对

多联系，具体阐述如表 3-26 所示。

表 3-26　实体联系类型

关　　系	符　　号
一对一	1∶1
一对多	1∶n
多对多	m∶n

E-R 模型基于 E-R 图来描述上述多样化的实体联系，即概念模型。E-R 图采用不同的几何形状——矩形、椭圆形以及菱形分别表示实体、属性和联系，具体说明如下。

(1) 矩形：表示实体，矩形框内写明实体名称。

(2) 椭圆形：表示属性，用无向边将其与对应的实体连接起来。

(3) 菱形：表示联系，菱形框内写明联系名，用无向边分别与有关实体连接起来，同时在无向边旁边标注联系的类型(1∶1、1∶n、m∶n)。如果联系具有属性，则相应属性需要用无向边与该联系连接起来。

3.2.4　数据库设计流程

数据库设计作为完整的结构化系统构建流程，依照结构化系统设计方法，分为以下六个阶段，各阶段的执行内容与关联关系如图 3-3 所示。

(1) 需求分析(Requirement-Analysis)；

(2) 概念结构设计(Conceptual Design)；

(3) 逻辑结构设计(Logical Design)；

(4) 物理结构设计(Physical Design)；

(5) 数据库实施(Database Implementation)；

(6) 数据库运行和维护(Database Running and Maintenance)。

下面将以 TPC-C(TPC 为事务处理性能委员会的简称)标准测试用例(商品批发销售关系)为例，按照数据库设计流程：需求分析、概念结构设计、逻辑结构设计、物理结构设计、数据库实施、数据库运行和维护，分别介绍各流程中数据库设计的核心内容与实现方式。

1. 需求分析

需求分析是任何系统设计过程中不可或缺的内容。同样的，数据库设计首先必须

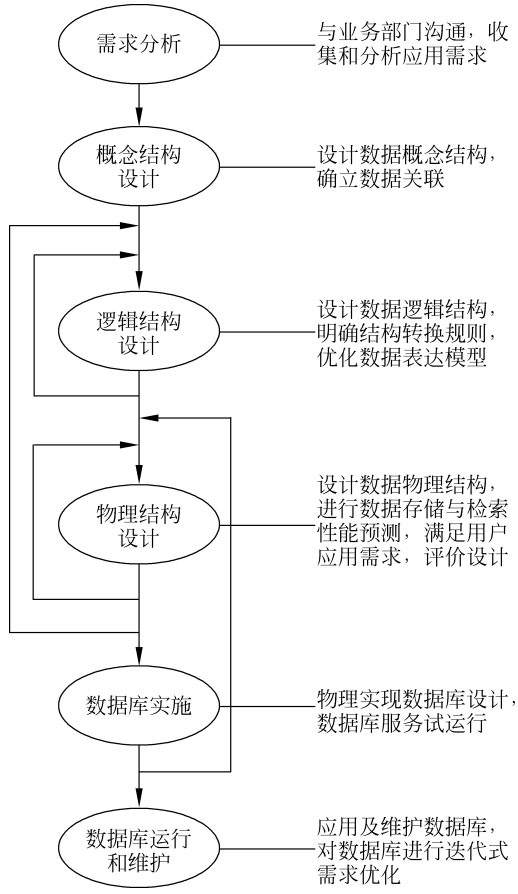

图 3-3　数据库设计流程及内容

了解并分析用户需求，包括数据特征与处理方式。需求分析是整个数据库设计过程的基础，也是最为困难与耗时的一步，它的质量直接决定了整个数据库系统的可用性与质量。需求分析本质内容是与用户交互，获取用户期望目标，而与用户交互获取用户期望的主要方式就是调查。调查通过与用户座谈、请用户填写调查问卷、跟班作业等方式，才能获取用户的本质需求。调查清楚用户的需求后，便需要通过分析方法进一步分析用户需求。分析方案是多样化的，在众多分析方法中，SA(Structured Analysis，结构化分析)方法是一种较为简单的方法。SA 方法于 20 世纪 80 年代起开始广为使用，其通过将系统概念转换为用数据及控制表示的数据流程图，进而描述数据在不同模块间流动的过程。SA 方法从最上层的组织结构入手，采用自顶向下、逐层分解的方式进行数据分析。

调查分析的重点是“数据”和“处理”，数据库设计者需通过调查、收集和分析获取用户对数据库的各项需求，包括信息、处理、安全性与完整性需求，而这个过程是一个逐步迭代的过程。

那实际过程中需求分析阶段到底要做什么呢？以 TPC-C 标准测试用例数据库的设计为例，应首先明确该大型商品批发公司面临怎样的数据存储与管理需求，包括其下属有多少个商品仓库，每个商品仓库与销售点的对应关系是怎样的，每个销售点与客户的对应关系是怎样的，客户的订单信息是如何构建的。除此之外，数据库设计者还要明确用户需要使用的数据管理功能接口特征，以及用户对于系统可用性（如可视化程度）、高效性（如查询响应时间）的具体需求。

