# SinoDB 数据库设计和实现指南

星瑞格 SinoDB 产品系列

SinoDB

版本 12.10

# 目录

# 插图清单

# 表格清单

# 简介

## 关于本出版物

本出版物提供信息来帮助您进行 SinoDB® 数据库的设计、实现和管理。包含说明进行数据库设计的不同方法的数据模型并阐述如何使用结构化查询语言 (SQL) 来实现和管理数据库。

本出版物是讨论 SQL SinoDB® 实现的若干出版物之一。《*SinoDB® SQL* 指南*:* 教程》 阐述如何使用基本的和高级的 SQL 以及存储过程语言 (SPL) 例程来访问和处理数据库中的数据。《*SinoDB® SQL* 指南*:* 语法》包含 SQL 和 SPL 的所有语法描述。《*SinoDB® SQL* 指南*:* 参考》 提供了 SQL 语句外的其他方面的参考信息。

### 用户类型

本出版物是为以下用户编写的：

- 数据库管理员
- 数据库服务器管理员
- 数据库应用程序程序员

本出版物假定您有以下知识背景：

- 对于计算机、操作系统和操作系统提供的实用程序的应用知识
- 使用关系数据库的相关经验或者了解数据库概念
- 一些计算机编程经验

### 演示数据库

DB-Access实用程序随SinoDB®数据库服务器产品一起提供，它包括一个或多个以下演示数据库：

- stores_demo 数据库以一家虚构的体育用品批发商的有关信息举例说明了关系模式。星瑞格®出版物中的许多示例均基于 stores_demo 数据库。
- superstores_demo 数据库举例说明了对象关系模式。superstores_demo 数据库包含扩展数据类型、类型和表继承以及用户自定义例程的示例。

有关如何创建和填充演示数据库的信息，请参阅《*SinoDB® DB-Access* 用户指南》。有关数据库及其内容的描述，请参阅《*SinoDB® SQL* 指南*:* 参考》。

用于安装演示数据库的脚本位于 UNIX™ 平台上的 $INFORMIXDIR/bin 目录和 Windows™ 环境中的 %INFORMIXDIR%\bin 目录中。

## 示例代码约定

SQL 代码的示例出现在整个出版物中。除非另有说明，代码不特定于任何单个的 SinoDB® 应用程序开发工具。

如果示例中仅列出 SQL 语句，那么它们将不用分号定界。例如：您可能看到以下示例中的代码：

```
CONNECT TO stores_demo
...

DELETE FROM customer
   WHERE customer_num = 121
...

COMMIT WORK
```

```
DISCONNECT CURRENT
```

要将此 SQL 代码用于特定产品，必须应用该产品的语法规则。例如，如果使用的是 SQL API，那么必须在每条语句的开头使用 EXEC SQL ，并在每条语句的结尾使用分号（或其他合适的定界符）。 如果使用的是 DB-Access，那么必须用分号将多条语句隔开。

> 提示： 代码示例中的省略点表示在整个应用程序中可添加更多的代码，但对于正在讨论的概念是不需要显示的。

有关使用特定应用程序开发工具或 SQL API 的 SQL 语句的详细指导，请参阅您的产品文档。

## 符合行业标准

SinoDB® 产品符合各种标准。

基于 SinoDB® SQL 的产品完全符合 SQL-92 入门标准（发布为 ANSI X3.135-1992），即 ISO 9075:1992 标准。另外，SinoDB® 数据库服务器的许多功能都遵循 SQL-92 中级和最高级标准，以及 X/Open SQL 公共应用程序环境 (CAE) 标准。

# 第 I 部分

# 数据库设计和实现基础

# 规划数据库

本章描述了若干问题，数据库管理员 (DBA) 必须了解这些问题才能有效地规划数据库。本章包含有关选择数据库的数据模型、使用符合 ANSI 标准的数据库以及对数据库使用自定义语言环境的信息。

## 为数据库选择数据模型

在使用 SinoDB® 产品创建数据库之前，您必须决定要使用哪种类型的数据模型来设计数据库。本手册描述了下列数据库模型：

关系数据模型

此数据模型代表用于联机事务处理 (OLTP) 的数据库设计。OLTP 的用途是处理大量的小型事务而不丢失任何事务。OLTP 数据库旨在处理企业的日常需求，并且针对这些需求调整数据库性能。本手册的数据库设计和实现基础 在第11页描述了如何为 OLTP 构建和实现关系数据模型。管理数据库 在第49页包含有关如何管理数据库的信息。

对象关系数据模型

SinoDB® 支持对象关系数据库，这些对象关系数据库利用基本的关系设计原理，但包括扩展数据类型、用户自定义例程、用户定义的强制转型和用户定义的聚合等功能来扩展关系数据库的功能。本手册的对象关系数据库 在第82页包含有关如何使用 SinoDB® 的可扩展功能来扩展可在数据库中存储的数据种类以及在组织和访问数据的方式方面提供更大灵活性的信息。

维数据模型

此数据模型通常用于构建数据集市，数据集市是一种数据仓库。在数据仓库环境中，优化数据库以便进行数据检索和分析。此类信息处理称为联机分析处理 (OLAP) 或决策支持处理。

本章的其余部分描述这些决定的含义以及总结您所作的决定将对数据库产生何种影响。

## 使用符合 ANSI 标准的数据库

当您在 CREATE DATABASE 语句中使用 MODE ANSI 关键字时，请创建符合 ANSI 标准的数据库。然而，创建符合 ANSI 标准的数据库并不能确保此数据库一直保持符合 ANSI 标准。如果对符合 ANSI 标准的数据库执行非 ANSI 操作（如 CREATE INDEX），那么您将收到警告，但应用程序不会禁止该操作。

由于以下原因，您可能希望创建符合 ANSI 标准的数据库：

- 对对象的权限和访问权

  ANSI 规则管理对对象（如表和同义词）的权限和访问权。
- 名称隔离

ANSI 表命名模式允许不同的用户在数据库中创建表而不会发生名称冲突。

- 事务隔离
- 数据恢复

符合 ANSI 标准的数据库为 SinoDB® 强制执行无缓冲日志记录和隐式事务。

可以对符合 ANSI 标准的数据库和不符合 ANSI 标准的数据库使用相同的 SQL 语句。

## 符合 ANSI 标准的数据库与不符合 ANSI 标准的数据库之间的差别

您指定为符合 ANSI 标准的数据库与不符合 ANSI 标准的数据库在以下方面具有不同行为：

### 事务

事务是可以视为单个工作单元的 SQL 语句集合。在符合 ANSI 标准的数据库中发出的所有 SQL 语句都将自动包含在事务中。对于不符合 ANSI 标准的数据库，事务处理是可选项。

在不符合 ANSI 标准的数据库中，事务用 BEGIN WORK 语句和 COMMIT WORK 或 ROLLBACK WORK 语句括起来。然而，在符合 ANSI 标准的数据库中，由于所有语句都自动包含在一个事务中，因此不需要 BEGIN WORK 语句。您需要只使用 COMMIT WORK 或 ROLLBACK WORK 语句来指示事务的结束。

有关事务的更多信息，请参阅实现关系数据模型 在第41页和《*SinoDB® SQL* 指南*:* 教程》。

### 事务日志记录

符合 ANSI 标准的数据库在运行时进行无缓冲事务日志记录。在符合 ANSI 标准的数据库中，不能将日志记录模式更改为缓冲日志记录，也不能关闭日志记录。

不符合 ANSI 标准的 SinoDB® 数据库在运行时能够进行缓冲日志记录或无缓冲日志记录。无缓冲日志记录能够提供更全面的数据恢复，但缓冲日志记录能够提供更好的性能。

有关更多信息，请参阅《*SinoDB® SQL* 指南*:* 语法》中对 CREATE DATABASE 语句的描述。

### 所有者命名

在符合 ANSI 标准的数据库中，所有者命名是强制进行的。当在 SQL 语句中提供对象名称时，除非您是该对象的所有者，否则 ANSI 标准会要求该名称包括前缀 *owner*。*owner* 与名称的组合在数据库中必须是唯一的。如果您是该对象的所有者，那么数据库服务器会将您的用户名作为缺省值。

不符合 ANSI 标准的数据库不会强制执行所有者命名。有关更多信息，请参阅《*SinoDB® SQL* 指南*:* 语法》中的所有者名称的部分。

### 对象的权限

在数据库中创建表时，对于缺省情况下将表级别权限授予哪些用户，符合 ANSI 标准的数据库和不符合 ANSI 标准的数据库是不同的。ANSI 标准指定数据库服务器将任何表级别权限只授予表所有者（以及 DBA，如果他们不是同一用户）。但是，在不符合 ANSI 标准的数据库中，会将权限授予 PUBLIC。另外，数据库服务器提供了两个表级别权限：Alter 和 Index，这两个权限没有包括在 ANSI 标准中。

要运行用户自定义例程，您必须拥有该例程的 Execute 权限。在为符合 ANSI 标准的数据库创建仅所有者有权限的过程时，只有用户自定义例程的所有者拥有 Execute 权限。在不符合 ANSI 标准的数据库中创建仅所有者有权限的例程时，缺省情况下数据库服务器会将 Execute 权限授予 PUBLIC。

将 NODEFDAC 环境变量设置为“yes”时，不符合 ANSI 标准的数据库会模拟符合 ANSI 标准的数据库的行为：在用户创建表或仅所有者有权限的例程时，不会自动授予 PUBLIC 权限。有关权限的更多信息，请参阅授予和限制对数据库的访问权 在第56页以及《*SinoDB® SQL* 指南*:* 语法》中对 GRANT 语句的描述。

### 缺省隔离级别

数据库隔离级别指定程序与其他程序的并发操作相隔离的程度。所有符合 ANSI 标准的数据库的缺省隔离级别都是 Repeatable Read。支持事务日志记录的不符合 ANSI 标准数据库的缺省隔离级别是 Committed Read。不使用事务日志记录的不符合 ANSI 标准数据库的缺省隔离级别是 Uncommitted Read。有关隔离级

别的信息，请参阅《*SinoDB® SQL* 指南*:* 教程》以及《*SinoDB® SQL* 指南*:* 语法》中对 SET TRANSACTION 和 SET ISOLATION 语句的描述。

字符数据类型

如果数据库不符合 ANSI 标准，那么在字符字段（CHAR、CHARACTER、LVARCHAR、NCHAR、NVARCHAR、VARCHAR 和 CHARACTER VARYING）接收到超出字段指定长度的字符串时，您不会收到错误消息。数据库服务器会截断多余的字符而不会产生错误消息。因此，对于 CHAR(*n*) 列或变量，当插入或更新值的长度超过 *n* 个字节时，不会强制实施数据的语义完整性。

在符合 ANSI 标准的数据库中，如果任何字符字段（CHAR、CHARACTER、LVARCHAR、NCHAR、NVARCHAR、VARCHAR 和 CHARACTER VARYING）接收到超出字段指定范围的字符串，那么您会收到错误消息。

DECIMAL 数据类型

如果数据库不符合 ANSI 标准，那么声明时使用了精度却无小数位的 DECIMAL 数据类型可以存储指定精度的浮点值。如果您既没有指定精度也没有指定小数位，那么缺省精度为 16。

在符合 ANSI 标准的数据库中，所有 DECIMAL 值都为定点值并且必须使用显式精度来声明。如果没有为 DECIMAL 数据类型指定小数位，那么小数位等于 0，并且只能存储整数值。

转义字符

在符合 ANSI 标准的数据库中，转义字符只能对百分号 (%) 和下划线 (_) 字符的特殊意义进行转义。您也可以使用转义字符对它本身进行转义。有关转义字符的更多信息，请参阅《*SinoDB® SQL* 指南*:* 语法》中的条件的部分。

游标行为

如果数据库不符合 ANSI 标准，那么为 SELECT 语句声明更新游标时必须使用 FOR UPDATE 关键字。SELECT 语句也必须满足下列条件：

- 从单个表中选择。
- 不包含任何聚合函数。
- 不包含 DISTINCT、GROUP BY、INTO TEMP、ORDER BY、UNION 或 UNIQUE 子句和关键字。

在符合 ANSI 标准的数据库中，声明游标时 FOR UPDATE 关键字是隐含的，且满足以上列表所述限制的所有游标都是潜在的更新游标。您可以在 DECLARE 语句中用 FOR READ ONLY 关键字将游标指定为只读的。

有关更多信息，请参阅《*SinoDB® SQL* 指南*:* 语法》中对 DECLARE 语句的描述。

SQL 通信区的 SQLCODE 字段

如果没有任何行满足 DELETE、INSERT INTO *tablename* SELECT、SELECT...INTO TEMP 或 UPDATE 语句的搜索条件，那么数据库服务器会将 SQLCODE 设置为 100（如果数据库符合 ANSI 标准）或 0（如果数据库不符合 ANSI 标准）。

有关更多信息，请参阅《*SinoDB® SQL* 指南*:* 教程》中对 SQLCODE 的描述。

同义词行为

同义词在符合 ANSI 标准的数据库中始终是专用的。如果尝试创建公共同义词或使用 PRIVATE 关键字在符合 ANSI 标准的数据库中指定专用同义词，那么您会接收到错误消息。

有关更多信息，请参阅《*SinoDB® SQL* 指南*:* 语法》中对 CREATE SYNONYM 语句的描述。

## 确定现有数据库是否符合 ANSI 标准

以下列表描述了确定数据库是否符合 ANSI 标准的两种方法：

- 从 sysmaster 数据库中，您可以执行以下语句：

```
SELECT name,is_ansi FROM sysmaster:sysdatabases
```

  对于数据库服务器上的每个数据库，此查询对符合 ANSI 标准的数据库返回值 1，对不符合 ANSI 标准的数据库返回值 0。
- 如果您正在使用 SQL API（如 SinoDB® ESQL/C），那么可以测试 SQL 通信区 (SQLCA)。具体地说，在使用 DATABASE 或 CONNECT 语句打开符合 ANSI 标准的数据库后，SQLCAWARN 结构中的第三个元素会立即包含 W。有关 SQLCA 的信息，请参阅《*SinoDB® SQL* 指南: 教程》或 SQL API 手册。

## 对数据库使用自定义语言环境 (GLS)

Global Language Support (GLS) 允许您使用不同的语言环境。GLS 语言环境是为特定语言或文化定义约定的环境。

在缺省情况下，SinoDB® 产品使用美国英语 ASCII 代码集并在具有 ASCII 排序顺序的美国英语环境中执行。如果您打算使用以下任何功能，请将环境设置为适应非缺省语言环境：

- 数据包含非 ASCII 字符
- 用户指定的对象名包含非 ASCII 字符
- 与非缺省代码集的排序和排序顺序一致
- 文化特异性的排序规则和排序顺序，如在字典或电话号码簿中使用的那些排序规则和排序顺序

SinoDB® 支持 UTF-8 Unicode 语言环境。

与其他语言环境不同，UTF-8 允许单个数据库存储使用不同代码集的两种或两种以上自然语言的字符串。

有关 GLS 环境变量的描述以及如何实现非缺省语言环境的详细信息，请参阅《*SinoDB® GLS* 用户指南》。

# 构建关系数据模型

创建关系数据库的第一步是构造数据模型：对要存储的数据进行精确和完整地定义。本章概述了数据建模的一种方法。有关定义数据模型的特定于列的属性的信息，请参阅选择数据类型 在第29页。要了解如何实现本章描述的数据模型，请参阅实现关系数据模型 在第41页。

要理解本章的内容，您需要对 SQL 及关系数据库理论有基本的了解。

## 构建数据模型

您已经了解了数据库中数据的类型以及如何组织该数据。这些信息是数据模型的开始。构建具有形式符号表示法的数据模型有下列优点：

- 您可完全通过数据模型来思考。

  脑海中的模型通常包含未经检查的假定；当设计正式出台时，您会证实这些假定。
- 可以使设计让其他人更容易理解。

  正规的表示法使得模型更加明确，因此其他人可以在同一形式中返回意见与建议。

## 实体关系数据模型概述

存在多种形式的数据建模方法。大多数方法会使您全面而精确地工作。如果您了解某种方法，那么请务必使用该方法。

本章总结了实体关系 (E-R) 数据模型。E-R 数据建模方法有以下步骤：

1. 标识和定义主体数据对象（实体、关系和属性）。
2. 使用 E-R 方法将数据对象图形化。
3. 将 E-R 数据对象转换成关系结构。
4. 解析逻辑数据模型。
5. 将逻辑数据模型规范化。

本章说明了步骤 1 到步骤 5。实现关系数据模型 在第41页包含有关将逻辑数据模型转换为物理模式的最终步骤的信息。

数据建模的最终产物是以图形编码的完全定义的数据库设计，其类似于图 *21:* 个人电话号码簿的数据模型 在第28页（该模型显示了个人电话号码簿的最终的表集合）。个人电话号码簿是本章中使用的一个示例。由于该示例足够小，可在一个章节中完全开发，但也足够大以显示整个方法，所以我们使用此示例而不是使用演示数据库。

# 标识和定义主体数据对象

要创建数据模型，您首先需标识和定义主体数据对象：实体、关系和属性。

## 发现实体

实体是用户特别感兴趣的主体数据对象。通常是在数据库中记录的人员、地点、事物或事件。如果数据模型是语言，那么实体就是名词。软件附带的演示数据库包含以下实体：

- *customer*
- *orders*
- *items*
- *stock*
- *catalog*
- *cust_calls*
- *call_type*
- *manufact*
- *state*

### 选择可能的实体

您大概可以立即列出数据库的若干实体。请列出您可以标识的所有实体的初步列表。然后，与数据库的潜在用户会谈，了解他们对数据库中必须记录的内容的意见。确定每个实体的基本特征，如“必须至少有一个地址与名称相关联”。您所作的关于实体的所有决定都将成为商业规则。图 *1:* 电话号码簿的部分页 在第16页页面上的电话薄示例提供了本章中示例的一些商业规则。

以后，在将数据模型规范化时，一些实体可以扩充或者变为其他数据对象。有关更多信息，请参阅规范化数据模型 在第27页。

### 实体列表

当实体列表似乎已完成时，请检查列表以确保每个实体都具有下列性质：

- 重要性。

  只列出对数据库用户重要的实体以及值得花费精力和成本进行计算机制表的实体。
- 通用性。

  只列出事物的类型而不是个别的实例。例如：*symphony* 可能是一个实体，而 *Beethoven's Fifth* 将是实体实例或实体出现形式。
- 基本性。

只列出独立存在并且不需要其他任何内容来说明的实体。任何可以称之为特性、特征或描述的事物都不是实体。例如：部件号是称为部件的基本实体的特征。并且，不要列出可以从其他实体派生的事物；例如：避免任何可以在 SELECT 表达式中计算得出的和、平均数或其他数量。

- 单一性。

  确保您命名的每个实体都表示单个类。不能将其划分成子类别（每个子类别都具有自己的特征）。在图 1: 电话号码簿的部分页 在第16页的电话号码簿示例中，电话号码（这是一个表面上很简单的实体）实际上由三个类别组成，每个类别都具有不同的特征。