2. 概念结构设计

概念结构设计需要对用户需求进行综合、归纳与抽象，形成一个独立于数据库系统的概念模型，而概念模型的主要描述方式就是采用 E-R 模型，E-R 模型可以简洁、清晰地描述出实体、属性与联系间的关系。

以 TPC-C 标准测试用例数据库场景为例，在一组关系模式中具有仓库、管理员、地区、商品四个实体，仓库具有容量属性、管理员具有工号属性、地区具有邮编属性、商品具有价格属性。当一个货物仓库只有一个管理员，一个管理员只对一个仓库负责时，仓库管理员和仓库间的联系就是管理，它们间的联系类型就是 1∶1。当一个地区有多个仓库，一个仓库只可能位于一个地区时，它们之间的联系就是位于，它们间的联系类型就是 1∶n。当一类商品可以存储于多个仓库，每个仓库中可以存储多种商品，它们之间的联系就是存储，联系类型就是 n∶m。上述三种场景对应的 E-R 概念模型如图 3-4 所示。

3. 逻辑结构设计

逻辑结构设计是将概念结构设计转换为数据库系统所支持的数据模型的过程。与概念结构设计不同的是：概念结构设计是独立于数据模型的信息结构，而逻辑结构设计的任务就是将概念结构设计阶段的基本 E-R 模型（E-R 图）转换为与数据库系统所支持的数据模型相符合的逻辑结构。

E-R 图向关系模型的转换就是将实体和实体之间的联系转换为关系模式，并确定这些关系模式的属性和码。E-R 图是由实体、实体属性和实体间联系三个要素组成的，所以将 E-R 图转换为关系模型实际上就是将实体、实体属性、实体间的联系转换为

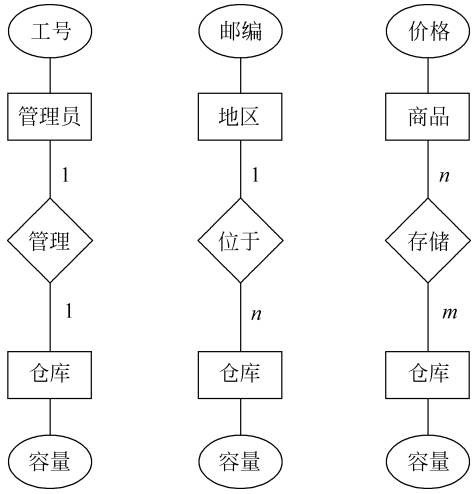

图 3-4　E-R 概念模型

关系模式。转换的方式是将一个实体转换为一个关系模式，实体的属性转换为关系的属性，实体的码转换为关系的码。在数据库关系中能够唯一区分每个记录（元组）的属性或属性的集合，称为码（候选码），被指定用来区分每个记录（元组）的码为主码。

数据库逻辑结构设计的方式与结果都是多样化的，为了获取高性能的数据库应用系统，设计者应该根据应用需求，进行数据逻辑结构的反复修改与调整，这个过程被称为数据逻辑结构的优化。关系数据逻辑结构的优化通常以规范化理论为指导，但过于高度规范化的逻辑结构也可能对数据库设计产生影响。这是因为高度规范化的逻辑设计，必然导致多个关系模式（多个表）的产生，而用户在进行连接查询时，大概率会触发多表间查询即关系模式的连接操作，而连接运算需要消耗大量的 CPU 资源，执行耗时较长，导致关系模型低效的主要原因就是连接运算。在这种情况下，低级范式更加适合，可以考虑将多个关系合并为一个关系。

4. 物理结构设计

物理结构设计是为逻辑结构选取一个最适合应用环境和数据库系统的物理结构，包括存储结构和存取方法。例如，关系数据库物理结构设计的内容主要包括关系模式对应的存储方法选取，关系、索引等数据库文件的物理存储结构设计。数据库管理系统一般提供多种存取方法，常用的如索引方法、聚簇方法等。其中 B＋树索引和哈希索引是数据库中经典的存取方法，使用最为普遍。

物理存储结构主要指数据的存放位置和存储结构，包括确定关系、索引、聚簇、日志与备份等存储策略和存储结构，确定系统配置等。以数据存放位置为例，为了提高系统性能，应根据应用实际情况将数据的易变部分与稳定部分、经常存取部分与存取频率较低部分分开存放；就系统配置而言，确定数据库物理结构后，还需对数据库系统的时间、空间效率，数据库运维代价和用户实际需求进行评价与权衡，如果实际应用效果不满足设计需求，还需要重新修改物理结构，甚至是修改数据库逻辑结构。因为不同数据库系统所提供的物理环境、存储结构、存取方法都不同，所以物理结构设计是具有多样性和不确定性的。设计者需要根据实际业务需求，进行细致的物理结构设计，并做迭代式优化。

对于 TPC-C 标准测试用例数据库而言，数据库物理结构设计要充分结合商品批发公司的实际存储环境、计算能力，需考虑客户提交订单时的系统快速响应能力等。

5. 数据库实施

数据库物理结构设计完成后，已完成数据库构建中的设计阶段，接下来进入数据库实施阶段。首先需要使用数据库系统提供的数据定义语言将数据库逻辑结构、物理结构描述出来，生成数据库系统识别的代码内容。然后对源代码进行调试生成目标模式，并组织数据入库。

事实上，数据库实施阶段包含两项重要的工作：一项是基础数据的载入，将有效数据进行整理、分类与关系构建，然后输入数据库；另一项是应用程序的编码和调试，根据入库数据调试应用的可用性。

6. 数据库运行和维护

数据库设计与实施完成后，数据库可正式投入运行。但由于应用环境不断变化，数据库运行的存储环境、计算环境也都在不断变化着，所以对数据库进行评价、优化等维护工作是一项长期迭代的任务，事实上也是数据库设计工作的延续。数据库的维护工作主要包括数据库的转存和恢复，数据库安全性、完整性的修正，数据库性能的监控、分析和改造，以及数据库的重组织与重构造。

以 TPC-C 标准测试用例数据库为例，当大型商品批发销售公司的营销方式、商品管理模式或客户订单管理方法发生改变，都可能影响到数据库服务的可靠性与性能，需要设计者根据实际需求变化，进行数据库构建的修正和完善。

3.2.5　数据库设计中的规范化设计

关系属性间可能存在着不同的依赖关系，而有些依赖关系具有不适合的性质。数据库设计过程中，按照属性间的依赖情况定义了不同的关系规范化程度，即范式。主要包括第一范式、第二范式、第三范式和 BCNF(Boyce Codd Normal Form)范式。基于这些范式，数据库设计过程中可以将具有不友好性质的关系转换为更合适的关系。

1. 函数依赖关系

函数依赖只能根据语义来确定，属于语义范畴。函数依赖的数学定义如定义 3.1 所示。

定义 3.1　存在某个属性集 U 及 U 的子集 ∂、β($\partial \subseteq R$，$\beta \subseteq R$)，假设 U 上存在关系模式 $R(U)$。若 $R(U)$ 中的全部元组 t_x 满足 $t_1[\partial]=t_2[\partial]$，则 $t_1[\beta]=t_2[\beta]$，则称 β 函数依赖于 ∂，记作 $\partial \rightarrow \beta$。

关于函数依赖关系的常见种类和表示方式如表 3-27 所示。

表 3-27　函数依赖关系

函数依赖关系	表 示 方 式
$X \rightarrow Y$ 是非平凡的函数依赖	$X \rightarrow Y$，$Y \not\subseteq X$
$X \rightarrow Y$ 是平凡的函数依赖	$X \rightarrow Y$，$Y \subseteq X$
X，Y 相互依赖	$X \rightarrow Y$，$Y \rightarrow X$($X \longleftrightarrow Y$)
Y 不函数依赖于 X	$X \not\rightarrow Y$

例如，一个客户居住在一个城市，具有唯一的姓氏，购买的货物价格是唯一的；一个城市里可能存在多个同姓氏的客户，购买的货物价格可能相同。则关系模式(**客户号，城市，货物价格，姓氏**)中客户号是所有属性的决定因素，存在函数依赖**客户号**→(**城市，货物价格，姓氏**)，但(**城市，货物价格，姓氏**)不是客户号的子集，所以该函数依赖是非平凡函数依赖。子关系模式(**城市，货物价格，姓氏**)中(**城市，货物价格，姓氏**)所有属性构成了决定因素组，存在函数依赖(**客户号，城市，货物价格，姓氏**)→(**城市，货物价格，姓氏**)，该函数依赖为平凡函数依赖。部分函数依赖的数学定义如定义 3.2 所示。

定义 3.2　存在某个属性集 U 及 U 的子集 ∂、β($\partial \subseteq R$，$\beta \subseteq R$)，假设 U 上存在关系模式 $R(U)$。若 ∂ 函数依赖于 β($\partial \rightarrow \beta$)，并且对于 ∂ 全部真子集 ∂^x($\partial^x \not\subseteq \partial$)，都满足 $\partial^x \not\rightarrow \beta$，