这些选择既不是简单的也不是自动的。要发现实体的最佳选择，您必须仔细考虑要存储的数据的性质。当然，那就是形式数据模型的关键所在。下一节将详细描述电话号码簿示例。

### 电话号码簿示例

假设您为个人电话号码簿创建数据库。数据库模型必须记录用户需要的人员和组织的名称、地址和电话号码。

首先定义实体。仔细查看电话号码簿中的某一页以确定其包含的实体。下图显示电话号码簿中的样本页。

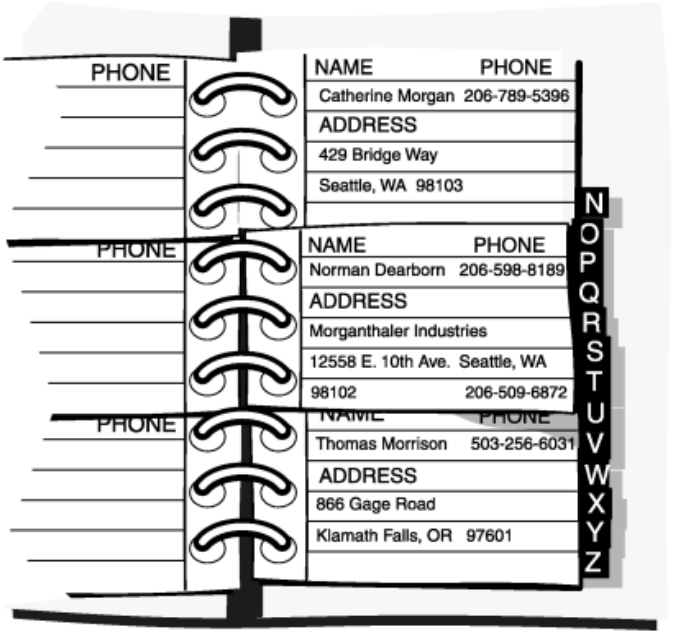


图 1: 电话号码簿的部分页

现有数据的物理形式可能会误导您。注意不要让电话号码簿中的页和条目布局误导您去尝试指定表示电话号码簿中条目（带有名称、电话号码和地址字段的按字母顺序排列的记录）的实体。您要做的是对数据而不是对介质建模。

### 通用的重要实体

首先我们会看到电话号码簿中记录的实体包含下列各项：

- （人员和组织的）名称
- 地址
- 电话号码

这些实体符合前面的标准吗？它们对模型显然很重要并且是通用的。

### 基本实体

一项好的测试是询问实体的数目是否可以独立于任何其他实体而变化。电话号码簿有时会列出没有电话号码或当前地址的人员（搬家或换工作的人员），并且也可以同时列出多个人员使用的地址和电话号码。这三个实体的数目全都可以独立地变化；这一事实有力地表明它们是基本的，而不是从属的。

### 单一实体

名称可以分为人员姓名和企业名。在此模型中，您决定所有名称都应具有相同的特征；即，您不打算对公司和人员分别记录不同的信息。同样，您决定只存在一种类型的地址；您无需将家庭地址与办公地址区别对待。

但是，您也认识到存在多种类型的电话号码。语音号码由人员接听，传真号码连接到传真机，而调制解调器号码连接到计算机。您决定要记录关于每类号码的不同信息，因此这三种类型是不同的实体。

对于个人电话号码簿示例，您决定要跟踪下列实体：

- Name
- Address (mailing)
- Telephone number (voice)
- Telephone number (fax)
- Telephone number (modem)

### 关系图 - 实体

在本章的后面，您可以学习如何使用 E-R 图。现在，为电话号码簿示例中的每个实体创建一个独立的矩形框，如下图所示。关系图 - 数据对象 在第21页显示如何通过关系使实体成为一体。

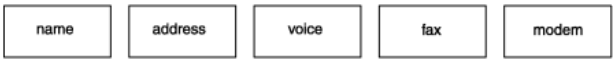


图 2: 个人电话号码簿示例中的实体

## 定义关系

在选择数据库实体之后，您必须考虑它们之间的关系。关系并不总是明显的，但必须找到所有值得记录的关系。确保找到所有关系的唯一方法是详尽地列出所有可能的关系。考虑每一对实体 *A* 和 *B* 并询问“*A* 与 *B* 之间是什么关系？”

关系是两个实体之间的关联。通常，连接两个实体的动词或介词意味着存在关系。实体之间的关系是从连接、存在依赖性和基数等方面描述的。

### 连接

连接指的是实体实例的数目。实体实例是实体的特定出现形式。下图显示三种连接类型，分别是一对一（写作 1:1）、一对多（写作 1:n）和多对多（写作 m:n）。

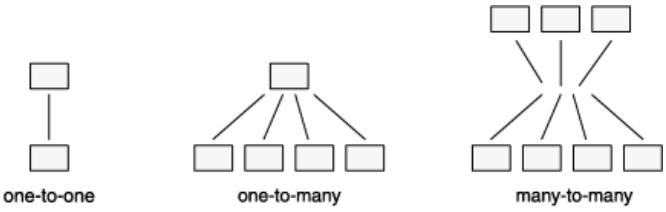


图 3: 关系中的连接

例如：在电话号码簿示例中，地址可以与多个名称相关联。所以，地址与名称实体之间的关系的连接就是一对多 (1:n)。

### 存在依赖性

存在依赖性描述关系中的实体是可选的还是必需的。请分析商业规则来标识实体是否必须存在于关系中。例如：商业规则可能会指定地址必须与名称相关联。这样的关联指示名称与地址实体之间的关系是必需存在依赖性。可选存在依赖性的示例可以是指示某个人可能有或者没有子女的商业规则。

基数

基数对实体在关系中可以出现的次数加以约束。1:1 关系的基数始终为 1。但 1:*n* 关系的基数是开放的；*n* 可以是任何数字。如果必须对 *n* 设置上限，那么请对关系指定基数。例如，在商店销售示例中，可以限制客户每次可以购买的商品项数。通常使用应用程序或存储过程语言 (SPL) 来设置基数约束。

发现关系

发现关系的一种方便方法是准备一个矩阵，该矩阵先在每一行上列出所有实体的名称，然后在每一列上再次这样做。图 *4:* 反映个人电话号码簿实体的矩阵 在第18页中的矩阵反映个人电话号码簿的实体。

| | name | address | number (voice) | number (fax) | number (modem) |
|---|---|---|---|---|---|
| name | | | | | |
| address | | | | | |
| number (voice) | | | | | |
| number (fax) | | | | | |
| number (modem) | | | | | |

图 4: 反映个人电话号码簿实体的矩阵

您可以忽略此矩阵的阴影部分。但您必须考虑对角单元格；即，您必须询问“*A* 与另一个 *A* 之间存在什么关系？”这个问题。在此模型中，答案始终是“none”。在不同的 name 之间或者不同的 address 之间不存在关系，至少您不需要在此模型中记录任何关系。当 *A* 与另一个 *A* 之间存在关系时，说明找到了递归关系。（请参阅解析其他特殊关系 在第27页。）

对于答案明显为 none 的所有单元格，请在矩阵中写下 none。图 *5:* 包含初始关系的矩阵 在第18页显示了当前矩阵。

| | name | address | number (voice) | number (fax) | number (modem) |
|---|---|---|---|---|---|
| name | none | | | | |
| address | | none | | | |
| number (voice) | | | none | | |
| number (fax) | | | | none | |
| number (modem) | | | | | none |

图 5: 包含初始关系的矩阵

虽然在此模型中没有实体与其本身相关，但在其他模型中情况并非总是如此。典型的示例是：一个雇员是另一个雇员的经理。另一个示例是在制造业中：一个部件实体是另一个部件的组件。

在其余单元格中，写下行上的实体与列上的实体之间存在的连接关系。下列是可能的关系类型：

- 一对一 (1:1)，在此关系中，对于一个实体 *B*，存在至多一个实体 *A*，并且对于一个实体 *A*，也是存在至多一个实体 *B*。
- 一对多 (1:n)，在此关系中，绝不会存在多个实体 *A*，但可以有若干个实体 *B* 与 *A* 相关（反之亦然）。
- 多对多 (m:n)，在此关系中，可以有若干个实体 *A* 与一个 *B* 相关，并且可以有若干个实体 *B* 与一个 *A* 相关。

一对多关系最为常见。电话号码簿模型显示了一对多和多对多关系。

如图 *5:* 包含初始关系的矩阵 在第18页所示，第一个未填充的单元格表示名称与地址之间的关系。这些实体之间存在什么样的连接？您可能会问自己“可以有多少个名称与一个地址相关联？”。您决定一个名称可以有零个或一个地址，但不能有多个。您在 name 对面及 address 下面写下 0-1，如图 *6: name* 与 *address* 之间的关系 在第19页所示。

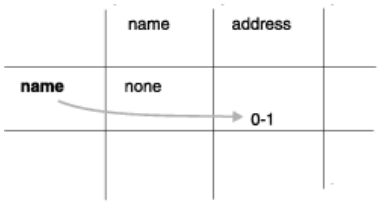


图 6: name 与 address 之间的关系

问问您自己：有多少个地址可以与一个名称相关联。您决定一个地址可以与多个名称相关联。例如：您可以了解到多个人在一家公司工作或两个以上的人的住所在同一地址。

地址可以与零个名称相关联吗？即，可能存在没有任何名称使用的地址吗？您的决定是“是的，可以存在”。在 address 下面及 name 对面，您写下 0-n，如图 *7: address* 与 *name* 之间的关系 在第19页所示。

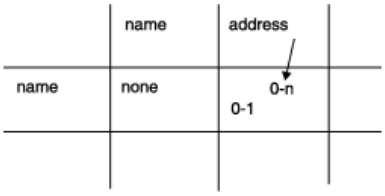


图 7: address 与 name 之间的关系

如果您决定除非地址与至少一个名称相关联否则不能存在，那么请写下 1-n 而不是 0-n。

当在任何一端将关系的基数限制为 1 时，就是 1:n 关系。在这种情况下，名称与地址之间的关系是 1:n 关系。

现在考虑图 *5:* 包含初始关系的矩阵 在第18页中的下一个单元格：名称与语音电话号码之间的关系。名称可以与多少个语音电话号码相关联，是一个还是多个？当您查看电话号码簿时，您会看到通常为某个人记录多个电话号码。一个繁忙的推销员可能会有家庭电话号码、办公室电话号码、寻呼机号码和车载电话号码。但也可能有不带相关联电话号码的名称。您在 name 对面及 number (voice) 下面写下 0-n，如图 *8: name* 与 *number* 之间的关系 在第19页所示。

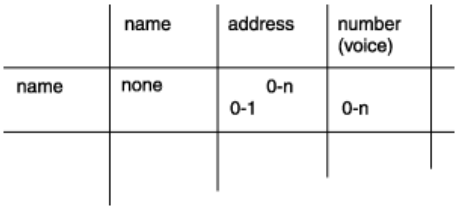


图 8: name 与 number 之间的关系

此关系的另一端是什么？有多少个名称可以与一个语音电话号码相关联？您决定只有一个名称可以与一个语音电话号码相关联。电话号码可以与零个名称相关联吗？您决定除非某人使用某个电话号码，否则不需要记录该电话号码。您在 number (voice) 下面及 name 对面写下 1，如图 *9: number* 与 *name* 之间的关系 在第20页所示。

| | name | address | number (voice) |
|---|---|---|---|
| name | none | 0-n<br>0-1 | 1<br>0-n |
| | | | |

图 9: number 与 name 之间的关系

要以同一方式填写矩阵的其余部分，请考虑下列因素：

- 一个名称可以与多个传真号码相关联；例如：一间公司可以有若干台传真机。反过来，一个传真号码可以与多个名称相关联；例如：若干个人员可以使用同一个传真号码。
- 调制解调器号码必须只与一个名称相关联。（这是为了使示例复杂化而作的任意规定；请将其作为一项设计需求予以接受。）但是，一个名称可以有多个相关联的调制解调器号码；例如：一台公司计算机可以带有若干拨号线路。
- 虽然现实世界中的语音电话号码与地址之间、调制解调器号码与地址之间以及传真号码与地址之间存在某种关系，但这些关系都不必记录在此模型中。通过 *name*，已存在间接的关系。

图 *10:* 已完成的电话号码簿矩阵 在第20页显示完成后的矩阵。

| | name | address | number (voice) | number (fax) | number (modem) |
|---|---|---|---|---|---|
| name | none | 0-n<br>0-1 | 1<br>0-n | 1-n<br>0-n | 1<br>0-n |
| address | | none | none | none | none |
| number (voice) | | | none | none | none |
| number (fax) | | | | none | none |
| number (modem) | | | | | none |

图 10: 已完成的电话号码簿矩阵

矩阵显示的其他决定包括传真号码与调制解调器号码之间、语音电话号码与传真号码之间或语音电话号码与调制解调器号码之间不存在任何关系。

您可能会不赞同其中某些决定（例如：不支持语音电话号码与调制解调器号码之间的关系）。但是，为了实现本示例，这些就是我们的商业规则。

关系图 - 关系

现在，保存您在本节创建的矩阵。您将在关系图 - 数据对象 在第21页中学习如何创建 E-R 图。

## 标识属性

实体包含属性，属性是特征或修饰符、性质、数量或特色。属性是关于实体的事实或不可分解的信息部分。当您将实体表示为表后，其属性将作为新列添加到模型中。

在可以标识数据库属性之前，必须标识实体。在确定实体之后，问问自己“我需要了解每个实体的哪些特征？” 例如：在 *address* 实体中，您可能需要有关 *street*、*city* 和 *zip code* 的信息。*address* 实体的这些特征中的每一个都将成为属性。

选择实体的属性

要选择属性，请确定属性具有以下性质：

- 它们是重要的。

  只包括对数据库用户有用的属性。

- 它们是直接的，而不是派生的。

  可以从现有属性派生（例如：通过 SELECT 语句中的表达式）的属性不应该作为该模型的一部分。派生数据会使数据库的维护复杂化。

  在设计的后面阶段，您可以考虑添加派生属性来改进性能，但在此阶段，请将它们排除。有关如何改进数据库服务器性能的信息，请参阅《*SinoDB*® 性能指南》。
- 它们是不可分解的。

  属性只可以包含单个值，而不能包含列表或重复组。必须将组合值分离到单独的属性中。
- 它们包含同一类型的数据。

  例如：在生日属性中，您只应输入日期值，而不应输入名称或电话号码。

属性的定义规则与列的定义规则相同。有关如何定义列的信息，请参阅对列进行约束 在第24页。

将下列属性添加到电话号码簿示例以生成图 *15:* 电话号码簿示例的初步实体关系图 在第23页所示的图：

- 将街道、城市、州和邮政编码添加到 *address* 实体。
- 将生日、电子邮件地址、周年纪念日和子女的名字添加到 *name* 实体。
- 将类型添加到 *voice* 实体以区分车载电话、家庭电话和办公室电话。语音电话号码只能与一种语音类型相关联。
- 将传真机运作的时间添加到 *fax* 实体。
- 将调制解调器是支持 9600 波特率、14400 波特率还是 28800 波特率添加到 *modem* 实体。

列表属性

现在，列出电话号码簿示例的属性以及您认为其所属的实体。列表的外观应如下图所示。

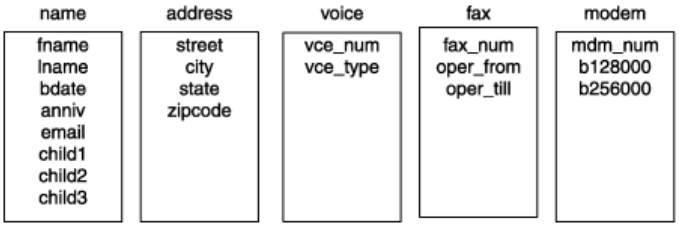


图 11: 电话号码簿示例的属性

关于实体出现形式

实体出现形式是一个附加的数据对象。表中的每一行都表示实体的单一特定出现形式。例如：如果 *customer* 是实体，那么 customer 表表示客户的想法；在该表中，每一行都表示一个特定的客户，如 Sue Smith。请记住，实体成为表，属性成为列，实体出现形式成为行。

## 关系图 - 数据对象

现在，您已经了解并理解数据库中的实体和关系，这是关系数据库设计过程中最重要的部分。在确定实体和关系之后，一种可在数据库设计期间显示您思考过程的方法可能会有帮助。

大多数数据建模方法提供了一些方法来以图形方式显示实体和关系。SinoDB® 文档使用最初由 C. R. Bachman 开发的 E-R 图方法。E-R 图有以下用途。这些图能够：

- 对组织的信息需求建模
- 标识实体及其关系
- 提供数据定义的起始点（数据流图）
- 为应用程序开发者以及数据库和系统管理员提供优良的文档源
- 创建数据库的逻辑设计，此逻辑设计可转换为物理模式

存在若干种不同样式的 E-R 图。如果您已经有了喜欢的样式，请使用该样式。图 *12:* 实体关系图的符号 在第22页显示了样本 E-R 图。

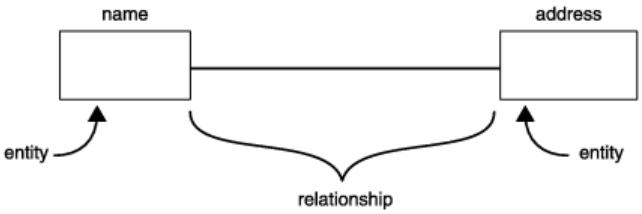


图 12: 实体关系图的符号

在 E-R 图中，一个框表示一个实体。线表示连接实体的关系。此外，图 *13:* 实体关系图中关系的各个部分 在第22页显示如何使用图形项来显示关系的下列特征：

- 关系链接上的圆表示关系中的可选性（可以出现零个实例）。
- 关系链接上的小竖线表示该实体只可以有一个实例与另一个实体相关联（将该竖线看作 1）。
- 末端分叉表示关系中的许多关系。

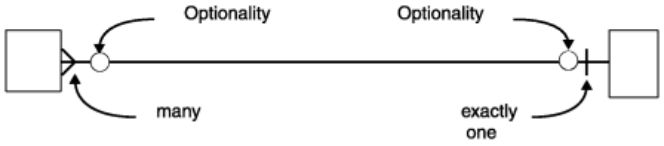


图 13: 实体关系图中关系的各个部分

## 如何读 E-R 图

您首先从左到右读图，然后从右到左读图。对于下图中的 name-address 关系，读这些关系的方式如下：name 可以与零个或只可以与一个 address 相关联；address 可以与零个、一个或许多 name 相关联。

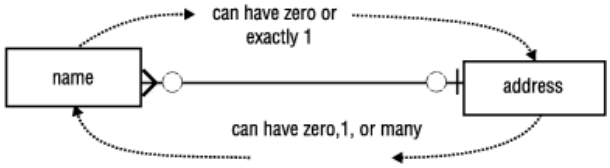


图 14: 读实体关系图

## 电话号码簿示例

下图显示电话号码簿示例并包括实体、关系和属性。此图包括您使用矩阵建立的关系。在研究图符号之后，请将下图中的 E-R 图与图 *10:* 已完成的电话号码簿矩阵 在第20页中的矩阵进行比较。请您自行验证两幅图中的关系是否相同。

在最初设计模型时，矩阵（如图 *10:* 已完成的电话号码簿矩阵 在第20页）是非常有用的工具，因为当您填写矩阵时，会迫使您思考每一种可能的关系。但是，这些关系也出现在类似下图所示的图中，当您检查现有模型时，这种类型的图可能读起来更容易。