则称∂完全函数依赖于β，记作$\partial \xrightarrow{F} \beta$；反之，若存在$\partial^x$ 满足$\partial^x \to \beta$，则称∂部分依赖于β，记作$\partial \xrightarrow{P} \beta$。例如，一个客户可能提交了多个货运订单，每个货运订单内只有该客户订购的一种商品，客户有唯一的收货地址；一个货运订单内包含了多个客户同时下定的商品，则关系模式（**客户号，货运订单号，商品名称，收货地址**）中，存在函数依赖（**客户号，货运订单号**）→（**商品名称**），因为客户号和货运订单号不能单独决定商品名称，则商品名称完全依赖于客户号和货运订单号。同时存在函数依赖（**客户号，货运订单号**）→**收货地址**，但因为客户号属性就可以决定收货地址，收货地址属性不完全依赖于属性组（**客户号，货运单号**），所以该函数依赖是部分依赖。传递函数的数学定义如定义 3.3 所示。

定义 3.3 存在某个属性集 U 及 U 的子集∂、β、γ（$\partial \subseteq R$，$\beta \subseteq R$，$\gamma \subseteq R$），假设 U 上存在关系模式 $R(U)$。若∂函数依赖于 β（$\partial \to \beta$），β 不属于∂的真子集（$\beta \subsetneq \partial$），$\beta$ 不函数依赖于∂（$\beta \not\to \partial$），且 β 函数依赖于 γ（$\beta \to \gamma$），则∂、β、γ 间存在 γ 对∂的传递函数依赖，记作$\partial \xrightarrow{传递} \gamma$。

这里强调了 $\beta \not\to \partial$，是因为 $\beta \to \partial$，则$\partial \longleftrightarrow \beta$，事实上$\partial \xrightarrow{直接} \beta$，是直接函数依赖并不是传递函数依赖。

这里强调了 $Y \not\to X$，是因为如果 $Y \to X$，则 $X \longleftrightarrow Y$，实际上 $X \xrightarrow{直接} Z$，是直接函数依赖而不是传递函数依赖。

例如，一个客户购买的商品都从唯一的商品仓库发货，每个城市只有唯一的商品仓库。

则关系模式（**客户号，商品所属仓库，发货城市**）中，存在函数依赖**客户号→商品所属仓库，商品所属仓库→发货城市**，则该关系模式中**客户号→发货城市**为传递依赖。

2. 码

码是关系模式中的一个重要概念，其数学定义如定义 3.4 所示。

定义 3.4 存在某个属性集 U 及 U 的子集∂（$\partial \subseteq R$），假设 U 上存在关系模式 $R(U)$。若 $R(U)$中没有两条元组$(t_1, t_2) \subseteq t_x$ 在属性集∂上具有相同值，即若 $t_1 \neq t_2$，则 $t_1[\partial] \neq t_2[\partial]$，那么 $R(U)$中一个∂值一定可以唯一标识一个元组，此时称∂是 $R(U)$的超码（Super Key）。超码中可能包含若干冗余属性不对元组标识起作用，即该超码的真子集也是超码，此时最小的超码（所有真子集均不是超码）被称为候选码

(Candidate Key)，被选取用于设计数据库的任一候选码被称为主码(Primary Key)。

候选码中的属性称为主属性(Primary Attribute)；不包含在任何候选码中的属性称为非主属性(Non Primary Attribute)或非码属性(Non-Key Attribute)。外码的数学定义如定义 3.5 所示。

定义 3.5　存在某个属性集 U，假设 U 上存在关系模式 $R(U)_1$、$R(U)_2$。若 ϑ 是 $R(U)_1$ 的非主属性或属性组，但其是 $R(U)_2$ 的主属性或属性组，此时称 ϑ 是 $R(U)_1$ 的外部码(Foreign Key)，也称外码。

例如，一个客户具有唯一的收货地址；购买的货物从不同的货物仓库邮寄，每个货物仓库具有不同的仓库名，存放多种货物。

在关系模式(**客户号，收货地址，发货仓库号**)中，客户号是关系模式中所有属性的决定性属性，即候选码/主码。虽然发货仓库号不是该关系模式的候选码，但却是关系模式(**发货仓库号，仓库名，货物数目**)的候选码/主码，因此发货仓库号是该关系的外码。外码和主码的结合用于描述关系间的联系。

3. 范式

范式理论(Normal Form)起源于 20 世纪 70 年代，由英国计算机科学家(Edgar F. Codd)基于关系数据库模型总结提出。Edgar F. Codd 首先于 1971—1972 年提出了 1NF、2NF、3NF 的概念，然后又于 1974 年与美国计算机科学家 Raymond F. Boyce 合作对 3NF 进行了修正，进一步提出了巴斯范式 BCNF。1976 年，美国数学家、计算机科学家 Ronald Fagin 又提出了继 BCNF 后又一规范化理论标准 4NF。后续的科学家们也在不断地进行范式理论的研究，提出了 5NF 等更高级别的范式理论。

关系数据库设计中需要进行大量的关系模式规范化处理(Normalization)，即将存在各种依赖关系(部分依赖、传递依赖)的关系模式，基于范式理论并通过模式分解的方式转换成若干个符合高级范式的关系模式的集合。目前在关系数据库设计中，常被引用的范式理论主要有 1NF、2NF、3NF、BCNF、4NF、5NF，上述范式理论为层层递进的子集关系，具体为 5NF⊂4NF⊂BCNF⊂3NF⊂2NF⊂1NF。