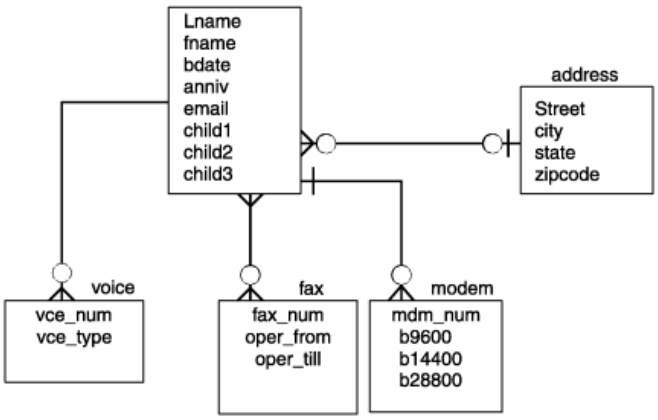


图 15: 电话号码簿示例的初步实体关系图

完成图之后

本章的其余部分描述如何执行下列任务：

- 将实体、关系和属性转换为关系结构。
- 解析 E-R 数据模型。
- 将 E-R 数据模型规范化。

实现关系数据模型 在第41页说明如何根据 E-R 数据模型创建数据库。

## 将 E-R 数据对象转换为关系结构

到目前为止您所了解的所有数据对象（实体、关系、属性和实体出现形式）都转换为 SQL 表，也就是表、列和行之间的连接。数据库的表、列和行必须满足定义表、行和列 在第23页中的规则。

在您将数据对象规范化之前，请检查它们是否满足这些规则。要将数据对象规范化，请分析实体、关系和属性之间的相关性。规范化数据模型 在第27页对规范化进行了说明。

在您将数据模型规范化之后，可使用 SQL 语句来创建基于数据模型的数据库。实现关系数据模型 在第41页描述如何创建数据库并提供电话号码簿示例的数据库模式。

您选择的每个实体都表示为模型中的一个表。表代表的是抽象概念的实体，每一行都表示该实体的个别特定出现形式。另外，实体的每个属性都由表中的一列表示。

下列想法是大多数关系数据模型方法（包括 E-R 数据模型）的基础。在设计数据模型时，请遵循下列规则以便将模型规范化时能够节省时间和减轻工作量。

### 定义表、行和列

您已熟悉了由行和列组成的表的想法。但是，在定义形式数据模型的表时，必须遵循下列规则：

- 行必须是独立的。

  表的每一行都是独立的，并且不依赖于同一个表的任何其他行。因此，表中行的顺序在模型中不重要。即使将表的所有行的顺序打乱成随机顺序，模型也应该仍然正确。

  在实现数据库之后，可以让数据库服务器按特定顺序存储行以提高效率，但该顺序并不影响模型。
- 行必须是唯一的。

  在每一行中，某些列必须包含唯一的值。如果没有任何一列具有此属性，那么作为一个整体的某组列的值在每一行中必须不同。
- 列必须是独立的。

  表中列的顺序在模型中是没有意义的。即使将各列重新排列，模型也应该仍然正确。

在实现数据库之后，那些使用星号来表示所有列的程序和存储查询取决于列的最终顺序，但该顺序不影响模型。

- 列值必须是单一的。

  列只可以包含单个值，而不能包含列表或重复组。必须将组合值分离到单独的列中。例如：如果您决定将人员的名和姓看作独立的值，就像本章中的示例所显示的，那么姓名必须位于独立的列中，而不能位于单个 name 列中。
- 每一列都必须具有唯一的名称。

  同一个表中的两列不能共享同一个名称。但是，可以有多个包含相似信息的列。例如：电话号码簿示例中的 name 表包含子女姓名的列。可以将每个列命名为 child1 和 child2 等等。
- 每一列都必须包含单一类型的数据。

  列必须包含具有相同数据类型的信息。例如：标识为整数的列必须只包含数字信息，而不能包含名称中的字符。

如果您以前只使用过作为数组或顺序文件组织的数据，那么这些规则似乎有点奇怪。然而，关系数据库理论指出只利用遵循这些规则的表、行和列就可以表示所有类型的数据。当您有了少许经验之后，您就会下意识地遵循这些规则了。

#### 对列进行约束

当使用 CREATE TABLE 语句来定义表和列时，您会对每一列进行约束。这些约束指定要让该列包含字符还是数字、要让日期使用的格式以及其他约束。当域标识属性可以具有的有效值的集合时，会描述这些约束。

#### 域特征

创建表时，您定义列的域特征。列可以包含下列域特征：

- 数据类型（INTEGER、CHAR、DATE 等）
- 格式（例如：yy/mm/dd）
- 范围（例如：1000 到 5400）
- 含义（例如：序列号）
- 允许的值（例如：只有等级 S 或 U）
- 唯一性
- 支持 Null
- 缺省值
- 引用约束

有关如何定义域的信息，请参阅选择数据类型 在第29页。有关如何创建表和数据库的信息，请参阅实现关系数据模型 在第41页。

## 确定表的键

表的列要么是键列要么是描述符列。键列是能够唯一标识表中特定行的列。例如：每个雇员的社会保险号是唯一的。描述符列指定表中特定行的非唯一特征。例如：两个雇员可以有相同的名字 Sue。名字 Sue 是雇员的非唯一特征。在表中，键的主要类型是主键和外键。

创建表时，请指定主键和外键。主键和外键用于在物理上关联表。下一个任务是指定每个表的主键。也就是说，必须标识表的某个可量化特征，以便区分每一行与所有其他各行。

#### 主键

表的主键是一个列，其值在每一行中都不同。因为不相同，所以每一行都是唯一的。如果不存在任何一个这样的列，那么主键是两个或更多列的组合，当这些列的值放到一起时，它们在每一行中都不同。

模型中的每个表都必须有主键。此规则自动遵循所有行都必须是唯一的规则。如果有必要的话，主键由所有的列组合而成。不应该将长字符串用作主键。

为了提高效率，主键应该是以下其中一种类型：

- 数字（INT 或 SMALLINT）
- 序列（BIGSERIAL、SERIAL 或 SERIAL8）
- 短字符串（如用作代码的短字符串）。

主键列中绝不允许 NULL 值。NULL 值是不可比较的；即，不能说它们是相似的或不同的。因此，它们不能使某一行对于其他各行来说是唯一的。如果某一列允许 NULL 值，那么它不能作为主键的一部分。定义 PRIMARY KEY 约束时，数据库服务器也会对同一列或组成主键的同一组列静默创建 NOT NULL 约束。

某些实体具有现成的主键，如目录代码或标识号，它们是在模型外部定义的。有时可将多个列或一组列用作主键。有资格成为主键的所有列或组都称为候选键。由于所有候选关键字的唯一性使其在 SELECT 操作中具有可预见性，因此这些候选关键字都不起作用。

复合键

一些实体缺少具有可靠唯一性的特征。不同的人可以有完全相同的姓名；不同的书籍可以有完全相同的标题。通常，您可以找到作为主键的属性组合。例如：人们很少会有完全相同的姓名和完全相同的地址，不同的书籍也很少具有完全相同的标题、作者和出版日期。

系统指定键

系统指定的主键通常优于复合键。系统指定的键是当某个实体最初进入数据库时与该实体的每个实例相连接的数字或代码。系统指定的键中最容易实现的是序列号，因为数据库服务器可以自动生成它们。SinoDB® 数据库服务器对序列号提供了 SERIAL、SERIAL8 和 BIGSERIAL 数据类型。但是，使用数据库的人可能不喜欢单纯的数字代码。其他代码可以基于实际的数据；例如：雇员标识代码可以基于人员的姓名首字母与其雇佣日期的数字的组合。在电话号码簿示例中，对 name 表使用系统指定的主键。

外键（连接列）

外键是一个表中的一列或一组列，包含与另一个表中的主键相匹配的值。外键用来连接表。下图显示了演示数据库中的 customer 和 orders 表的主键和外键。

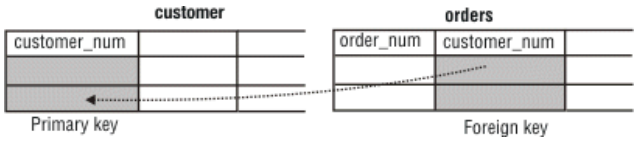


图 16: customer-order 关系中的主键和外键

> 提示： 为了便于维护和使用表，请务必对主键和外键使用合适的名称使关系显而易见。在 图 *16: customer-order* 关系中的主键和外键 在第25页中，主键和外键列具有相同的名称 customer_num。或者，也可以将 图 *16: customer-order* 关系中的主键和外键 在第25页中的列命名为 customer_custnum 和 orders_custnum，这样每个列就都具有不同的名称。

模型中会标注所有出现外键的位置，因为其存在会限制从表中删除行的能力。在可以安全删除行之前，必须删除所有通过外键引用该行的行，或者必须用特殊的语法来定义关系，以允许使用单个删除命令来从主键和外键列中删除行。数据库服务器不允许违反引用完整性的删除操作。

为了保持引用完整性，在删除外键行引用的主键之前，应删除所有那些外键行。如果对数据库施加引用约束，那么数据库服务器不允许删除带有匹配外键的主键。数据库服务器也不允许添加未引用现有主键值的外键值。有关参照完整性的更多信息，请参阅《*SinoDB® SQL* 指南: 教程》。

为电话号码簿图添加键

下图显示对主键和外键的初始选择。此图反映了一些重要的决定。

对于 name 表，选择主键 rec_num。rec_num 的数据类型是 SERIAL。rec_num 的值由系统生成。如果查看 name 表中的其他列（或属性），您会看到与列相关联的数据类型大部分是基于字符。在这些列中，没有任何一个能单独作为主键的合适候选键。如果将表的元素组成复合键，那么会创建繁琐的键。SERIAL 数据类型提供了一个键，您也可以使用该键将其他表连接到 name 表。

voice、fax、modem 和 address 表都通过 rec_num 键连接到 name。

对于 voice、fax 和 modem 表，电话号码都用作主键。address 表包含特殊列 (id_num)，该列除了用作主键没有其他用途。因为如果 id_num 不存在，那么所有其他列将必须一起作为一个复合键使用，以确保不会存在重复的主键。将所有列用作主键将是非常低效和混乱的。

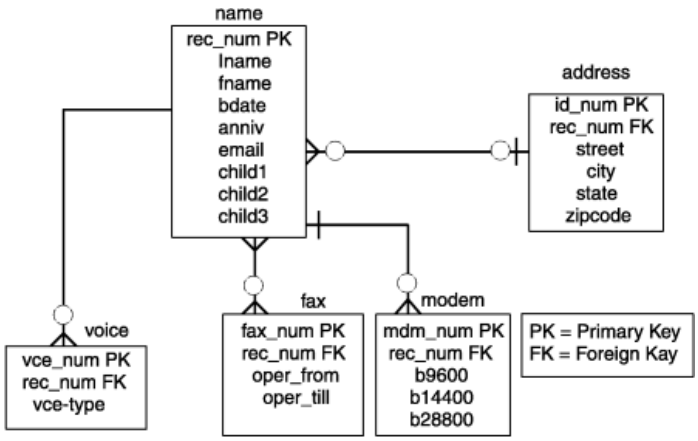


图 17: 添加了主键和外键的电话号码簿图

## 解析关系

良好数据模型的目标是创建一个为数据库服务器提供快速访问的结构。要进一步改进电话号码簿数据模型，您可以解析关系并将数据模型规范化。本节阐述如何以及为何要解析数据库关系。规范化数据模型 在第27页说明如何将数据模型规范化。

### 解析 m:n 关系

多对多 (m:n) 关系使模型和应用程序开发过程更加复杂并容易令人混淆。解析 m:n 关系的关键是将两个实体分开并用第三个相交实体在它们之间创建两个一对多 (1:n) 关系。相交实体通常包含来自两个相连接的实体的属性。

要解析 m:n 关系，请再次分析商业规则。您是否已准确绘制关系图？如图 *17:* 添加了主键和外键的电话号码簿图 在第26页所示，电话号码簿示例在 *name* 与 *fax* 实体之间具有 m:n 关系。商业规则指出“一个人可以有零个、一个或许多传真号码；一个传真号码可以用于若干个人”。根据我们先前为 *voice* 实体选择作为主键的内容，存在 m:n 关系。

*fax* 实体中存在一个问题，其原因在于电话号码（已被指定为主键）可以在 *fax* 实体中出现多次；这违反了主键的条件。请记住，主键必须是唯一的。

要解析这种 m:n 关系，可以在 *name* 与 *fax* 实体之间添加相交实体，如图 *18:* 解析多对多 *(m:n)* 关系 在第27页所示。新的相交实体 *faxname* 包含两个属性：fax_num 和 rec_num。此实体的主键是这两个属性的组合。从个别来看，每个属性都是一个外键，它引用作为其来源的表。由于一个名称可以与许多传真号码相关联，并且在另一个方向上每个 faxname 组合可以与一个 rec_num 相关联，所以 name 与 faxname 表之间的关系是 1:n。由于每个号码可以与许多 faxname 组合相关联，所以 fax 与 faxname 表之间的关系是 1:n。

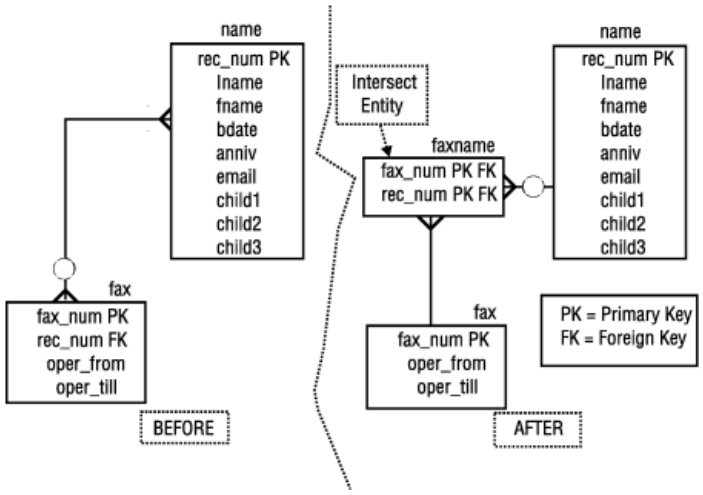


图 18: 解析多对多 (m:n) 关系

## 解析其他特殊关系

您可能会遇到其他可能会导致数据库无法平稳运行的特殊关系。以下内容描述了这些关系：

复杂关系

复杂关系是三个或更多实体之间的关联。要使该关系存在，所有实体都必须存在。为了降低这种复杂性，请将所有复杂关系重新分类为一个实体，通过二元关系与每个原始实体相关联。

递归关系

递归关系是同一实体类型的多个出现形式之间的关联。这些类型的关系不会经常出现。递归关系的示例包括物料单（部件由子部件组成）和组织结构（雇员管理其他雇员）。您可能无法解析递归关系。关于递归关系的扩展示例，请参阅《*SinoDB® SQL* 指南*:* 教程》。

冗余关系

当两个或更多关系表示同一概念时，存在冗余关系。冗余关系提高了数据模型的复杂程度，并会导致开发者在模型中不正确地放置属性。冗余关系可能会作为 E-R 图中的重复条目出现。例如：可能有两个包含相同属性的实体。要解析冗余关系，请复查数据模型。有没有多个包含相同属性的实体？可能需要向模型添加一个实体来解决冗余性。《*SinoDB®* 性能指南》包含与数据模型中冗余性相关的其他主题的信息。

# 规范化数据模型

本章中的电话号码簿示例似乎是一个不错的模型。此时，可以将其实现到数据库中，但此示例之后可能会导致应用程序开发和数据处理操作出现问题。规范化是一种应用一组规则将属性与实体相关联的形式方法。

将数据模型规范化时，可以实现下列目标。您可以：

- 使设计具有更大的灵活性。
- 确保将属性置于正确的表中。
- 降低数据冗余度。
- 提高程序员的效率。
- 降低应用程序维护成本。
- 使数据结构的稳定性达到最高程度。

规范化由若干个步骤组成，用于将实体简化为更合适的物理属性。这些步骤称为规范化规则，也称为范式。存在若干个范式；本章包含有关前三个范式的信息。每个范式都比上一个范式对数据更具约束性。因此，在可以进行第二范式之前必须先达成第一范式，并且，在可以进行第三范式之前必须先达成第二范式。

## 第一范式

如果实体不包含重复组，那么该实体处于第一范式。就关系而言，如果表不包含重复列，那么该表处于第一范式。重复列会降低数据的灵活性、浪费磁盘空间和导致更难搜索数据。在以下电话号码簿示例中，name 表包含重复列 child1、child2 和 child3。

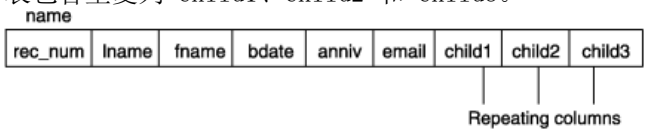


图 19: 规范化之前的 Name 实体

您可以在当前表中发现一些问题。这个表总是在磁盘上为三个子女记录保留空间，而无论该人员是否有子女。可以记录的最大子女数是 3，但您的一些熟人可能有四个或更多的子女。要查找特定的子女，就必须在每一行中搜索所有这三列。

要消除重复列并使该表成为第一范式，请将该表分为两个表。将重复列放到其中一个表中。两个表之间的关联通过主键与外键的组合建立。因为 name 表中不存在关联就不能存在子女，所以可使用外键 rec_num 来引用 name 表。

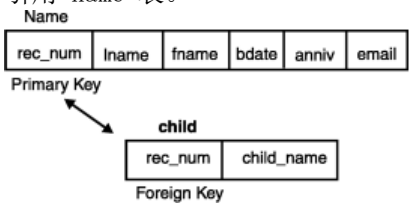


图 20: Name 实体达到第一范式

现在，请检查图 *17: 添加了主键和外键的电话号码簿图* 在第26页中的电话号码簿结构，以找出不属于第一范式的组。因为 b9600、b14400 和 b28800 列是重复列，所以 name-modem 关系不是处于第一范式。向 modem 表添加名为 b_type 的新属性，以包含 b9600、b14400 和 b28800 的出现形式。下图显示按照第一范式进行规范化的数据模型。

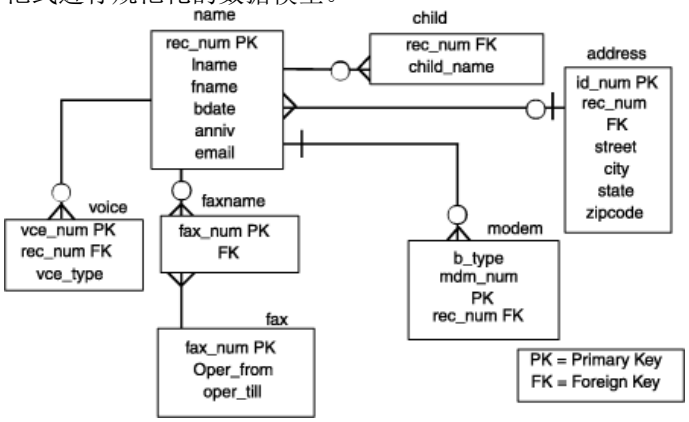


图 21: 个人电话号码簿的数据模型

## 第二范式

如果实体的所有属性都依赖于整个（主）键，那么该实体处于第二范式。就关系而言，表中的每个列都必须在功能上依赖于该表的整个主键。功能依赖性指示两个不同列中的值之间存在链接。

如果某个属性的值依赖于某列，那么在该列的值发生变更时，该属性的值必然变更。该属性是功能依赖于该列。下列说明使这一点更为明确：

- 如果该表具有一个单列主键，那么该属性必须依赖于该主键。
- 如果该表具有复合主键，那么该属性必须依赖于该主键的全部各列中作为一个整体的值，而不是依赖于那些列的其中一列或其中某些列。
- 如果该属性也依赖于其他列，那么它们必须是候选键的列；也就是在每一行中都唯一的列。

如果不将模型转换为第二范式，那么有数据冗余和难以更改数据的风险。要将第一范式表转换为第二范式表，请移除不依赖于主键的列。

### 第三范式

如果实体满足第二范式且其所有属性都不以传递方式依赖于主键，那么该实体为处于第三范式。传递相关性表示描述符键属性不仅依赖于整个主键，还依赖于其他描述符键属性，而这些描述符键属性又依赖于主键。就 SQL 而言，第三范式表示在表中没有任何列依赖于某个描述符列，而该描述符列又依赖于主键。

要转换为第三范式，请移除那些依赖于其他描述符键属性的属性。

### 规范化规则的摘要

本节说明以下范式：

第一范式

如果表不包含重复列，则该表处于第一范式。

第二范式

如果表满足第一范式且只包含依赖于整个（主）键的列，则该表处于第二范式。

第三范式

如果表满足第二范式且只包含以非传递方式依赖于主键的列，则该表处于第三范式。

按照 E. F. Codd（关系数据库的发明者）的规定，遵循这些规则的话，模型中的所有表皆会处于第三范式。如果表不是处于第三范式，那么不是模型中存在冗余数据，就是在尝试更新表时出现问题。

如果在模型中找不到位置放置遵循这些规则的属性，那么可能是发生了下列其中一个错误：

- 属性定义不明确。
- 该属性是派生的，不是直接的。
- 该属性实际上是实体或关系。
- 模型中缺少某些实体或关系。

# 选择数据类型

准备数据模型之后，必须将其作为数据库和表来实现。要实现数据模型，首先为每个列定义域或一组数据值。本章包含有关定义列数据类型和约束时必须做出的决定的信息。

实现关系数据模型 在第41页包含有关第二个步骤如何使用 CREATE DATABASE 和 CREATE TABLE 语句来实现模型并使用数据填充表的信息。

## 定义域

要完成构建关系数据模型 在第14页描述的数据模型，您必须为每个列定义域。列的域描述约束并标识属性（列）可以具有的有效值集合。

域的用途是保护模型中数据的语义完整性；也就是说，确保数据切合实际地反映事实。如果能够用名称来替换电话号码或者可以在只有整数才是有效值的位置中输入小数，那么数据模型的完整性是有风险的。

要定义域，请指定在数据值可以作为域的一部分之前必须满足的约束。要指定列域，请使用下列约束：

- 数据类型
- 缺省值
- 检查约束
- 引用约束

# 数据类型

对任何列的第一个约束都是该列的数据类型中隐式的约束。选择数据类型时，您就对列进行了约束，使其只包含该数据类型可以表示的值。

每种数据类型都表示特定类型的信息，而不表示其他类型的信息。列的正确数据类型应该表示适合该列的所有数据值，但是尽可能少包含不适合该列的值。

本章描述内置数据类型。

有关 SinoDB® 支持的扩展数据类型的信息，请参阅在 *SinoDB* 中创建和使用扩展数据类型 在第82页。

## 选择数据类型

表中的每一列都必须具有数据类型。由于下列原因，数据类型的选择至关重要：

- 它建立该列可以存储的有效数据项的集合。
- 它决定可以对数据执行的操作类型。

  例如：不能对使用字符数据类型定义的列应用聚合函数，如 SUM。
- 它决定每个数据项在磁盘上占用多少空间。

  容纳数据项所需的空间对于小型表的重要程度与对于带有数十万行的表的重要程度是不相同的。当表的行数有如此之多时，4 字节与 8 字节数据类型之间的差别可能极为重要。

下图显示了决策树，此决策树概述了在内置数据类型之间所作的选择。下列各章节对这些选择作了说明。

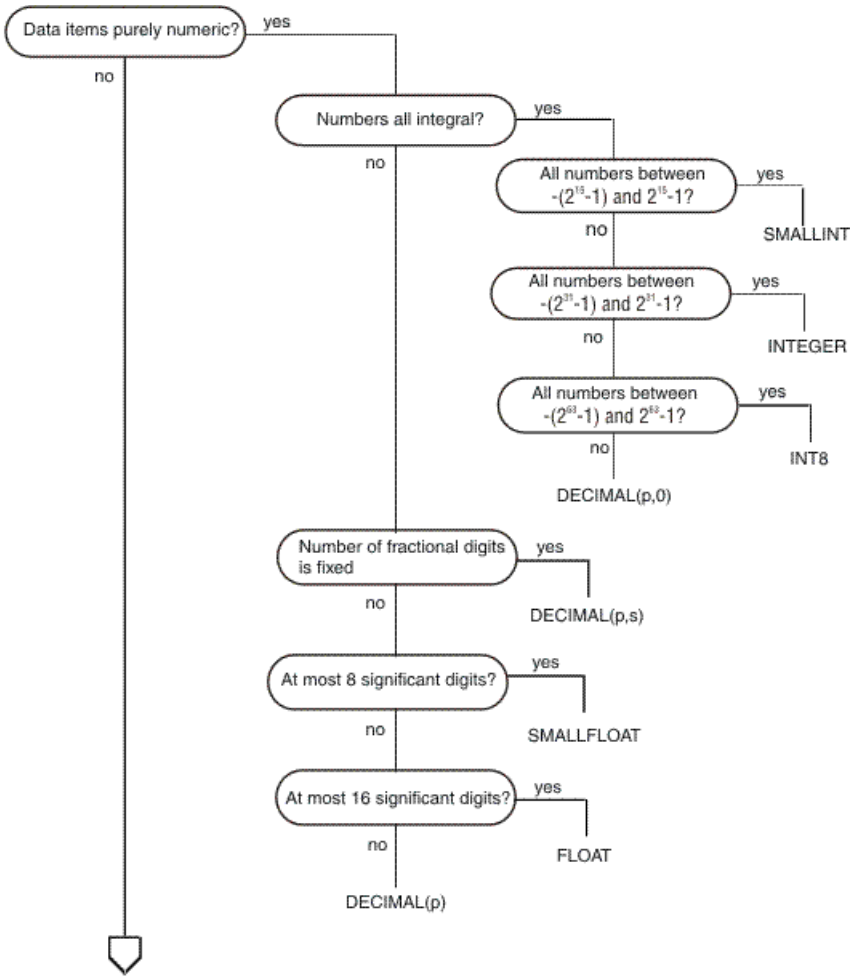


图 22：选择数据类型

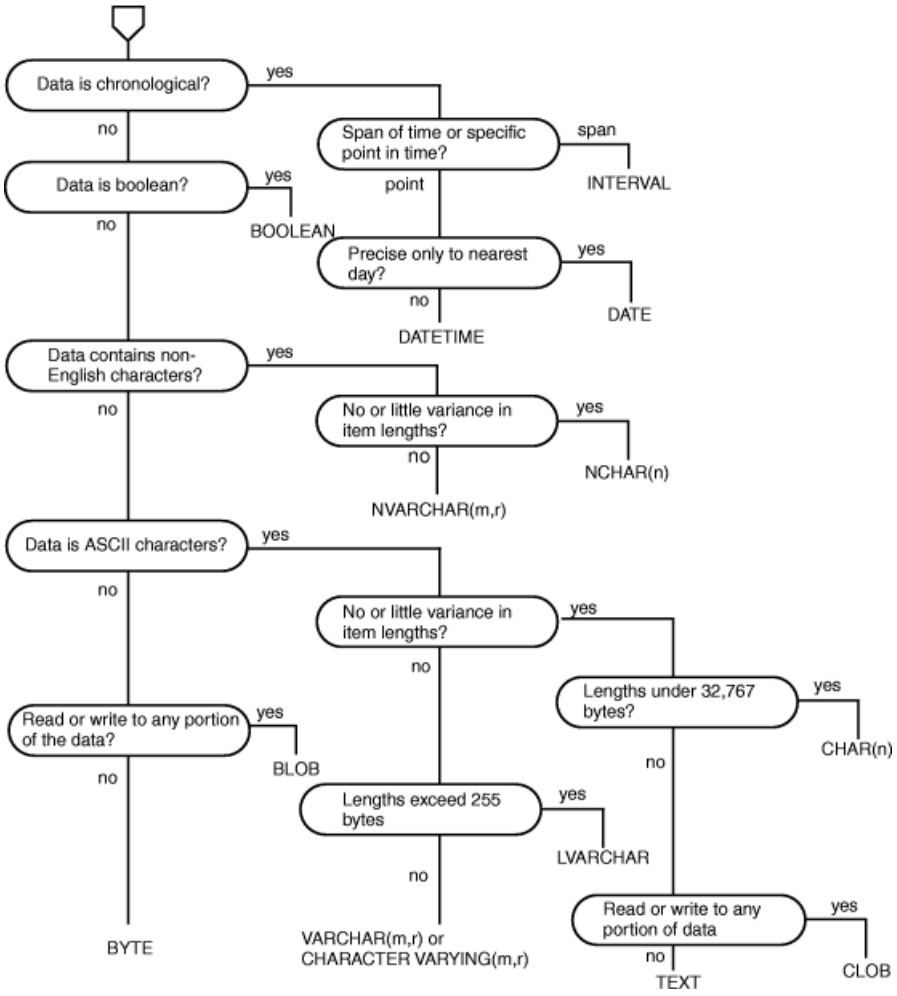


图 23: 选择数据类型（续）

## 数字类型

一些数字数据类型最适合用于计数器和代码，一些最适合用于工程数量，一些最适合用于货币。

计数器和代码：BIGINT、INT8、INTEGER 和 SMALLINT

INTEGER 和 SMALLINT 数据类型可以存放小整数。当您事先知道要存储的最大值和最小值时，这两种数据类型适合包含计数、序号、数字标识代码或任何范围的整数的列。

这两种数据类型都是作为带符号二进制整数存储的。INTEGER 值有 32 位，可以表示从 $-2^{31}-1$ 到 $2^{31}-1$ 的整数。

SMALLINT 值只有 16 位，可以表示从 -32767 到 32767 的整数。

INT 和 SMALLINT 数据类型具有下列优点：

- 占用很少的空间（对于 SMALLINT，每个值占用 2 个字节，对于 INTEGER，每个值占用 4 个字节）。
- 可以执行算术表达式（如 SUM 和 MAX）和排序比较。

使用 INTEGER 和 SMALLINT 的缺点是其可以存储值的范围有限。数据库服务器不能存储超出整数容量的值。当然，当您知道要存储的最大值和最小值时，这样的超限不是问题。

如果必须在 INTEGER 存储更大范围的值，那么可以使用 BIGINT 或 INT8。这些数据类型具有下列优点：

- 可以容纳大范围的值。（范围从 $-(2^{63}-1)$ 到 $2^{63}-1$ 的整数。）
- 可以执行算术表达式（如 SUM 和 MAX）和排序比较。BIGINT 在存储和计算效率方面优于 INT8。

使用 BIGINT 或 INT8 的缺点是其使用的磁盘空间大于 INTEGER。实际大小取决于平台的字长度。一个 INT8 或 SERIAL8 值需要 10 个字节的存储空间。BIGINT 和 BIGSERIAL 值需要 8 个字节的存储空间。

自增序列：BIGSERIAL、SERIAL 和 SERIAL8

SERIAL 数据类型的特点是具有非零的正 INTEGER 范围。同样，BIGSERIAL 和 SERIAL8 数据类型也具有非零正 INT8 范围的特点。每当将新行插入表中时，数据库服务器都将自动生成 BIGSERIAL、SERIAL 或 SERIAL8 列的新值。

一个表只能有一个 SERIAL 列，但是可以有一个 SERIAL 列和一个 SERIAL8 列或 BIGSERIAL 列。因为数据库服务器将生成值，所以新行中的序列值总是不相同的，即使多个用户同时添加行。由于普通程序很难在那些情况下生成唯一的数字代码，所以此服务很有用。（但是，SinoDB® 也支持序列对象，序列对象也可通过 CURRVAL 和 NEXTVAL 运算符支持该功能。有关序列对象的更多信息，请参阅《*SinoDB® SQL* 指南: 语法》中对 CREATE SEQUENCE 的描述。

SERIAL 数据类型最多可以生成 $2^{31}-1$ 个正整数。因此，数据库服务器将使用所有的正数序列号，直到它在表中插入 $2^{31}-1$ 行为止。然而，对于大多数用户来说，并不用担心发生用完正数序列号的问题，因为单个应用程序需要在 68 年内每秒插入一行，或者 68 个应用程序需要在一年内每秒插入一行，才能用完所有正数序列号。但是，如果使用了所有的正数序列号，那么数据库服务器将回绕并开始生成以 1 开头的整数值。

BIGSERIAL 和 SERIAL8 数据类型最多可以生成 $2^{63}-1$ 个正整数。使用合理的起始值，实际上不可能导致这些类型的值在插入期间回绕。

对于这些数据类型，所生成数字的序列总是递增的。从表中删除行之后，所删除行的序列号将不重复使用。根据这些类型的列进行排序的行是按其创建顺序返回的。

可以在 CREATE TABLE 语句中指定 BIGSERIAL、SERIAL 或 SERIAL8 列中的初始值。这使得在不同的表中可以生成系统指定的键的不同子序列。stores_demo 数据库使用了此技术。在 stores_demo 中，客户号从 101 开始，而订单号从 1001 开始。如果这家小企业注册的客户数不超过 899 个，那么所有客户号都将为三位数，并且订单号为四位数。

BIGSERIAL、SERIAL 或 SERIAL8 列不会自动成为唯一列。如果要完全确保不出现重复的序列号，那么必须应用唯一约束（请参阅使用 *CREATE TABLE* 在第42页）。如果使用 DB-Access 中的交互式模式编辑器来定义表，那么将对任何 BIGSERIAL、SERIAL 或 SERIAL8 列自动应用唯一约束。

BIGSERIAL、SERIAL 和 SERIAL8 数据类型具有下列优点：

- 它们提供了一种很方便的方法来生成系统指定的键。
- 它们生成唯一的数字代码，即使当多个用户更新表时也是如此。
- 不同的表可以使用不同范围的数字。

BIGSERIAL、SERIAL 和 SERIAL8 数据类型具有下列缺点：

- 一个表只能有一个 SERIAL 数据类型列且只能有一个 SERIAL8 或 BIGSERIAL 数据类型列。
- 这些数据类型只能生成任意非空的正整数。

有关 SERIAL、SERIAL8 和 BIGSERIAL 数据类型在表层次结构中的使用和行为的信息，请参阅表层次结构中的 *SERIAL* 类型 在第102页。

变更下一个 BIGSERIAL、SERIAL 或 SERIAL8 数

数据库服务器在创建 BIGSERIAL、SERIAL 或 SERIAL8 列时设置该列的起始值（请参阅使用 *CREATE TABLE* 在第42页）。您可以在以后使用 ALTER TABLE 语句来复位下一个值，即用于下一个插入行的值。

可以将下一个值设置为任何大于当前最大值的值。这样做会在序列中创建间隔。

如果您尝试将下一个值设置为小于列中当前最大值的值，那么虽然不会有错误但是值并没有得到设置。允许下一个值设置为小于列中的部分值，会在某些情况下导致重复值，因此这是不允许的。

近似数：FLOAT 和 SMALLFLOAT

在科学、工程和统计应用程序中，数字通常被认为只有几位的准确度，并且数字的量值与精确位数一样重要。

浮点数据类型就是为这些类型的应用程序设计的。它们可以表示任何数量，可以是小数或整数，并且其数量级范围非常广（从天文级到微观级）。它们可以很容易地表示从地球到太阳的平均距离（1.5 x $10^{11}$ 米）或普朗克常量（6.626 x $10^{-34}$ 焦耳秒）。例如：

```
CREATE TABLE t1 (f FLOAT);
INSERT INTO t1 VALUES (0.00000000000000000000000000000000000001);
INSERT INTO t1 VALUES (1.5e11);
INSERT INTO t1 VALUES (6.626196e-34);
```

存在两种大小的浮点数据类型。FLOAT 类型是在计算机上用 C 语言实现的双精度二进制浮点数。FLOAT 数据类型值通常占用 8 个字节。SMALLFLOAT（也称为 REAL）数据类型是单精度二进制浮点数，通常占用 4 个字节。这两种数据类型之间的主要区别是它们的精度。

浮点数具有下列优点：

- 它们存储非常大和非常小的数字，包括小数。
- 它们以 4 个或 8 个字节表示数字。
- 算术函数（如 AVG 和 MIN）和排序比较对这些数据类型有效。

浮点数的主要缺点是其精度范围之外的各位数都被视为零。

可调整精度浮点：DECIMAL(***p***)

在不符合 ANSI 标准的数据库中，DECIMAL(*p*) 数据类型是与 FLOAT 和 SMALLFLOAT 类似的浮点数据类型。重要区别是指定它保留多少个有效位。*p* 的精度范围为 1 到 32，也就是从低于 SMALLFLOAT 的精度到 FLOAT 精度的两倍。DECIMAL(*p*) 数的量级范围从 $10^{-130}$ 到 $10^{124}$。DECIMAL(*p*) 数使用的存储空间取决于其精度；它们占用 1 + *p*/2 个字节（如果有必要的话，向上舍入到整数）。

但是，在符合 ANSI 标准的数据库中，DECIMAL(*p*) 是小数位为零的定点数据类型，所以如果数据值有 *p* 或更多有效位数，那么 DECIMAL(*p*) 将始终存储精度为 *p* 的整数值。任何小数部分都将被截断。

不要将 DECIMAL(*p*) 数据类型与下一章节说明的 DECIMAL(*p,s*) 数据类型相混淆。DECIMAL(*p*) 数据类型只指定了精度。

与 FLOAT 相比，DECIMAL(*p*) 数据类型具有下列优点：

- 可以设置精度以适合应用程序（从近似到精确）。
- 可以精确表示 32 位的数字。
- 使用的存储空间与数字的精度成比例。
- 无论使用什么主机操作系统，每个 SinoDB® 数据库服务器都支持相同的精度和量级范围。

DECIMAL(*p*) 数据类型具有下列缺点：

- 对 DECIMAL(*p*) 值执行的算术运算和排序的性能在一定程度上低于对 FLOAT 值执行的算术运算和排序的性能。
- 许多编程语言不像支持 FLOAT 和 INTEGER 那样支持 DECIMAL(*p*) 数据格式。当程序从数据库中提取 DECIMAL(*p*) 值时，它可能需要将该值转换为另一种格式才能处理。
- DECIMAL(*p*) 数据类型的格式和值取决于数据库是否符合 ANSI 标准。

固定精度数：DECIMAL 和 MONEY

大多数商业应用程序存储的数字在小数点左右两边有固定位数。例如：美国货币金额在小数点右边保留两位。正常情况下，根据记录的交易类型，您也会知道左边所需的位数：个人预算可能是 5 位，小企业可能是 7 位，国家预算可能是 12 或 13 位。