第一范式(1NF)：关系模式中所有的属性都应该是原子性的，即数据表的每一列都不可分解。关系数据库中 1NF 是对关系模式的最基本要求，一般数据库设计中都需要满足 1NF。

例如，每个客户有唯一的姓氏，唯一的电话，可能有多个收货地址，则关系模式(**客户号，姓氏，电话**)中所有属性都不可分解，具有原子性，属于 1NF。而关系模式(**客户**

号，姓氏，电话，收货地址(**地址 1**，**地址 2**，**地址 3**))中收货地址可以分解为多个属性，不具有原子性，不满足 1NF。

第二范式(2NF)：在 1NF 基础上保证非码属性必须完全依赖于候选码，消除非码属性对候选码的部分函数依赖问题，2NF 的数学定义如定义 3.6 所示。

定义 3.6 存在某个属性集 U，假设 U 上存在关系模式 $R(U)$。若 $R(U)\in$1NF，且 $R(U)$中每个非主属性完全依赖于任何一个候选码(Candidate Key)，则称 $R(U)\in$2NF。

例如，每个客户有唯一的收货地址，提交了多个订单；每个订单对应了一种商品，有唯一的收货地址，则关系模式(**客户号**，**收货地址**，**订单号**)中存在函数依赖(**客户号**，**订单号**)→**收货地址**，因为订单号和客户号都可以独立决定收货地址，该关系模式存在部分函数依赖，因此不符合 2NF。不符合 2NF 的关系模式，往往会出现插入、删除异常以及修改复杂等问题。

第三范式(3NF)：在 2NF 的基础上消除了非主属性对其他非主属性的传递依赖问题，3NF 的数学定义如定义 3.7 所示。

定义 3.7 存在某个属性集 U，假设 U 上存在关系模式 $R(U)$。若 $R(U)\in$1NF，且 $R(U)$中不存在主属性∂、属性组 β、非主属性 $\gamma(\gamma\supsetneq\beta)$，满足$\partial\rightarrow\beta$，$\beta\rightarrow\gamma$ 成立，$\beta\not\rightarrow\partial$，则满足这种要求的关系 $R(U)$属于第三范式，记作 $R(U)\in$3NF。

例如，每位客户拥有唯一的货物运单号，每个货物运单包含多位客户的货物信息，每个货物运单对应唯一的发货仓库，则关系模式(**客户号**，**货物运单号**，**发货仓库**)中，存在函数依赖**客户号→货物运单号**，**货物运单号↛客户号**，**货物运单号→发货仓库**，则**客户号→发货仓库号**，即存在传递依赖。为满足 3NF，需通过关系模式分解将关系模式映射为多个关系模式，进而消除该传递依赖。若一个关系不属于 3NF，也会产生插入异常、删除异常以及修改复杂等类似的问题，3NF 的不彻底性还表现在可能存在主属性对码的部分依赖和传递依赖。

BCNF 范式是 3NF 的修正和补充，其数学定义如定义 3.8 所示。

定义 3.8 存在某个属性集 U 及 U 的子集∂、$\beta(\partial\subseteq R$，$\beta\subseteq R)$，假设 U 上存在关系模式 $R(U)$。若 $R(U)\in$1NF 且 $R(U)$中每个决定因素都包含码，即$\partial\rightarrow\beta$ 且$\beta\not\subseteq\partial$时$\partial$必包含码，则 $R(U)$属于巴斯范式，记作 $R(U)\in$BCNF。

例如，一个仓库只有一个仓库名和唯一的销售税；一个城市内可能有多个同名或同销售税的仓库，则关系模式(**仓库号**，**仓库名**，**城市**，**销售税**)中，存在函数依赖**仓库号→**(**仓库名**，**城市**，**销售税**)，关系模式中只有唯一的主码仓库号，没有其他属性对主码存在部分依赖和传递依赖，所以其属于 BCNF。

R∈BCNF，由于关系模式排除了任何属性（码和非码）对码的传递依赖和部分依赖，所以 $R \in 3NF$。但是反之不成立，若 $R \in 3CNF$，R 未必属于 BCNF。BCNF 是在函数依赖的条件下，对模式分解所能达到的最高分离程度，其彻底消除了插入和删除异常。

事实上，1NF～BCNF 都只是在函数依赖的范畴内讨论关系模式的优化方式。而除了函数依赖外，关系模式的属性间还存在着其他数据依赖，如多值依赖（Multi-Valued Dependency）、连接依赖（Join Dependency）。针对多值依赖问题，后续又引出了 4NF 的概念，而消除了 4NF 关系模式中存在的连接依赖后，则可进一步达到 5NF 的规范化关系模式。本节只介绍了 1NF，2NF，3NF，BNCF 范式，对于规范化程度更高的 4NF 与 5NF 不做详细介绍。

满足 1NF 往往是数据库设计中的最基本要求，并且满足 1NF 的关系模式就是合法的。但是插入异常、删除异常、修改复杂及数据冗余等问题，严重影响了 1NF 关系模式的应用。因此，设计了规范化标准来解决这些问题，通过对关系模式的投影分解，将低级关系模式分解为若干高级关系模式，进而规避上述问题。综上所述，规范化的核心思想就是逐步消除关系模式中不友好的关系模式，使各关系模式达到高度单一化。

3.3　数据库约束

用数据表来模拟现实世界中数据的实体集和联系，数据表支持数据的存储操作。为了保证数据的完整性，需要在数据上附加一些限制，只有满足这些限制条件的操作，才可以被数据库系统接受。上述这种限制，称作数据库的约束，约束主要有五类，如表 3-28 所示。

表 3-28　数据库约束分类

约束类型	约束名称	约束描述
NOT NULL	非空约束 C	指定的列不允许为空值
UNIQUE	唯一约束 U	指定的列中没有重复值，或该表中每一个值或者每一组值都将是唯一的

续表

约束类型	约束名称	约束描述
PRIMARY KEY	主键约束 P	唯一地标识出表的每一行，且不允许空值，一个表只能有一个主键约束
FOREIGN KEY	外键约束 R	一个表中的列引用了其他表中的列，使得存在依赖关系，可以指向引用自身的列
CHECK	条件约束 C	指定该列是否满足某个条件

当然，还包括一些其他约束，如默认约束、自增约束、级联约束等。本节将主要介绍表 3-28 所述五类数据库核心约束，并列举相应示例进行详细说明。

3.3.1 数据完整性

约束可以维护数据的完整性。而数据的完整性有不同的场景，针对不同的数据完整性，需要选择不同的约束，常见的数据完整性分类如表 3-29 所示。

表 3-29 数据完整性分类

类别	类别描述	涉及约束
实体完整性	表中记录不重复并且每条记录都有一个非空主键	PRIMARY KEY、UNIQUE
域完整性	属性值必须与属性类型、格式、有效范围相吻合	CHECK、FOREIGN KEY、NOT NULL
参照完整性	不能引用不存在的值	FOREIGN KEY
自定义完整性	根据特定业务领域定义的需求完整性（例如某个属性的取值为 0～100）	CHECK

3.3.2 约束操作

约束可以指定一列或多列作为一组，还可以为整个表设计约束。一般指定单列的时候，可以跟在字段定义之后或是在表定义的最后；指定多列的时候，则必须在表定义的最后；表约束要在表定义外面。具体使用可以在创建表的时候指定，也可以后续用 ALTER 命令修改。

在创建约束的时候，建议每一个约束指定一个名称，方便操作人员修改和查找，即使数据库系统内部会为其维护一个内部名称。约束在数据库系统中具体的操作示例，将在下面进行介绍。

3.3.3　非空约束

非空约束(NOT NULL),能够限制数据不能为空,这种约束只作用于列级。例如:创建如表 3-30 所示的 customer 表,用于记录消费者信息,并进行相应数据的插入操作。表中 c_custkey 字段表示顾客编号,c_name 字段表示顾客姓名。

表 3-30　customer 表结构

customer		
Column	c_custkey	c_name
Type	INTEGER	VARCHAR

创建 customer 表时,在字段后直接添加“NOT NULL”关键字,即可对此列创建非空约束。具体语句如下:

```
CREATE TABLE customer(
    c_custkey INTEGER NOT NULL,
    c_name VARCHAR(25)
);
```

执行上述 SQL 语句后,即可创建 customer 表,并设置 c_custkey 列为非空约束。查看表结构,可以看到 c_custkey 列被设置了 NOT NULL 属性,如表 3-31 所示。

表 3-31　customer 表字段类型及属性

Column	Type	Modifiers
c_custkey	INTEGER	NOT NULL
c_name	VARCHAR(25)	

在 customer 表中插入数据时,如插入一条张三的信息{0,zhangsan},信息完整可以插入。但如果把 c_custkey 设成 NULL 进行插入时,受限于非空约束,这个操作会被数据库系统拒绝。例如:

```
INSERT INTO customer (c_custkey,c_name) VALUES (0,'zhangsan');
INSERT INTO customer (c_custkey,c_name) VALUES (NULL,'zhangsan');
```

以上两条插入语句,第一条顺利执行。第二条将提示类似“violates not-NULL constraint”的违反约束错误,因为非空约束能限制插入数据不为空值。SQL 语句执行