由于小数点固定在特定的位置（而无论数值多少），所以这些数是定点数。DECIMAL(*p,s*) 数据类型保存十进制数。当指定此类型的列时，将其精度 (*p*) 写作可以存储的总位数（从 1 到 32）。将其小数位 (*s*) 写作位于小数点右边的那些位数。（下图显示了精度与小数位之间的关系。）小数位可以是零，表示只存储整数。当只存储整数时，DECIMAL(*p,s*) 提供了存储多达 32 位整数的方法。

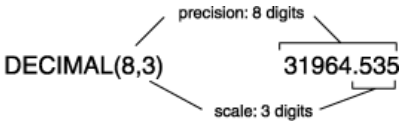


图 24: 定点数中的精度与小数位之间的关系

与 DECIMAL(*p*) 数据类型相似，DECIMAL(*p,s*) 占用的空间与其精度成正比。一个值占用 (*p* +3)/2 个字节（如果小数位是偶数）或 (*p* + 4)/2 个字节（如果小数位是奇数），并且向上舍入至整数个字节。

除了具有一项额外的特征外，MONEY 类型与 DECIMAL(*p,s*) 完全相同。每当数据库服务器将 MONEY 值转换为字符来显示时，都将自动包括货币符号。

与 INTEGER 和 FLOAT 相比，DECIMAL(*p,s*) 的优点是可用精度更高（最高 32 位，而 INTEGER 是 10 位，FLOAT 是 16 位），并且精度和所需的存储空间都可以调整，以适应应用程序。

DECIMAL(*p,s*) 的缺点是算术运算的效率较低，并且许多编程语言不支持此格式的数字。因此，当程序抽取数字时，通常必须将该数字转换为另一数字格式才能处理。

选择货币格式

要自定义此货币格式，请选择适当的语言环境或设置 DBMONEY 环境变量。有关更多信息，请参阅《*SinoDB® SQL* 指南*:* 参考》。

Global Language Support (GLS)

每个国家都有自己显示货币值的方式。当 SinoDB® 数据库服务器显示 MONEY 值时，它是指用户指定的货币格式。缺省语言环境指定美国英语货币格式，具体格式如下：$7,822.45

对于非英语语言环境，可使用语言环境文件的 MONETARY 类别来更改当前格式。有关如何使用语言环境的更多信息，请参阅《*SinoDB® GLS* 用户指南》。

## 时间值数据类型

时间值数据类型记录时间。

DATE 数据类型存储日历日期。DATETIME 可记录从年到几分之一秒的任何精度的时间点。INTERVAL 数据类型存储时间跨度，即持续时间。

相关链接

*《SinoDB SQL 指南: 参考》: INTERVAL 数据类型*

日历日期：DATE

DATE 数据类型存储日历日期。DATE 值实际上是带符号整数，其内容解释为从 1899 年 12 月 31 日午夜开始的整天数。

DATE 格式有足够的精度记录遥远的将来（58,000 个世纪）的日期。负的 DATE 值将解释为纪元日之前的天数；即，DATE 值 -1 表示 1899 年 12 月 30 日。

由于 DATE 值是整数，所以可以在算术表达式中使用。例如：您可以获取 DATE 列的平均值，也可以将 DATE 列加上 7 或 365。另外，还提供了大量专门处理 DATE 值的函数。有关更多信息，请参阅《*SinoDB® SQL* 指南*:* 语法》。

DATE 数据类型是压缩的，每一项占用 4 个字节。可以对 DATE 列快速执行算术函数和比较。

选择日期格式 (GLS)

您可以采用多种方式对日期组件加标点和排序。当应用程序显示 DATE 值时，将引用用户指定的日期格式。缺省语言环境指定美国英语日期格式，具体格式如下：10/25/2001

要自定义此日期格式，请选择适当的语言环境或设置 DBDATE 环境变量。有关更多信息，请参阅《*SinoDB® SQL* 指南*:* 参考》。

对于非缺省语言环境，可以使用 GL_DATE 环境变量指定日期格式。有关如何使用语言环境的更多信息，请参阅《*SinoDB® GLS* 用户指南》。

精确时间点：DATETIME

DATETIME 数据类型存储从公元 1 年开始的时期中的任何时刻。事实上，DATETIME 具有 28 种数据类型，其中每种数据类型的精度都不相同。在定义 DATETIME 列时，请指定其精度。该列可以包含列表中的任何序列：

- 年
- 月
- 日
- 小时
- 分钟
- 秒
- 小数

因此，您可以定义只存储年、只存储月和日或者精确到小时甚至精确到毫秒的日期和时间的 DATETIME 列。下表显示 DATETIME 值的大小范围为 2 到 11 个字节（具体取决于其精度）。

DATETIME 的优点是可以存储特定的日期和时间值。与 DATE 列相比，DATETIME 列通常需要更多的存储空间（这取决于 DATETIME 限定符）。此外，Datetime 的显示格式不灵活。有关如何避开显示格式的信息，请参阅强制 *DATETIME* 或 *INTERVAL* 值的格式 在第36页。

表 1: DATETIME 数据类型的精度

| 精度 | 大小（当 f 是奇数时，将大小取整到下一个整字节） | 精度 | 大小（当 f 是奇数时，将大小取整到下一个整字节） |
|---|---|---|---|
| 年到年 | 3 | 日到小时 | 3 |
| 年到月 | 4 | 日到分钟 | 4 |
| 年到日 | 5 | 日到秒 | 5 |
| 年到小时 | 6 | 日到小数(*f*) | 5 + *f*/2 |
| 年到分钟 | 7 | 小时到小时 | 2 |
| 年到秒 | 8 | 小时到分钟 | 3 |
| 年到小数(*f*) | 8 + *f*/2 | 小时到秒 | 4 |
| 月到月 | 2 | 小时到小数(*f*) | 4 + *f*/2 |
| 月到日 | 3 | 分钟到分钟 | 2 |
| 月到小时 | 4 | 分钟到秒 | 3 |
| 月到分钟 | 5 | 分钟到小数(*f*) | 3 + *f*/2 |
| 月到秒 | 6 | 秒到秒 | 2 |
| 月到小数(*f*) | 6 + *f*/2 | 秒到小数(*f*) | 2 + *f*/2 |
| 日到日 | 2 | 小数到小数(*f*) | 1 + *f*/2 |

使用 INTERVAL 的持续时间

INTERVAL 数据类型存储持续时间，即时间长度。两个 DATETIME 值之间的差就是 INTERVAL，表示这两个值之间的时间跨度。下列示例可能有助于您弄清楚这些差：

- 一个雇员在 1997 年 1 月 21 日开始工作（DATE 或 DATETIME）。
- 她已工作 254 天（一个 INTERVAL 值，也就是 TODAY 函数与起始 DATE 或 DATETIME 值之间的差）。
- 她每天从 0900 点（一个 DATETIME 值）开始工作。
- 她工作 8 小时（一个 INTERVAL 值），其间 45 分钟用来吃午餐（另一个 INTERVAL 值）。
- 她下班时间是 1745 点（她开始工作时的 DATETIME 与两个 INTERVAL 之和）。

与 DATETIME 相似，INTERVAL 是一系列具有不同精度的数据类型。INTERVAL 值可以表示年和月的计数；也可以表示日、小时、分钟、秒或秒的小数的计数；可能的精度有 18 种。INTERVAL 值的大小范围是 2 到 12 个字节，这取决于表 2: *INTERVAL* 数据类型的精度 在第36页显示的公式。

表 2: INTERVAL 数据类型的精度

| 精度 | 大小（将小数大小取整到下一个整字节） | 精度 | 大小（将小数大小取整到下一个整字节） |
|---|---|---|---|
| 年(*p*) 到年 | 1 + *p*/2 | 小时(*p*) 到分钟 | 2 + *p*/2 |
| 年(*p*) 到月 | 2 + *p*/2 | 小时(*p*) 到秒 | 3 + *p*/2 |
| 月(*p*) 到月 | 1 + *p*/2 | 小时(*p*) 到小数(*f*) | 4 + (*p* + *f*)/2 |
| 日(*p*) 到日 | 1 + *p*/2 | 分钟(*p*) 到分钟 | 1 + *p*/2 |
| 日(*p*) 到小时 | 2 + *p*/2 | 分钟(*p*) 到秒 | 2 + *p*/2 |
| 日(*p*) 到分钟 | 3 + *p*/2 | 分钟(*p*) 到小数 (*f*) | 3 + (*p* + *f*)/2 |
| 日(*p*) 到秒 | 4 + *p*/2 | 秒(*p*) 到秒 | 1 + *p*/2 |
| 日(*p*) 到小数(*f*) | 5 + (*p* + *f*)/2 | 秒(*p*) 到小数 (*f*) | 2 + (*p* + *f*)/2 |
| 小时(*p*) 到小时 | 1 + *p*/2 | 小数到小数(*f*) | 1 + *f*/2 |

INTERVAL 值可以是负数或正数。可以对它们进行加减运算，也可以通过乘以或除以某个数将它们缩放。对于 DATE 或 DATETIME，情况并非如此。您可以合理地问“到 4 月 23 日为止的天数的一半是多少？”，但“4 月 23 日的一半是什么？”却不合理。

强制 DATETIME 或 INTERVAL 值的格式

数据库服务器总是按 year-month-day hour:minute:second.fraction 顺序显示 INTERVAL 或 DATETIME 值的组件。数据库服务器不会引用对操作系统定义的日期格式，如格式化 DATE 值时的做法一样。

您可以编写以系统定义的格式显示 DATETIME 值日期部分的 SELECT 语句。其诀窍是使用 EXTEND 函数来截取部分字段，并通过 MDY() 函数（此函数将它们转换为 DATE）传递。下列代码显示一个不完整的示例：

```
SELECT ... MDY (
   EXTEND (DATE_RECEIVED, MONTH TO MONTH),
   EXTEND (DATE_RECEIVED, DAY TO DAY),
   EXTEND (DATE_RECEIVED, YEAR TO YEAR) )
   FROM RECEIPTS ...
```

选择 DATETIME 格式 (GLS)

当应用程序显示 DATETIME 值时，将引用用户指定的 DATETIME 格式。缺省语言环境指定美国英语 DATETIME 格式，具体格式如下：2001-10-25 18:02:13

对于非缺省语言环境，可以使用 GL_DATETIME 环境变量来指定 DATETIME 格式。有关如何使用语言环境的更多信息，请参阅《*SinoDB® GLS* 用户指南》。

要自定义此 DATETIME 格式，请选择适当的语言环境或者设置 GL_DATETIME 或 DBTIME 环境变量。有关这些环境变量的更多信息，请参阅 《*SinoDB® GLS* 用户指南》。

## BOOLEAN 数据类型

BOOLEAN 数据类型是单字节的数据类型。合法布尔值包括 true ('t')、false ('f') 或 NULL。值不区分大小写。

可以将 BOOLEAN 列与另一个 BOOLEAN 列或布尔值作比较。例如：可使用下列 SELECT 语句：

```
SELECT * FROM sometable WHERE bool_col = 't';
SELECT * FROM sometable WHERE bool_col IS NULL;
```

## 字符数据类型 (GLS)

SinoDB® 数据库服务器支持若干种字符数据类型，包括 CHAR、NCHAR 和 NVARCHAR（特殊用途的字符数据类型）。

字符数据：CHAR(***n***) 和 NCHAR(***n***)

CHAR(*n*) 数据类型包含 *n* 个字节的序列。这些字符可以是英语和非英语字符的混合，并且可以是单字节或多字节（亚洲语言）字符。长度 *n* 的范围是 1 到 32767。

每当数据库服务器检索或存储 CHAR(*n*) 值时，都将精确地传输 *n* 个字节。如果插入值的长度小于 *n*，那么数据库服务器用单字节 ASCII 空格字符扩展该值来构成 *n* 个字节。如果插入的值的长度超过 *n* 个字节，那么数据库服务器将截断额外的字符，而不返回错误消息。因此，对于 CHAR(*n*) 列或变量，当插入或更新的值的长度超过 *n* 个字节时，不强制实施数据的语义完整性。

CHAR 列中的数据是按代码集顺序排序的。例如：在 ASCII 代码集中，字符 *a* 的代码集值为 97，*b* 的值为 98，依此类推。数据库服务器以此顺序来排序 CHAR(*n*) 数据。

NCHAR(*n*) 数据类型也包含 *n* 个字节的序列。这些字符可以是英语和非英语字符的混合，并且可以是单字节或多字节（亚洲语言）字符。长度 *n* 具有与 CHAR(*n*) 数据类型相同的限制。每当检索或存储 NCHAR(*n*) 值时，都将精确地传输 *n* 个字节。 如果数据包含多字节字符，那么传输的字符数可以小于字节数。如果插入值的长度小于 *n*，那么数据库服务器用空格字符扩展该值来构成 *n* 个字节。

数据库服务器根据语言环境指定的顺序对 NCHAR(*n*) 列中的数据进行排序。有关如何使用语言环境的更多信息，请参阅《*SinoDB® GLS* 用户指南》。

> 提示： CHAR(*n*) 与 NCHAR(*n*) 数据之间的唯一区别是数据的排序和比较方式不同。可以在 CHAR(*n*) 列中存储非英语字符。但是，由于数据库服务器对 CHAR(*n*) 列的任何排序或比较都使用代码集顺序，所以您可能无法获得具有您期望顺序的结果。

CHAR(*n*) 或 NCHAR(*n*) 值可包含制表符和空格，但通常不包含其他不可打印的字符。当用 INSERT 或 UPDATE 插入行时，或者在使用实用程序载入行时，输入不可打印字符是没有意义的。然而，当使用嵌入式 SQL 的程序创建行时，程序可插入除空字符（二进制零）以外的任何字符。由于标准程序和实用程序不期望遇到不可打印字符，所以在字符列中存储不可打印字符不是一个好的想法。

CHAR(*n*) 或 NCHAR(*n*) 数据类型的优点是在所有数据库服务器上都可用。CHAR(*n*) 或 NCHAR(*n*) 的唯一缺点是具有固定的长度。当行与行的数据值长度变化很大时，将浪费空间。

可变长度字符串：CHARACTER VARYING(***m,r***)、VARCHAR(***m,r***)、NVARCHAR(***m,r***) 和 LVARCHAR(***m***)

通常，字符列中的数据值具有不同的长度。即，许多值具有平均长度，只有少数值具有最大长度。对于以下数据类型，*m* 表示最大字节数，*r* 表示列存储的最小字节数。这些可变长度数据类型旨在当您存储此类数据时节省磁盘空间：

CHARACTER VARYING (***m,r***)

CHARACTER VARYING (*m,r*) 数据类型包含最多 *m* 个字节、最少 *r* 个字节的序列。此数据类型是可变长度字符数据的符合 ANSI 标准的格式。CHARACTER VARYING (*m,r*) 支持代码集顺序以进行字符数据比较。

VARCHAR (*m,r*)

VARCHAR (*m,r*) 是特定于 SinoDB® 的数据类型，用于存储长度可变的字符数据。在功能上与 CHARACTER VARYING(*m,r*) 相同。

NVARCHAR (*m,r*)

NVARCHAR (*m,r*) 也是特定于 SinoDB® 的数据类型，用于存储长度可变的字符数据。按语言环境指定的顺序比较字符数据。

LVARCHAR(*m*)

LVARCHAR 是特定于 SinoDB® 的数据类型，用于存储 1 到 32,739 个字节长度可变的字符数据。如果在列长度声明中未指定最大大小，那么缺省值为 2,048 个字节。LVARCHAR 支持将代码集顺序用于排序，并且也由数据库服务器用于对字符串的内部操作，其最大大小依赖于操作系统。

提示: NVARCHAR(*m,r*) 数据与 CHARACTER VARYING(*m,r*) 或 VARCHAR(*m,r*) 数据的数据比较方式有所不同。有关语言环境如何确定代码集和排序顺序的更多信息，请参阅字符数据#*CHAR(n)* 和 *NCHAR(n)* 在第37页。

在创建为 NLSCASE INSENSITIVE 的数据库中，数据库服务器只处理 NCHAR 和 NVARCHAR 数据类型，而不用考虑字母大小写，因此（例如）NCHAR 字符串“pH”和“Ph”在顺序、排序和比较操作中将视为重复值。

将列定义为可变长度数据类型时，可以指定 *m* 表示最大字节数。如果插入的值所包含的字节数小于 *m*，那么数据库服务器不会（像对 CHAR(*n*) 和 NCHAR(*n*) 值所做的那样）使用单字节空格来扩展值。相反，数据库服务器在磁盘上只用 1 个字节长度的字段来存储实际内容。对于已建立索引的列，*m* 的限制是 254 字节，对于未建立索引的列，限制是 255 字节。

第二个参数 *r* 是可选的保留长度，对字节数设置的限制比存储在磁盘上的值所需要的低。即使值需要的字节数小于 *r*，也仍然会分配 *r* 个字节来存放该值。其目的是在更新行时节省时间。（请参阅可变长度执行时间 在第38页。）LVARCHAR 数据类型不支持任何保留长度。

与 CHAR(*n*) 数据类型相比，CHARACTER VARYING(*m,r*)、LVARCHAR(*m*) 或 VARCHAR(*m,r*) 数据类型的优点如下：

- 当数据项所需的字节数变化范围非常大时，或者当只有少数几项需要超过平均字节数时，能够节省磁盘空间。
- 对压缩程度更高的表执行查询时的速度更快。

与 NCHAR(*n*) 数据类型相比，NVARCHAR(*m,r*) 数据类型也同样有这些优点。

使用可变长度数据类型的潜在缺点如下：

- 除 LVARCHAR 之外，不支持超出 255 个字节的长度。
- 在某些情况下，表的更新速度可能会比较慢。

可变长度执行时间

当使用任何 CHARACTER VARYING(*m,r*)、VARCHAR(*m,r*) 或 NVARCHAR(*m,r*) 数据类型时，表中各行具有可变的字节数而不是固定的字节数。当表中各行具有可变的字节数时，数据库操作的速度会受到影响。

由于可以将更多的行放到一个磁盘页中，所以，与各行具有固定字节数相比，数据库服务器可以进行较少的磁盘操作来搜索表。因此，可以更快速地执行查询。基于同一原因，插入和删除的操作速度也会更快一点。

在更新行时，数据库服务器必须执行的工作量取决于新行中的字节数与旧行中的字节数的比较。如果新行使用相同的字节数或更少，那么与定长行相比，执行时间没有显著的不同。然而，如果新行与旧行相比需要更多的字节数，那么数据库服务器可能需要执行数倍的磁盘操作。因此，更新使用 CHARACTER VARYING(*m,r*)、VARCHAR(*m,r*) 或 NVARCHAR(*m,r*) 数据的表有时会比更新固定长度的字段慢。

要减轻这种影响，请将 *r* 指定为适合大部分数据项的字节数。那么大多数行使用保留字节数，填充操作将只浪费少量空间。只有当替换使用保留字节数的值为使用超出保留字节数的值时，更新速度才会较慢。

大字符对象：TEXT

TEXT 数据类型存储文本块。此数据类型用于存储独立的文档：商业表单、程序源、数据文件或备忘录。虽然可以在 TEXT 项中存储任何数据，但 SinoDB® 工具希望 TEXT 项可以打印，因此请将此数据类型限制为可打印的 ASCII 文本。

TEXT 值不和其所在的行存储在一起。它们分配在完整的磁盘页中（通常是在与行分开的区域中）。有关更多信息，请参阅《*SinoDB®* 管理员指南》。

与 CHAR(*n*) 和 VARCHAR(*m,r*) 相比，TEXT 数据类型的优点是 TEXT 数据项没有大小限制（但存放它的磁盘存储容量除外）。TEXT 数据类型的缺点如下：