结果字段状态如表 3-32 所示。

表 3-32　SQL 语句执行结果字段状态表 1

Status	c_custkey	c_name
OK	0	zhangsan
ERR	NULL	zhangsan

3.3.4　唯一约束

使用唯一约束(UNIQUE)指定某属性列,标识该列不会存在重复值,即该列中每一个值都是唯一的。需要注意的是,唯一约束下的列可以存在 NULL 值,而且 NULL 可以存在多个。

例如:新建一张 nation 表,表结构如表 3-33 所示,并指定唯一约束。

表 3-33　nation 表结构

nation		
Column	n_nationkey	n_name
Type	INTEGER	VARCHAR

执行下面的 SQL 语句,创建 nation 表,表中 n_nationkey 字段表示国家代号,n_name 字段为国家名称。

```
CREATE TABLE nation(
  n_nationkey INTEGER,
  n_name VARCHAR(25)
);
```

接下来为 n_nationkey 字段增加 UNIQUE 约束,然后插入数据记录进行检验,SQL 语句具体如下。

```
ALTER TABLE nation ADD CONSTRAINT uk_nation_idunique UNIQUE (n_nationkey);
INSERT INTO nation(n_nationkey, n_name) VALUES(0, 'china');
INSERT INTO nation(n_nationkey, n_name) VALUES(0, 'china1');
INSERT INTO nation(n_nationkey, n_name) VALUES(1, 'china1');
```

插入 n_nationkey 为 0 的数据后,再次插入(0,'china1'),会提示“unique constraint”约束,插入失败。修改 n_nationkey 后,插入(1,'china1')数据成功。至此,上述操作示例

验证了唯一约束对有效数据的校验。

对于特殊的 NULL 值，执行如下两条 SQL 语句，可以看到 NULL 值正常插入，可以存在多个。

```
INSERT INTO nation(n_nationkey, n_name) VALUES(NULL,'china2');
INSERT INTO nation(n_nationkey, n_name) VALUES(NULL,'china3');
```

SQL 语句执行结果字段状态如表 3-34 所示。

表 3-34　SQL 语句执行结果字段状态表 2

Status	n_nationkey	n_name
OK	0	China
ERR	0	China
OK	1	China1
OK	NULL	China2
OK	NULL	China3

3.3.5　主键约束

主键约束(PRIMARY KEY)可以看作非空约束和唯一约束的结合：既不允许为 NULL，也不允许重复，且一张表只能有一个主键约束。

例如：创建表 orders 实现主键约束。orders 表为订单信息表，希望每个订单都有一个唯一且有效的标识可供查找。本例通过主键约束实现 o_orderkey，其表示订单编号，o_totalprice 表示订单金额，如表 3-35 所示。

表 3-35　orders 表结构

orders		
Column	o_orderkey	o_totalprice
Type	INTEGER	DOUBLE

执行下面的 SQL 语句，创建 orders 表。

```
CREATE TABLE orders(
  o_orderkey INTEGER PRIMARY KEY,
  o_totalprice DOUBLE
);
```

接着进行数据插入操作，分别插入 NULL 值和重复值进行测试。具体 SQL 语句如下：

```
INSERT INTO orders(o_orderkey, o_totalprice) VALUES(0, 10.1);
INSERT INTO orders(o_orderkey, o_totalprice) VALUES(0, 20.2);
INSERT INTO orders(o_orderkey, o_totalprice) VALUES(NULL, 30.3);
INSERT INTO orders(o_orderkey, o_totalprice) VALUES(1, 40.4);
```

以上四条插入语句，第一条顺利插入；对于重复值报了“violates unique constraint”错误；对于 NULL 报了“violates not-NULL constraint”错误；第四条语句可以顺利执行。对于表中的记录，主键通过保证一个有效且唯一的数据，保证了实体完整性。所有 SQL 语句执行结果字段状态如表 3-36 所示。

表 3-36　SQL 语句执行结果字段状态表

Status	o_orderkey	o_totalprice
OK	0	10.1
ERR	0	20.2
ERR	NULL	30.3
OK	1	40.4

3.3.6　外键约束

数据库中，如果将所有的数据都存放到一张表中，会存在表结构不清晰、扩展性差等问题。因此需要设计多张表，通过外键(FOREIGN KEY)表明表之间的关系。一张表的 FOREIGN KEY 指向另一张表的 PRIMARY KEY。

以表 3-35 所示的 orders 表结构为例，首先创建 orders 表，接着创建 lineitem 表，表示在线商品的信息，其中 l_orderkey 为 FOREIGN KEY，表示订单编号，指向 orders 表的 o_orderkey，如图 3-5 所示。

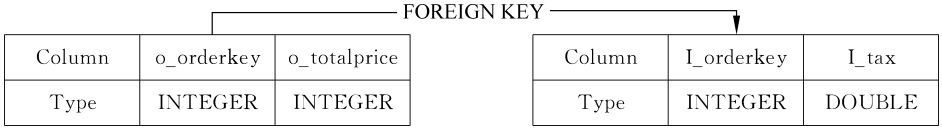

图 3-5　orders、lineitem 双表关系结构

执行下面的 SQL 语句，创建 lineitem 表。

```
CREATE TABLE lineitem(
    l_orderkey INTEGER,
    l_tax DOUBLE,
    FOREIGN KEY (l_orderkey) REFERENCES orders(o_orderkey)
);
```