- 在完整的磁盘页中分配，因此 TEXT 长度较短时将浪费空间。
- 对您可以如何在 SQL 语句中使用 TEXT 列有限制。（有关该限制的更多信息，请参阅使用 *TEXT* 和 *BYTE* 数据类型 在第39页。）

二进制对象：BYTE

BYTE 数据类型旨在存放程序可以生成的任何数据：图形图像、程序对象文件以及由任何字处理器保存的文档或电子表格。在 BYTE 列中，数据库服务器允许任何类型的、任何长度的数据。

就像 TEXT 一样，BYTE 数据项通常存储在与普通行数据不同的磁盘区域的完整磁盘页中。

与 TEXT 或 CHAR(*n*) 相反，BYTE 数据类型的优点是其接受任何数据。它具有与 TEXT 数据类型相同的缺点。

使用 TEXT 和 BYTE 数据类型

数据库服务器存储和检索 TEXT 和 BYTE 列。要提取和存储 TEXT 或 BYTE 值，您通常使用以支持嵌入式 SQL 的语言（如 SinoDB® ESQL/C）来编写的程序。在这样的程序中，您可以按照类似读写顺序文件的方式来提取、插入或更新 TEXT 或 BYTE 值。

在任何 SQL 语句中（无论是交互式的还是编程的），都不能以下列方式使用 TEXT 或 BYTE 列：

- 在算术或布尔表达式中
- 在 GROUP BY 或 ORDER BY 子句中
- 在 UNIQUE 测试中
- 用于建立索引（无论是独自作为索引还是作为组合索引的一部分）

在以交互方式输入的或者在表单或报告中的 SELECT 语句中，您可以对 TEXT 或 BYTE 值执行下列操作：

- 选择列名（可选择通过下标来抽取它的某部分）。
- 使用 LENGTH(*column_name*) 来返回列的长度。
- 使用 IS [NOT] NULL 谓词来测试该列。

在交互式 INSERT 语句中，您可以使用 VALUES 子句来插入 TEXT 或 BYTE 值，但唯一可以对该列指定的值是空值。但是，您可以使用 SELECT 格式的 INSERT 语句从另一个表复制 TEXT 或 BYTE 值。

在交互式 UPDATE 语句中，您可以将 TEXT 或 BYTE 列更新为空值或更新为返回 TEXT 或 BYTE 列的子查询。

更改数据类型

在构建表之后，您可以使用 ALTER TABLE 语句更改指定给列的数据类型。尽管这样的变更有时是必需的，但您应该避免进行这样的更改，原因如下：

- 要更改数据类型，数据库服务器必须复制并重新构建表。对于大型的表，复制和重新构建可能需要大量的时间和磁盘空间。
- 一些数据类型更改会导致丢失信息。例如：当将列由较长的字符类型更改为较短的字符类型时，长值将被截断；当更改为精度较低的数字类型时，将截断低位数位。
- 可能也必须更改现有的程序、表单、报告和存储查询。

### 空值

在大多数情况下，表中的列可以包含空值。空值表示该列的值可能是未知的或不适用的。例如，在构建关系数据模型 在第14页的电话号码簿示例中，name 表的 anniv 列可以包含空值；如果您不知道该人员的周年纪念日，可以不指定。请不要将空值与零值或空白值混淆。例如：以下语句将一行插入到 stores_demo 数据库的 manufact 表中，并指定 lead_time 列的值为空：

```
INSERT INTO manufact VALUES ('DRM', 'Drumm', NULL)
```

集合列不能包含空元素。在 *SinoDB* 中创建和使用扩展数据类型 在第82页描述了集合数据类型。

## 缺省值

缺省值是当 INSERT 语句中没有指定显式的值时插入到列中的值。缺省值可以是您定义的文字字符串或以下其中一个 SQL 常量表达式：

- USER
- CURRENT
- TODAY
- DBSERVERNAME

并非所有列都需要缺省值，但是，当您使用数据模型时，可能会发现在某些情况下，使用缺省值能够缩短数据输入时间或防止数据输入错误。例如：电话号码簿模型有一个 state 列。查看此列的数据时，您会发现超过 50% 的地址将 California 列为州。为了节省时间，请指定字符串 CA 作为 state 列的缺省值。

## 检查约束

检查约束在执行 INSERT 或 UPDATE 语句期间，可以将数据指定给列之前，指定数据值要满足的条件或要求。在插入或更新期间，如果某行对表上定义的任何检查约束求值为 *false*，那么数据库服务器将返回错误。但是，当检查约束值为 NULL 时，数据库服务器不会报告错误或拒绝记录。因此，创建表时，您可能想同时使用检查约束和 NOT NULL 约束。

要定义约束，请使用 CREATE TABLE 或 ALTER TABLE 语句。例如：以下要求将整数域的值约束在特定范围之内：

```
Customer_Number >= 50000 AND Customer_Number <= 99999
```

要表达对基于字符的域的约束，请使用 MATCHES 谓词和它支持的正则表达式语法。例如：以下约束将电话域限制为美国本地电话号码格式：

```
vce_num MATCHES '[2-9][2-9][0-9]-[0-9][0-9][0-9][0-9]'
```

有关检查约束的其他信息，请参阅《*SinoDB® SQL* 指南: 语法》中的 CREATE TABLE 和 ALTER TABLE 语句。

## 引用约束

您可以在每个表中标识主键和外键来对列设置引用约束。构建关系数据模型 在第14页包含有关如何标识这些键的信息。

当您尝试为主键和外键挑选列时，几乎所有数据类型组合都必须匹配。例如：如果将主键定义为 CHAR 数据类型，那么您也必须将外键定义为 CHAR 数据类型。

但是，当您对一个表中的主键指定 SERIAL 数据类型时，需要对该关系的外键指定 INTEGER。同样，当您对一个表中的主键指定 SERIAL8 数据类型时，需要对该关系的外键指定 INT8；当您对一个表中的主键指定 BIGSERIAL 数据类型时，需要对该关系的外键指定 BIGINT 数据类型。

可以在关系中混合使用的数据类型组合只有以下这些：

- SERIAL 和 INTEGER
- SERIAL8 和 INT8
- BIGSERIAL 和 BIGINT

有关如何创建具有引用约束的表的信息，请参阅《*SinoDB® SQL* 指南: 语法》中的 CREATE TABLE 和 ALTER TABLE 语句。

# 实现关系数据模型

本章显示如何使用 SQL 语法来实现构建关系数据模型 在第14页描述的数据模型。换言之，本章阐述如何创建数据库和表以及如何使用数据来填充表。本章也包含有关数据库日志记录选项、表同义词和命令脚本的信息。

## 创建数据库

现在您已准备好将数据模型创建为数据库中的表。您可以使用 CREATE DATABASE、CREATE TABLE 和 CREATE INDEX 语句来执行此操作。《*SinoDB® SQL* 指南: 语法》描述了这些语句的语法。本节包含有关如何使用 CREATE DATABASE 和 CREATE TABLE 语句来实现数据模型的信息。

请记住，电话号码簿数据模型只是用于举例说明。对于本示例，已将其转换为 SQL 语句。

您可能需要多次创建同一个数据库模型。您可以存储创建模型的语句并在之后再次执行这些语句。有关更多信息，请参阅使用命令脚本 在第46页。

当表已存在时，您必须使用数据行对它们进行填充。您可以通过实用程序或自定义编程手动完成此操作。

### 使用 CREATE DATABASE

数据库是存放数据模型的所有部件的容器。这些部件不仅包括表，还包括与数据库相关联的视图、索引、同义词和其他对象。必须先创建数据库，然后才可以创建任何其他内容。

当数据库服务器创建数据库时，会存储从其系统目录中的 DB_LOCALE 环境变量派生的数据库语言环境。此语言环境决定数据库服务器如何解释存储在数据库中的字符数据。在缺省情况下，数据库语言环境是使用 ISO8859-1 代码集的美国英语语言环境。有关如何使用备用语言环境的信息，请参阅《*SinoDB® GLS* 用户指南》。

当数据库服务器创建数据库时，将记录数据库的存在情况及其日志记录模式。因为数据库服务器直接管理磁盘空间，所以这些记录对操作系统命令不可视。

#### 避免名称冲突

通常，只有一个数据库服务器副本在计算机上运行，数据库服务器管理属于该计算机所有用户的数据库。数据库服务器只保留一份数据库名的列表。数据库的名称必须与数据库服务器管理的任何其他数据库的名称不同。（执行数据库服务器的多个副本是有可能的。例如：您可以创建数据库服务器的多个副本来创建一个安全的测试环境，除了操作数据外。在这种情况下，请确保创建数据库时使用正确的数据库服务器，并在之后访问数据库时也使用正确的数据库服务器。）

#### 选择数据库空间

数据库服务器允许您在特定的数据库空间中创建数据库。数据库空间是磁盘存储器的命名区域。请向数据库服务器管理员询问是否应使用特定的数据库空间。您可以将数据库放在独立的数据库空间中来将它与其他数据库隔离开，也可将它放在特定的磁盘设备上。有关数据库空间及其与磁盘设备的关系的信息，请参阅《*SinoDB®* 管理员指南》。

一些数据库空间是镜像的（在两个磁盘设备上重复以获得高可靠性）。 如果数据库的内容非常重要，那么可以将其放在镜像的数据库空间中。

选择日志记录模式

要指定日志记录数据库或非日志记录数据库，请使用 CREATE DATABASE 语句。数据库服务器为事务日志记录提供下列选项：

- 完全不进行日志记录。

  不建议使用此选项。如果由于硬件故障而丢失数据库，那么您将丢失自上次备份后的所有数据变更。

  ```
  CREATE DATABASE db_with_no_log
  ```

  如果没有选择日志记录，数据库中将不允许出现 BEGIN WORK 语句以及与事务处理相关的其他 SQL 语句。这种情况将影响使用数据库的程序的逻辑。
- 常规（无缓冲）日志记录。

  此选项最适合用于大多数数据库。发生故障时，您将只丢失未提交的事务。

  ```
  CREATE DATABASE a_logged_db WITH LOG
  ```
- 缓冲日志记录。

  即使数据库故障，也只丢失少量最新变更或可能不丢失任何变更。这种小风险的回报是变更操作期间的性能略有改进。

  ```
  CREATE DATABASE buf_log_db WITH BUFFERED LOG
  ```

  缓冲日志记录最适合用于频繁更新的数据库（因而更新速度至关重要），但发生故障时可以根据其他数据重新创建更新。使用 SET LOG 语句在缓冲日志记录与常规日志记录之间进行切换。
- 符合 ANSI 标准的日志记录。

  此日志记录与常规日志记录相同，但还会强制实施用于事务处理的 ANSI 规则。有关更多信息，请参阅使用符合 *ANSI* 标准的数据库 在第11页。

  ```
  CREATE DATABASE std_rules_db WITH LOG MODE ANSI
  ```

  ANSI SQL 的设计禁止使用缓冲日志记录。在创建符合 ANSI 标准的数据库时，您不能关闭事务日志记录。

对于不符合 ANSI 标准的数据库，数据库服务器管理员 (DBA) 可以打开和关闭事务日志记录，或由缓冲日志记录更改为无缓冲日志记录。例如：可以在插入大量新行之前关闭日志记录。

您可以使用 ondblog 和 ontape 实用程序来更改日志记录状态或缓冲模式。有关这些工具的信息，请参阅《*SinoDB*® 管理员指南》。您也可以使用 SET LOG 语句在缓冲日志记录与无缓冲日志记录之间进行切换。有关 SET LOG 的信息，请参阅《*SinoDB*® *SQL* 指南*:* 语法》。

## 使用 CREATE TABLE

使用 CREATE TABLE 语句创建您在数据模型中设计的每个表。此语句具有复杂的格式，但它基本上是表列的列表。对于每个列，提供下列信息：

- 列的名称
- 数据类型（来自您创建的域列表）

该语句也可包含下列其中一个或多个约束：

- 主键约束
- 外键约束
- NOT NULL 约束（或 NULL 约束，允许 NULL 值）
- 唯一约束
- 缺省约束
- 检查约束

简而言之，CREATE TABLE 语句是您在图 *21:* 个人电话号码簿的数据模型 在第28页的数据模型图中所绘制的表的映像。以下示例显示电话号码簿数据模型的语句：

```
CREATE TABLE name
   (
   rec_num  SERIAL PRIMARY KEY,
   lname    CHAR(20),
   fname    CHAR(20),
   bdate    DATE,
   anniv    DATE,
   email    VARCHAR(25)
   );

CREATE TABLE child
   (
   child    CHAR(20),
   rec_num  INT,
   FOREIGN KEY (rec_num) REFERENCES NAME (rec_num)
   );

CREATE TABLE address
   (
   id_num    SERIAL PRIMARY KEY,
   rec_num   INT,
   street    VARCHAR (50,20),
   city      VARCHAR (40,10),
   state     CHAR(5) DEFAULT 'CA',
   zipcode   CHAR(10),
   FOREIGN KEY (rec_num) REFERENCES name (rec_num)
   );

CREATE TABLE voice
   (
   vce_num    CHAR(13) PRIMARY KEY,
   vce_type   CHAR(10),
   rec_num    INT,
   FOREIGN KEY (rec_num) REFERENCES name (rec_num)
   );

CREATE TABLE fax
   (
   fax_num    CHAR(13),
   oper_from  DATETIME HOUR TO MINUTE,
   oper_till  DATETIME HOUR TO MINUTE,
   PRIMARY KEY (fax_num)
   );

CREATE TABLE faxname
   (
   fax_num    CHAR(13),
   rec_num    INT,
   PRIMARY KEY (fax_num, rec_num),
   FOREIGN KEY (fax_num) REFERENCES fax (fax_num),
   FOREIGN KEY (rec_num) REFERENCES name (rec_num)
   );

CREATE TABLE modem
   (
   mdm_num    CHAR(13) PRIMARY KEY,
   rec_num    INT,
   b_type     CHAR(5),
   FOREIGN KEY (rec_num) REFERENCES name (rec_num)
```

```
   );
```

在上述每个示例中，由于 CREATE TABLE 语句没有指定存储选项，所以表数据存储在您为数据库指定的同一个数据库空间中。您可以对表指定与数据库的存储位置不同的数据库空间，也可以将表分段存储到多个数据库空间中。有关 SinoDB® 数据库服务器支持的不同存储选项的信息，请参阅《*SinoDB® SQL* 指南*:* 语法》中的 CREATE TABLE 语句。下一章节阐述一种将表分段存储到多个数据库空间中的方法。

创建分段表

您可以在创建表时使用 FRAGMENT BY 子句来控制数据在表中的分布。以下语句根据循环分布方案创建存储数据的分段表。在此示例中，数据行几乎均匀分布在分段 dbspace1、dbspace2 和 dbspace3 。

```
CREATE TABLE name
   (
   rec_num  SERIAL PRIMARY KEY,
   lname    CHAR(20),
   fname    CHAR(20),
   bdate    DATE,
   anniv    DATE,
   email    VARCHAR(25)
   ) FRAGMENT BY ROUND ROBIN IN dbspace1, dbspace2, dbspace3;
```

有关可以用来创建分段表的不同分布方案的更多信息，请参阅表分段存储策略 在第49页。

删除或修改表

使用 DROP TABLE 语句可移除表及其相关联的索引和数据。例如：要通过添加检查约束来更改表的定义，请使用 ALTER TABLE 语句。使用 TRUNCATE 语句可移除表中的所有行和所有相应的索引数据，同时保留表的定义。有关这些语句的信息，请参阅《*SinoDB® SQL* 指南*:* 语法》。

## 使用 CREATE INDEX

使用 CREATE INDEX 语句可对表中的一个或多个列创建索引，并可选择按索引的顺序对物理表进行集群。本节介绍创建索引时可用的一些选项。有关 CREATE INDEX 语句的更多信息，请参阅《*SinoDB® SQL* 指南*:* 语法》。

假设您创建表 customer：

```
CREATE TABLE customer
(
   cust_num   SERIAL(101) UNIQUE
   fname      CHAR(15),
   lname      CHAR(15),
   company    CHAR(20),
   address1   CHAR(20),
   address2   CHAR(20),
   city       CHAR(15),
   state      CHAR(2),
   zipcode    CHAR(5),
   phone      CHAR(18)
);
```

以下语句显示如何对 customer 表的 lname 列创建索引：

```
CREATE INDEX lname_index ON customer (lname);
```

组合索引

可以创建包含多列的索引。例如：可以创建以下索引：

```
CREATE INDEX c_temp2 ON customer (cust_num, zipcode);
```

索引的双向遍历

ASC 和 DESC 关键字指定数据库服务器保存索引的顺序。在对列创建索引时，如果省略这些关键字或指定 ASC 关键字，那么数据库服务器按升序存储键值。如果指定 DESC 关键字，那么数据库服务器按降序存储键值。

升序顺序表示键值按从最小键到最大键的顺序存储。例如：如果对 customer 表的 lname 列创建升序索引，那么姓氏按以下顺序存储在索引中：Albertson, Beatty, Currie。

降序顺序表示键值按从最大键到最小键的顺序存储。例如：如果对 customer 表的 lname 列创建降序索引，那么姓氏按以下顺序存储在索引中：Currie, Beatty, Albertson。

数据库服务器的双向遍历功能允许您只对列创建一个索引并对指定按排序列的升序或降序顺序对结果进行排序的查询使用该索引。

## 使用表名的同义词

同义词是可以用来替换另一个 SQL 标识符的名称。可使用 CREATE SYNONYM 语句来声明表、视图或（对于 SinoDB®）序列对象的备用名称。

通常，使用同义词来引用不在当前数据库中的表。例如：可以执行下列语句来创建 customer 和 orders 表名的同义词：

```
CREATE SYNONYM mcust FOR masterdb@central:customer;
CREATE SYNONYM bords FOR sales@boston:orders;
```

创建同义词之后，可以在许多原始表名有效的上下文中使用该同义词，如以下示例所示：

```
SELECT bords.order_num, mcust.fname, mcust.lname
   FROM mcust, bords
   WHERE mcust.customer_num = bords.Customer_num
   INTO TEMP mycopy;
```

CREATE SYNONYM 语句在当前数据库中的系统目录表 syssyntable 中存储同义词名称。该同义词可用于在该数据库中进行的任何查询。（然而如果设置了 USETABLENAME 环境变量，那么 SQL 的某些 DDL 语句不支持用同义词代替表名。）

简短的同义词可以简化查询的编写工作，但同义词也可以起其他作用。它们允许将表移至另一个数据库中，甚至移至另一台计算机，同时保持查询不变。

假设有几个引用表 customer 和 orders 的查询。这些查询嵌入在程序、表单和报告中。这些表是演示数据库的一部分，演示数据库存放在数据库服务器 avignon 上。

现在，您决定将这些程序、表单和报告提供给网络中的另一台计算机（数据库服务器 nantes）的用户使用。这些用户有包含表 orders（该表包含他们所在位置的订单）的数据库，但他们必须能够访问 avignon 上的表 customer。

对于这些用户，customer 表是外部的。这是否意味着您必须准备特殊版本（即必须用 customer 表符合数据库服务器名的版本）的程序和报告，更好的解决方案是在用户的数据库中创建同义词，如以下示例所示：

```
DATABASE stores_demo@nantes;
CREATE SYNONYM customer FOR stores_demo@avignon:customer;
```