创建 lineitem 表之后，执行下面的 SQL 语句，插入数据：

```
INSERT INTO lineitem(l_orderkey, l_tax) VALUES(1, 2.34);
INSERT INTO lineitem(l_orderkey, l_tax) VALUES(2, 2.34);
```

示例中，插入(1,2.34)成功，但插入(2,2.34)失败。这是因为在执行操作的时候，数据库系统通过外键约束从 orders 表检查 l_orderley 对应的外键 o_orderkey 中的值，在存在记录{1}时接受了操作请求；在不存在记录{2}时，拒绝了操作请求。这样就保证了 lineitem 表中的每条记录在 orders 中都有对应的数据存在，即维护了引用完整性。示例中 SQL 语句执行结果状态如表 3-37 所示。

表 3-37　SQL 语句执行结果字段状态表 3

Status	I_orderkey	I_tax
OK	1	2.34
ERR	2	2.34

3.3.7　条件约束

条件约束(CHECK)用来限制列值的范围。在表定义时对单个列进行 CHECK 约束，则该列只允许特定的值。例如，在上面 lineitem 表中添加 l_linenumber 列，根据 l_linenumber 的实际含义表示每条数据的记录值，该列值是一个非负整数。所以将其类型设置为整数且大于零。执行如下 SQL 语句，插入非法值时数据库系统会报错。

```
ALTER TABLE lineitem
ADD l_linenumber INTEGER CHECK (l_linenumber > 0);
INSERT INTO lineitem VALUES(1, 2.34, 5);
INSERT INTO lineitem VALUES(1, 2.34, -6);
```

第一条插入语句顺利执行。第二条插入语句产生“check constraint”错误提示，以保证插入的准确性。所有 SQL 语句执行结果字段状态如表 3-38 所示。

表 3-38　SQL 语句执行结果字段状态表 4

Status	I_orderkey	I_tax	I_linenumber
OK	1	2.34	5
ERR	1	2.34	−6

在实际的数据库设计中，需要考虑到现实数据的属性，则可以通过类似的条件约束对用户的数据操作提前校验、规范输入。程序设计中的枚举就是一种典型的条件约束。

3.4　小结

本章介绍了数据库设计中所涉及的关系代数理论，基本设计流程和规范化设计，以及为维护数据完整性所需要的约束。

关系代数是关系数据库标准查询的基础，本章一开始介绍了关系代数的基本概念，从具体到抽象，用实际例子介绍了关系代数中的各种操作，并结合 SQL 演示了在数据库系统中的关系运算。

数据库设计主要讨论了数据库设计的方法和步骤，详细介绍了数据库设计各个阶段的目标、方法等，同时介绍了数据库设计中的规范化设计。但是要注意，规范化设计理论为数据库设计提供了基础的思路，但并不是规范化程度越高，数据库模式就越好，要在实际工作学习中运用这些思想，结合应用环境和实际的需求场景合理规划选择，设计符合需求的数据库系统。

约束是数据库模式的一部分，在数据库中能够保证数据的完整性和一致性，所有违反约束的插入或修改操作都是不允许的。本章主要介绍和演示了非空约束、唯一约束、主键约束、外键约束、条件约束。

习题

(1) 什么是关系代数?

(2) 试述等值连接与自然连接的区别和联系。

(3) 关系代数的基本运算有哪些? 如何用这些基本运算来表示其他运算?

(4) 设有关系 R 和 S,如图 3-6 所示,其中(a<b<c<d),分别计算:

① $R \bowtie S$;

② $R \bowtie S(C<D)$;

③ $\sigma_{B=C}(R \times S)$。

C	D
c	e
b	f

(a) 关系 R

A	B	C
a	b	c
d	a	e

(b) 关系 S

图 3-6　关系 R、S

(5) 试述数据库设计的基本步骤及其各个阶段的设计描述。

(6) 试述数据库设计过程中结构设计部分形成的数据库模式。

(7) 需求分析阶段的设计目标是什么? 调查的内容是什么?

(8) 试述数据库概念结构设计的重要性和设计步骤。

(9) 试述数据库物理结构设计的内容和步骤。

(10) 假设有 customers(id,name,age,sex)这张表,请以数据库设计人员的角度,对表中 4 个字段设置不同的约束。

(11) 有以下 2 张表:

```
country(id,name,capital,nationalday);
customers(id,name,sex,age,countryname);
```

使用 SQL 语句在创建表的时候指定 2 张表中字段需要的约束,同时设置 customers 表中的 countryname 和 country 表中的 name 绑定外键约束。