当在您的数据库中执行存储查询时，名称 customer 指的是实际的表。当在其他数据库中执行这些查询时，该名称会通过同义词解析为对存在于数据库服务器 avignon 上的表的引用。（在不符合 ANSI 标准的数据库中，同义词必须在数据库中的同义词、表、视图和顺序对象的名称中具有唯一性。在符合 ANSI 标准的数据库中，*owner.synonym* 组合必须在使用 tabid 值在数据库中注册的对象名称空间中具有唯一性。）

### 使用同义词链

为了继续前面的示例，假设将一台新的计算机添加到网络中。它的名称是 db_crunch。将 customer 表和其他表移至该计算机上以降低 avignon 上的负载。尽管在新数据库服务器上可以轻松地重新生成该表，但如何将所有访问重定向到它呢？一种方法是创建同义词来替换旧表，如以下示例所示：

```
DATABASE stores_demo@avignon EXCLUSIVE;
RENAME TABLE customer TO old_cust;
CREATE SYNONYM customer FOR stores_demo@db_crunch:customer;
CLOSE DATABASE;
```

当在 stores_demo@avignon 中执行查询时，对表 customer 的引用将找到该同义词并重定向到新计算机上的版本。在上述示例中，对于从数据库服务器 nantes 执行的查询，也会发生此类重定向。数据库 stores_demo@nantes 中的同义词仍将对 customer 的引用重定向到数据库 stores_demo@avignon；在该数据库中，新的同义词将查询发送到数据库 stores_demo@db_crunch。

当您想要在一个操作中将所有访问重定向到表时（就像在本示例中一样），同义词链将非常有用。但是，您应该尽快更新所有用户的数据库，以使其同义词直接指向该表。如果不这样做，数据库服务器处理额外的同义词时就会产生额外的开销，并且如果链中的任何计算机已关闭，则无法找到该表。

可以对本地数据库执行某个应用程序并且之后对另一台计算机上的数据库执行同一个应用程序。该程序在这两种情况下可以运行得一样好（尽管在网络数据库上运行速度可能会慢得多）。如果数据模型相同，那么程序将无法区分两个数据库之间的差别。

### 使用命令脚本

您可以交互式地输入 SQL 语句以创建数据库和表。在某些情况下，您可能需要两次或多次创建数据库和表。例如：在测试版本令人满意后，可能需要再次创建数据库以生成生产版本，或者可能需要在数台计算机上实现同一数据模型。为了节省时间和降低出错机率，您可以将所有创建数据库的语句放在一个文件中并在之后再次执行这些语句。

#### 捕获模式

dbschema 实用程序是一个这样的程序：检查数据库的内容并生成重新创建该数据库所需的所有 SQL 语句。您可以构建数据库的第一个版本，进行更改，直到完全令您满意为止。然后，您可以使用 dbschema 生成复制该数据库所必需的 SQL 语句。有关 dbschema 实用程序的信息，请参阅《*SinoDB® 迁移指南*》。

#### 执行文件

可以从命令文件运行用于以交互方式输入 SQL 语句的程序（如 DB-Access）。您可以启动 DB-Access 来读取并执行您或 dbschema 准备的命令文件。有关更多信息，请参阅《*SinoDB® DB-Access 用户指南*》。

#### 示例

大部分 SinoDB® 数据库服务器产品附带了演示数据库（本书中的大多数示例使用的数据库）。演示数据库是作为一个操作系统命令脚本提供的，此命令脚本会调用 SinoDB® 产品来构建数据库。您可以复制此命令脚本并将其用作自动化数据模型的基础。

## 填充数据库

对于初始测试，填充数据库的最简单方法是在 DB-Access 中输入 INSERT 语句。例如：要将一行插入到演示数据库的 manufact 表中，请在 DB-Access 中输入以下命令：

```
INSERT INTO manufact VALUES ('MKL', 'Martin', 15);
```

如果您准备应用程序，如用 C 编写的应用程序，那么可以使用该应用程序将行输入到数据库表中。

下表列出可以将信息输入到数据库中的 SinoDB® 工具。参考列中的首字母缩写词在表下面有解释。

| 工具 | 用途 | 参考 |
| --- | --- | --- |
| dbaccessdemo | 准备并填充样本数据库。 | DB-A、SQLR |
| DB-Access | 通过输入显式命令来编辑数据库。 | DB-A、SQLS |
| onunload 和 onload | 从磁带或磁盘上的文件中复制整个数据库或所选数据库表，或者将整个数据库或所选数据库表复制到磁带或磁盘上的文件中。 | MG、AR |
| dbload | 将数据从一个或多个文本文件载入到一个或多个现有的表中。 | MG |
| LOAD 和 UNLOAD | 从文本文件中载入数据或将数据载入文本文件。 | SQLS |
| dbexport 和 dbimport | 使用文本文件来复制整个数据库。 | MG |
| Enterprise Replication | 每次更新指定的表时，会更新所选数据库。 | ER |
| C 应用程序 | 使用嵌入在 C 程序中的 SQL 命令来更新数据库。 | ESQLC、DAPI |
| Java™ 应用程序 | 使用嵌入在 Java™ 程序中的 SQL 命令来更新数据库。 | Java™ |
| 网关应用程序 | 访问非 SinoDB® 数据库中的数据。 | GM、GU |

SQLR

《*SinoDB® SQL* 指南*:* 参考》

SQLS

《*SinoDB® SQL* 指南*:* 语法》

MG

《*SinoDB®* 迁移指南》

AR

《*SinoDB®* 管理员参考》

ESQL/C

《*SinoDB® ESQL/C* 程序员指南》

Java™

《*SinoDB® J/Foundation* 开发者指南》

DB-A

《*SinoDB® DB-Access* 用户指南》

ER

《*SinoDB® Enterprise Replication* 指南》

DAPI

《*SinoDB® DataBlade® API* 程序员指南》

## 从其他 SinoDB® 数据库移动数据

通常可以根据存储在表（这些表位于其他 SinoDB® 数据库或操作系统文件内）中的数据衍生某个表的初始行。下列实用程序允许移动大量数据：

- onunload 和 onload 实用程序
- dbexport 和 dbimport 实用程序
- dbload 实用程序
- SQL LOAD 语句

您也可以从另一数据库服务器上的其他数据库中选择您所需的数据来作为数据库中 INSERT 语句的一部分。如以下示例所示，可以从演示数据库的 items 表中选择信息来插入新表中：

```
INSERT INTO newtable
   SELECT item_num, order_num, quantity, stock_num,
      manu_code, total_price
   FROM stores_demo@otherserver:items;
```

## 将源数据载入表中

当数据源不是 SinoDB® 数据库时，您必须找到一种方法来将其转换为 ASCII 文件，即可打印数据的文件，其中每一行代表一个表行的内容。

在将数据存放到 ASCII 文件中后，您可以使用 dbload 实用程序将该数据载入表中。有关 dbload 的更多信息，请参阅《*SinoDB*® 迁移指南》。DB-Access 中的 LOAD 语句也可以从 ASCII 文件载入行。有关 LOAD 和 UNLOAD 语句的信息，请参阅《*SinoDB*® *SQL* 指南*:* 语法》。

在将数据存放到文件中后，您可以使用外部表将该数据载入表中。有关外部表的更多信息，请参阅《*SinoDB*® 管理员指南》。

## 执行大批量载入操作

如果关闭事务日志记录，那么插入数百或数千行会更快。由于发生故障时可以很容易地重新创建丢失的工作，所以日志记录这些插入操作是没有意义的。以下列表包含大批量载入操作的步骤：

- 如果其他用户有可能正在使用数据库，那么请使用 DATABASE EXCLUSIVE 语句来排除。
- 请求管理员关闭数据库的日志记录。

  可以使用现有的日志将数据库恢复到目前状态，并且可以再次执行批量插入操作来恢复那些行（如果丢失了那些行的话）。
- 执行将数据载入表的语句或实用程序。
- 备份新载入的数据库。

  请求管理员执行完全备份或增量备份，或请求管理员使用 onunload 实用程序只创建数据库的二进制副本。
- 还原事务日志记录并释放对数据库的互斥锁定。

# 第 II 部分

# 管理数据库

# 表分段存储策略

本章描述数据库服务器支持的分段存储策略并提供不同分段存储策略的示例。包含有关分段存储、表分段存储的分布方案、创建与修改分段表以及提供分段表权限的信息。

有关如何制定分段存储策略来减少数据争用和提高查询性能的信息，请参阅《*SinoDB*® 性能指南》。

## 什么是分段存储？

分段存储是一项数据库服务器功能，允许您在表级别控制数据的存储位置。分段存储使您能够根据某种算法或方案在表中定义行或索引键组。您可以将每组或分段（也称为分区）存储在与特定物理磁盘相关联的独立数据库空间中。使用 SQL 语句来创建分段并将其指定给数据库空间。

将行或索引键分组为分段的方案称为分布方案。分布方案以及在其中放置分段的数据库空间集共同构成分段存储策略。《*SinoDB*® 性能指南》中说明了制定分段存储策略时必须做出的决定。

在决定是否将表行和/或索引键分段以及决定应该如何在各分段间分布行或键之后，您决定实现此分布的方案。有关 SinoDB® 数据库服务器支持的分布方案的描述，请参阅表分段存储的分布方案 在第50页。

创建分段表和索引时，数据库服务器将每个表和索引分段的位置以及其他相关信息存储在名为 *sysfragments* 的系统目录表中。您可以使用这个表来访问关于分段表和索引的信息。如果将用户自定义例程用作分段存储表达式的一部分，那么该信息会记录在 sysfragexprudrdep 中。有关这些系统目录表包含的信息的描述，请参阅《*SinoDB*® *SQL* 指南*:* 参考》。

从最终用户或客户端应用程序的角度来看，分段表与非分段表完全相同。不要求客户端应用程序作任何修改就可以允许其访问分段表中的数据。

对于某些分布方案，数据库服务器具有有关哪些分段包含哪些数据的信息，因此它可以将客户端数据请求传递到正确分段，而无需访问不相关的分段。（对于循环分布方案和某些基于表达式的分布方案，数据库服务器无法将客户端数据请求传递到正确分段。）有关更多信息，请参阅表分段存储的分布方案 在第50页。

## 为何使用分段存储？

如果您的目标是要改进下列各项中的至少一项，请考虑将表分段：

- 单用户响应时间
- 并行性
- 可用性
- 备份与还原特性
- 数据载入

对于您最终实现的分段存储策略，上述每个目标都有其特有的含义。主分段存储目标确定（至少是影响）实现分段存储策略的方式。当您决定是否使用分段存储来实现上述任何目标时，请记住，分段存储要求进行一些附加的管理和监视活动。

有关上述目标以及如何规划分段存储策略的更多信息，请参阅《*SinoDB*® 性能指南》。

### 分段存储是谁的职责？

在分段存储方面，数据库服务器管理员的职责与数据库管理员 (DBA) 的职责之间存在一些重叠。DBA 创建数据库模式，这可以包括表分段存储。然而，数据库服务器管理员负责分配分段表所在的磁盘空间。由于这些职责都不能以相互隔离的方式执行，所以实现分段存储要求 DBA 与数据库服务器管理员合作。本手册只描述 DBA 为了实现分段存储策略而执行的那些任务。有关数据库服务器管理员为了实现分段存储策略而执行的任务的信息，请参阅《*SinoDB*® 管理员指南》和《*SinoDB*® 性能指南》。

### 分段存储和日志记录

分段表既可以属于日志记录数据库，也可以属于非日志记录数据库。与非分段表相同，如果某个分段表是非日志记录数据库的一部分，那么发生故障时有可能会导致数据不一致。

## 表分段存储的分布方案

分布方案是数据库服务器用于将行或索引条目分布到分段的方法。SinoDB® 数据库服务器支持下列分布方案：

基于表达式的

此分布方案将包含指定值的行放在同一个分段中。指定分段存储表达式，它定义为每个分段指定一组行的条件（作为范围规则或者某个仲裁规则）。可以指定余项分段来存放与任何其他分段的条件都不匹配的所有行（尽管余项分段会降低基于表达式的分布方案的效率）。

循环法

此分布方案在分段中逐个放入行，循环经过一系列分段以便均匀分布各行。数据库服务器以内部方式定义规则。

对于 INSERT 语句，数据库服务器对随机数使用散列函数，以确定放入行的分段。对于 INSERT 游标，数据库服务器将第一行放在随机分段中，将第二行放在下一个顺序分段中，依此类推。如果某个分段满了，那么将跳过该分段。

有关用于指定分布方案的 SQL 语法的完整描述，请参阅《*SinoDB*® *SQL* 指南*:* 语法》中的 CREATE TABLE 和 CREATE INDEX 语句。有关分段存储的性能方面的说明，请参阅《*SinoDB*® 性能指南》。

### 基于表达式的分布方案

要指定基于表达式的分布方案，请使用 CREATE TABLE 或 CREATE INDEX 语句的 FRAGMENT BY EXPRESSION 子句。以下示例包含 FRAGMENT BY EXPRESSION 子句，以便使用基于表达式的分布方案创建分段表：

```
CREATE TABLE accounts (id_num INT, name char(15))
FRAGMENT BY EXPRESSION
id_num <= 100 IN dbspace_1,
id_num <100 AND id_num <= 200 IN dbspace_2,
id_num > 200 IN dbspace_3
```

当使用 CREATE TABLE 语句的 FRAGMENT BY EXPRESSION 子句来创建分段表时，您必须为正在创建的表的每个分段提供一个条件。

您可以定义范围规则或仲裁规则，这些规则向数据库服务器指示要如何将行分布到分段。下列各节描述不同类型的基于表达式的分布方案。

范围规则

范围规则使用 SQL 关系和逻辑运算符来定义表中每个分段的边界。范围规则可包含运算符的以下有限集合：

- 关系运算符 >、<、>= 和 <=
- 逻辑运算符 AND 和 OR
- 包含内置函数的代数表达式

如以下示例所示，范围规则可以基于简单的代数表达式。在该示例中，表达式是对某一列的简单引用。

```
FRAGMENT BY EXPRESSION
id_num > 0  AND id_num <= 20 IN dbsp1,
id_num > 20 AND id_num <= 40 IN dbsp2,
id_num > 40 IN dbsp3
```

按照范围规则的表达式可以是多个代数表达式的连接或分离。以下示例显示了用来定义两组范围的两个代数表达式。第一组范围基于代数表达式：“YEAR(Died) - YEAR(Born)”；第二组范围则基于“MONTH(Born)”。

```
FRAGMENT  BY  EXPRESSION
YEAR(Died) - YEAR(Born) < 21 AND MONTH(Born) >= 1  AND MONTH(Born) < 4 IN dbsp1,
YEAR(Died) - YEAR(Born) < 40 AND MONTH(Born) >= 4  AND MONTH(Born) <  7 IN dbsp2,
```

仲裁规则

仲裁规则使用 SQL 关系和逻辑运算符。与范围规则不同，仲裁规则允许您使用任何关系运算符和任何逻辑运算符来定义规则。另外，您可以在规则中引用任意数目的表列。如以下示例所示，仲裁规则通常包含使用 OR 逻辑运算符来对数据进行分组：

```
FRAGMENT BY EXPRESSION
zip_num = 95228 OR zip_num = 95443 IN dbsp2,
zip_num = 91120 OR zip_num = 92310 IN dbsp4,
REMAINDER IN dbsp5
```

使用 MOD 函数

您可以在 FRAGMENT BY EXPRESSION 子句中使用 MOD 函数将表中的每一行映射至一组整数（散列值）。数据库服务器使用这些值来确定给定的行将存储在哪个分段中。以下示例显示在基于表达式的分布方案中如何使用 MOD 函数：

```
FRAGMENT BY EXPRESSION
MOD(id_num, 3) = 0 IN dbsp1,
MOD(id_num, 3) = 1 IN dbsp2,
MOD(id_num, 3) = 2 IN dbsp3
```

插入和更新行

当您插入或更新行时，数据库服务器按指定的顺序对分段表达式求值，以了解该行是否属于任何分段。如果是的话，数据库服务器在其中一个分段中插入或更新该行。如果该行不属于任何分段，那么将该行放到余项子句指定的分段中。如果分布方案没有包含余项子句，并且该行与任何现有分段表达式的条件都不匹配，那么数据库服务器将返回错误。

## 循环分布方案

要指定循环分布方案，请使用 CREATE TABLE 语句的 FRAGMENT BY ROUND ROBIN 子句。以下语句举例说明具有循环分布方案的分段表：

```
CREATE TABLE account_2
   ...
```

```
   ...
FRAGMENT BY ROUND ROBIN IN dbspace1, dbspace2, dbspace3
```

当数据库服务器接收到将多个行插入使用循环分布的表的请求时，数据库服务器以这样的方式分布行：每个分段中的行数保持大致相同。由于在各分段之间均匀地分布信息，所以循环分布也称为均匀分布。将行分布至使用循环分布的表的规则是数据库服务器的内部规则。

重要： 只能对表分段存储使用循环分布方案。不能使用此分布方案来对索引进行分段。

## 创建分段表

本节说明如何使用 SQL 语句来创建和管理分段表。您可以在创建表时将该表分段，也可以对现有的非分段表进行分段。下列各节概述了这两种备用方法。有关可用来创建分段表的 SQL 语句的完整语法，请参阅《*SinoDB® SQL* 指南: 语法》。

在创建分段表之前，您必须确定适当的分段存储策略。有关如何制定分段存储策略的信息，请参阅《*SinoDB®* 性能指南》。

### 创建新分段表

要创建分段表，请使用 CREATE TABLE 语句的 FRAGMENT BY 子句。

假设您想要创建与 stores_demo 数据库的 orders 表类似的分段表。您决定使用具有三个分段的循环分布方案，并咨询数据库服务器管理员以设置三个数据库空间（每个分段设置一个数据库空间）：dbspace1、dbspace2 和 dbspace3。以下 SQL 语句用来创建分段表：

```
CREATE TABLE my_orders (
   order_num      SERIAL(1001),
   order_date     DATE,
   customer_num   INT,
   ship_instruct  CHAR(40),
   backlog        CHAR(1),
   po_num       CHAR(10),
   ship_date    DATE,
   ship_weight  DECIMAL(8,2),
   ship_charge  MONEY(6),
   paid_date    DATE,
   PRIMARY KEY (order_num),
   FOREIGN KEY (customer_num) REFERENCES customer(customer_num))
   FRAGMENT BY ROUND ROBIN IN dbspace1, dbspace2, dbspace3
```

假设您想要创建多个使用循环法分段的表并想要分段数随着表增大而自动增加。将 AUTOLOCATE 配置参数或会话环境变量设置为创建的起始分段数。在 CREATE TABLE 语句中没有包含 FRAGMENT BY 子句的表缺省情况下会按循环法分段到服务器选择的数据库空间中。缺省情况下，所有的数据库空间都可用，但您可以控制可用的数据库空间列表。

或者，您也可以决定使用基于表达式的分段存储来创建表。假设 my_orders 表有 30000 行，并且您想对三个存储在 dbspace1、dbspace2 和 dbspace3 中的分段均匀地分布行。以下语句显示了如何使用 order_num 列来定义基于表达式的分段存储策略：

```
CREATE TABLE my_orders (order_num SERIAL, ...)
   FRAGMENT BY EXPRESSION
      order_num < 10000 IN dbspace1,
      order_num >= 10000 and order_num < 20000 IN dbspace2,
      order_num >= 20000 IN dbspace3
```

相关链接

*《SinoDB 管理员参考》: AUTOLOCATE 配置参数*

*《SinoDB SQL 指南: 语法》: FRAGMENT BY 子句*

## 从非分段表创建分段表

在以下情况下，您可能需要将非分段表转换为分段表：

- 您有应用程序实现的表分段存储版本。

  您可能想要将若干个小表转换为一个大的分段表。下一章节将介绍如何处理这种情况。请遵循多个非分段表 在第53页章节中的说明。
- 要对现有的大表进行分段。

  请遵循使用单个非分段表 在第53页章节中的说明。

  切记： 在执行转换之前，您必须设置适当数量的数据库空间来存储新创建的分段表。

### 多个非分段表

您可以将两个或更多非分段表合并到单个分段表中。这些非分段表必须具有完全相同的表结构，并且必须存储在独立的数据库空间中。要将非分段表合并到一起，请使用 ALTER FRAGMENT 语句的 ATTACH 子句。

例如：假设有三个非分段表 account1、account2 和 account3，并且将这三个表分别存储在数据库空间 dbspace1、dbspace2 和 dbspace3 中。这三个表具有完全相同的结构，并且要将这三个表合并到一个表中，该表是通过公共列 acc_num 上的表达式进行分段的。

您要将 acc_num 小于或等于 1120 的行存储在 dbspace1 中。acc_num 大于 1120 但小于或等于 2000 的行存储在 dbspace2 中。最后，acc_num 大于 2000 的行将存储在 dbspace3 中。

要使用此分段存储策略对表进行分段，请执行以下 SQL 语句：

```
ALTER FRAGMENT ON TABLE tab1 ATTACH
   tab1 AS acc_num <= 1120,
   tab2 AS acc_num >  1120 and acc_num <= 2000,
   tab3 AS acc_num > 2000;
```

结果是单个表 tab1。另外两个表 tab2 和 tab3 已被使用且不再存在。

有关如何使用 ALTER FRAGMENT 语句的 ATTACH 和 DETACH 子句来提高性能的信息，请参阅《*SinoDB*® 性能指南》。

### 使用单个非分段表

要从非分段表创建分段表，请使用 ALTER FRAGMENT 语句的 INIT 子句。例如：假设要将 orders 表转换为通过循环法进行分段的表。以下 SQL 语句会执行此转换：

```
ALTER FRAGMENT ON TABLE orders INIT
   FRAGMENT BY ROUND ROBIN IN dbspace1, dbspace2, dbspace3;
```

该非分段表上的任何现有索引都将变为采用与该表相同的分段存储策略进行分段。

## 分段表中的 rowid

*rowid* 指的是一个整数，用来定义行的物理位置。在非分段表中，行的 rowid 是唯一且恒定的值。与之相反，分段表中的行不指定 rowid。

重要： 使用主键而不是 rowid 来作为应用程序中的访问方法。由于主键是用 SQL 的 ANSI 规范来定义的，因此使用主键来访问数据可以使应用程序更易于移植。

为了适应必须引用分段表的 rowid 的应用程序，您可以为分段表显式创建 rowid 列。然而，对于类型表，您无法使用 WITH ROWIDS 子句。

要创建 rowid 列，请使用以下 SQL 语法：

- CREATE TABLE 语句的 WITH ROWIDS 子句
- ALTER TABLE 语句的 ADD ROWIDS 子句
- ALTER FRAGMENT 语句的 INIT 子句

创建 rowid 列时，数据库服务器会执行下列操作：

- 对表中的每一行添加 4 个字节的唯一值
- 创建内部索引，数据库服务器使用该索引来通过 rowid 访问表中的数据
- 在 sysfragments 系统目录表中为该内部索引插入一行

### 分段智能大对象

您可以在 CREATE TABLE 语句的 PUT 子句中指定多个智能大对象空间以实现对某个列中的智能大对象的循环分段存储。如果对 CLOB 或 BLOB 列指定多个智能大对象空间，那么数据库服务器将该列的智能大对象以循环法方式分布至指定的智能大对象空间。给定以下 CREATE TABLE 语句，数据库服务器可以将 cat_photo 列中的大对象以循环法方式分布至 sbcat1、sbcat2 和 sbcat3。

```
CREATE TABLE catalog (
   catalog_num  SERIAL,
   stock_num    SMALLINT,
   manu_code    CHAR(3),
   cat_descr    LVARCHAR,
   cat_photo    BLOB)
PUT cat_photo in (sbcat1, sbcat2, sbcat3;
```

## 修改分段存储策略

您可以对分段表进行两种类型的修改。第一种类型为对非分段表进行的修改。这样的修改包括添加列、删除列以及更改列数据类型等。对于这些修改，请使用通常对非分段表使用的 ALTER TABLE 语句。第二种类型的修改为对分段存储策略所作的更改。本节说明如何使用 SQL 语句来修改分段存储策略。

有时，在实现分段存储之后，可能需要变更分段存储策略。最常见的情况是，在将分段存储与查询内或查询间并行化配合使用时，您将需要修改分段存储策略。在这些情况下，修改分段存储策略是提高数据库服务器系统性能的几种方法之一。

### 重新初始化分段存储策略

您可以使用带有 INIT 子句的 ALTER FRAGMENT 语句来对非分段表定义并初始化新的分段存储策略，或对分段表转换现有分段存储策略。也可以使用 INIT 子句来更改分段表达式的求值顺序。

以下示例说明如何使用 INIT 子句来彻底重新初始化分段存储策略。

假定最初创建了以下分段表：

```
CREATE TABLE account (acc_num INTEGER, ...)
   FRAGMENT BY EXPRESSION
      acc_num <= 1120 in dbspace1,
      acc_num > 1120 and acc_num < 2000 in dbspace2,
      REMAINDER IN dbspace3;
```

假定使用此分布方案运作了数个月后您发现 dbspace2 包含的分段中的行数是另外两个分段所包含行数的两倍。这种不平衡导致包含 dbspace2 的磁盘成为 I/O 瓶颈。

要解决这种情况，您决定修改分布，以使每个分段中的行数大致均匀。应修改分布方案以使其包含四个分段而不是三个分段。新的数据库空间 dbspace2a 用于包含新分段，该新分段存储先前 dbspace2 中包含的前半部分的行。dbspace2 中的分段包含它先前存储的后半部分的行。

要实现新的分布方案，首先要创建数据库空间 dbspace2a，然后执行以下语句：

```
ALTER FRAGMENT ON TABLE account INIT
   FRAGMENT BY EXPRESSION
      acc_num <= 1120 in dbspace1,
      acc_num >  1120 and acc_num <= 1500 in dbspace2a,
      acc_num >  1500 and acc_num < 2000 in dbspace2,
```

```
      REMAINDER IN dbspace3;
```

执行此语句时，数据库服务器将立即废弃旧的分段存储策略，并且根据新的分段存储策略将表包含的行重新分布。

也可以使用 ALTER FRAGMENT 的 INIT 子句来执行下列操作：

- 将单个非分段表转换为分段表
- 将分段表转换为非分段表
- 将通过任何策略进行分段的表转换为采用任何其他分段存储策略

有关更多信息，请参阅《*SinoDB® SQL* 指南*:* 语法》中的 ALTER FRAGMENT 语句。

## 修改分段存储策略

您可以使用 ADD、DROP 和 MODIFY 子句来更改对表或索引的分段存储策略。有关这些选项的语法信息，请参阅《*SinoDB® SQL* 指南*:* 语法》中的 ALTER FRAGMENT 语句。

### ADD 子句

定义分段存储策略时，您可能需要添加一个或多个分段。可使用 ALTER FRAGMENT 语句的 ADD 子句将新分段添加到表。假定要将一个分段添加到使用以下语句创建的表中：

```
CREATE TABLE sales (acc_num INT, ...)
   FRAGMENT BY ROUND ROBIN IN dbspace1, dbspace2, dbspace3;
```

要向表 sales 添加新分段 dbspace4，请执行以下语句：

```
ALTER FRAGMENT ON TABLE sales ADD dbspace4;
```

如果分段存储策略基于表达式，那么 ALTER FRAGMENT 的 ADD 子句包含在现有数据库空间之前或之后添加数据库空间的选项。

### DROP 子句

定义分段存储策略时，您必须删除一个或多个分段。对于 SinoDB®，可使用 ALTER FRAGMENT ON TABLE 语句的 DROP 子句从表中删除分段。假定要从使用以下语句创建的表中删除分段：

```
CREATE TABLE sales (col_a INT), ...)
   FRAGMENT BY ROUND ROBIN IN dbspace1, dbspace2, dbspace3;
```

以下 ALTER FRAGMENT 语句使用 DROP 子句从 sales 表中删除第三个分段 dbspace3：

```
ALTER FRAGMENT ON TABLE sales DROP dbspace3;
```

发出此语句时，将把 dbspace3 中所有的行移至剩余的数据库空间 dbspace1 和 dbspace2。

### MODIFY 子句

使用带有 MODIFY 子句的 ALTER FRAGMENT 语句可修改现有分段存储策略中的一个或多个表达式。

假定最初创建了以下分段表：

```
CREATE TABLE account (acc_num INT, ...)
   FRAGMENT BY EXPRESSION
      acc_num <= 1120 IN dbspace1,
      acc_num > 1120 AND acc_num < 2000 IN dbspace2,
      REMAINDER IN dbspace3;
```

执行以下 ALTER FRAGMENT 语句时，请确保在 dbspace1 包含的分段中没有存储值小于或等于零的帐号：

```
ALTER FRAGMENT ON TABLE account
   MODIFY dbspace1 TO acc_num > 0 AND acc_num <=1120;
```

不能使用 MODIFY 子句来变更分布方案包含的分段数。请使用 ALTER FRAGMENT 的 INIT 或 ADD 子句。

## 授予和撤销分段权限

如果要授予有用的分段权限，那么您必须有一种策略来控制数据分布。一种有效的策略是通过表达式将数据记录分段。然而，循环法数据记录分布策略不是有用的策略，因为此策略会将每个新数据记录添加至下一个分段。循环分布使任何跟踪数据分布的清除方法无效，因此取消了对分段权限的任何实际使用。由于基于表达式的分布与循环分布之间存在这种差异，因此 GRANT FRAGMENT 和 REVOKE FRAGMENT 语句只适用于采用基于表达式的分段存储的表。

创建分段表时，不存在缺省分段权限。使用 GRANT FRAGMENT 语句可授予对一个或多个分段的 Insert、Update 或 Delete 权限。如果要同时授予这三项权限，请使用 GRANT FRAGMENT 语句的 ALL 关键字。但是，不能仅仅通过命名包含分段的表来授予分段权限。您必须命名特定的分段。

当您要撤销 Insert、Update 或 Delete 权限时，请使用 REVOKE FRAGMENT 语句。此语句撤销一个或多个用户对分段表的一个或多个分段的权限。如果要撤销当前对某个表的所有权限，可使用 ALL 关键字。如果不在此命令中指定任何分段，那么所撤销的权限适用于表中当前具有权限的所有分段。

有关更多信息，请参阅《*SinoDB® SQL* 指南: 语法》中的 GRANT FRAGMENT、REVOKE FRAGMENT 和 SET 语句。

# 授予和限制对数据库的访问权

本章介绍如何控制对数据库的访问权。在某些数据库中，每个用户都可以访问所有数据。在其他数据库中，某些用户被拒绝访问一些数据或全部数据。

## 使用 SQL 来限制对数据的访问

可以在下列级别限制对数据的访问：

- 可使用 GRANT 和 REVOKE 语句来授予或拒绝对数据库或特定表的访问权，并可以控制人们对数据库的使用种类。
- 可使用 CREATE PROCEDURE 或 CREATE FUNCTION 语句来编写和编译一个用户自定义例程，此例程控制并监视可以读取、修改或创建数据库表的用户。
- 可使用 CREATE VIEW 语句来准备数据的受限制或经修改的视图。此限制可以是垂直的（不包括特定的列）和/或水平的（不包括特定的行）。
- 可以将 GRANT 和 CREATE VIEW 语句组合使用，以便精确控制用户可以修改表的哪些部分以及使用哪些数据。

## 控制对数据库的访问

授予权限 在第57页包含有关正常数据库权限机制是如何基于 GRANT 和 REVOKE 语句的信息。但是，有时可使用操作系统的工具来作为对数据库访问的附加控制方法。

无论操作系统提供了哪些访问控制，当整个数据库的内容高度敏感时，您可能不想将其留在固定于计算机中的公用磁盘上。当数据必须安全时，您可以绕过正常软件控制。

当您或另一位授权人员不使用数据库时，不必将数据库联机。您可以通过下列其中一种方法使数据库无法被访问，而这些方法具有不同程度的不方便性：

- 将物理介质从计算机上拆离并带走。如果磁盘本身不可卸下，那么磁盘驱动器可能是可卸下的。
- 将数据库目录复制到磁带并占有该磁带。

  重要： 在后两种情况下，创建副本之后，您必须记住使用以 NULL 数据覆盖被擦除文件的程序来擦除原始数据库文件。

与移除整个数据库目录相反，您可以复制，然后擦除代表个别表的文件。不要忽视索引文件包含索引列中的数据副本这个事实。移除并擦除索引和表文件。

# 授予权限

访问数据库的权限称为 *access* 权限。例如：使用数据库的权限称为 *Connect* 权限；将行插入表中的权限称为 *Insert* 权限。使用 GRANT 语句可授予对数据库、表、视图或过程的权限，或可将某个角色授予用户或另一个角色。使用 REVOKE 语句可撤销对数据库或数据库对象的权限，或可从用户或另一个角色中撤销某个角色。

角色是 DBA 分配的一类访问权限，如工资单。使用 CREATE ROLE 语句创建角色后，DBA 可以使用 GRANT 语句将访问权限分配给角色，然后将角色分配给个别用户（或分配给其他角色），从而使有相似工作任务的用户可以拥有其工作任务需要的一组访问权限。通过将权限分配给角色，然后将角色分配给用户，可以简化对权限的管理。有关角色中管理访问权限的角色的其他信息，另请参阅外部例程 在第63页和角色 在第64页。

下列权限组控制用户可以对数据和数据库对象执行的操作：

- 数据库级别权限
- 所有权权限
- 表级别权限
- 列级别权限
- 类型级别权限
- 例程级别权限
- 语言级别权限
- 自动化权限

有关 GRANT 和 REVOKE 语句的语法，请参阅《*SinoDB® SQL* 指南: 语法》。

## 数据库级别权限

三个级别的数据库权限提供了全面的方法来控制哪些人可以访问数据库。只有个别用户（而不是角色）才可以拥有数据库级别的权限。

Connect 权限
最低的权限级别是 Connect 权限，它向用户提供查询和修改表的基本能力。

拥有 Connect 权限的用户可以执行下列功能：

- 执行 SELECT、INSERT、UPDATE 和 DELETE 语句（如果用户拥有必需的表级别权限的话）。
- 执行 SPL 例程（如果用户拥有必需的表级别权限的话）。
- 创建视图（如果允许用户查询视图所基于的表的话）。
- 创建临时表并对临时表创建索引。

在用户可以访问数据库之前，他们必须拥有 Connect 权限。通常，在不包含高度敏感或专用数据的数据库中，您在创建数据库后很快就可以用 GRANT CONNECT TO PUBLIC 授予权限。

如果不将 Connect 权限授予 PUBLIC，那么可以通过数据库服务器访问数据库的用户就只包括那些明确授予 Connect 权限的用户。如果有限的用户应拥有访问权，那么此权限允许您将其提供给他们并对所有其他用户拒绝提供此权限。

相关链接

数据库管理员权限 在第58页

用户和 PUBLIC

权限是通过名称授予单个用户或以名称 PUBLIC 授予所有用户的。授予 PUBLIC 的任何权限都将作为缺省权限。

在执行语句之前，数据库服务器会确定用户是否拥有必需的权限。此信息位于系统目录中。有关更多信息，请参阅系统目录表中的权限 在第59页。

数据库服务器将首先查找明确授予发出请求的用户的权限。如果找到这样的授权，那么使用该信息。然后，数据库服务器会查看是否已将限制性较弱的权限授予 PUBLIC。如果有的话，数据库服务器将使用限制性更弱的权限。如果没有对该用户进行任何授权，那么数据库服务器会查找授予 PUBLIC 的权限。如果找到相关的权限，那么将使用该权限。

因此，要对所有用户设置最低级别的权限，请将权限授予 PUBLIC。在特定的情况下您可以通过将更高的个别权限授予用户来覆盖该权限。

Resource 权限

Resource 权限具有与 Connect 权限相同的权限。另外，拥有 Resource 权限的用户还可以创建新的永久表、索引和 SPL 例程，从而永久分配磁盘空间。

相关链接

数据库管理员权限 在第58页

数据库管理员权限

数据库权限的最高级别是数据库管理员，即 DBA。当您创建数据库时，将自动成为 DBA。

拥有 DBA 权限的用户可以执行下列功能：

- 执行 DROP DATABASE、START DATABASE 和 ROLLFORWARD DATABASE 语句。
- 删除或变更任何对象，而不管该对象的所有者是谁。
- 创建其他用户拥有的表、视图和索引。
- 将数据库权限（包括 DBA 权限）授予另一个用户。

只有用户 informix 可以直接修改系统目录表。如果您是用户 informix，那么星瑞格®强烈建议您不要修改任何系统目录表的内容或模式，因为此类操作可能会影响数据库的完整性。

相关链接

*《SinoDB SQL 指南: 语法》: 数据库级别权限*

*Connect* 权限 在第57页

*Resource* 权限 在第58页

## 所有权权限

数据库以及其中的每个表、视图、索引、过程和同义词都具有所有者。尽管拥有 DBA 权限的用户可以创建其他用户拥有的对象，但是对象的所有者通常是创建该对象的人员。

数据库对象的所有者对该对象拥有所有权限，并可以在不具有附加权限的情况下变更或删除该对象。

## 表级别权限

您可以逐个表应用七项权限以允许非所有者拥有所有者的权限。其中的四项权限（Select、Insert、Delete 和 Update 权限）会控制用户对表中数据的 DML 访问。Index 权限控制索引的创建。Alter 权限授予更改表定义的权限。References 权限授予对表指定引用约束的权限。

在符合 ANSI 标准的数据库中，只有表所有者拥有所有权限。在其他数据库中，除非为了对 PUBLIC 限制所有表权限而将 NODEFDAC 环境变量设置为“yes”，否则数据库服务器会在创建表时自动将除 Alter 和 References 以外的所有表权限授予 PUBLIC。当您允许数据库服务器自动将所有表权限授予 PUBLIC 时，任何拥有 Connect 权限的用户都可以访问新创建的表。如果这不是您所期望的（如果存在拥有 Connect 权限的用户，而他们本不能访问此表），那么创建表之后必须撤销 PUBLIC 对该表的所有权限。

访问权限

四项权限控制用户如何访问表。作为表的所有者，您可以独立授予或撤销下列权限：

- Select 允许用户进行查询，包括查询临时表。
- Insert 允许用户添加新行。
- Update 允许用户修改现有的行。
- Delete 允许用户删除行。

用户必须拥有 Select 权限才能检索表的内容。然而，Select 权限不是其他权限的前提条件。用户不必拥有 Select 权限就可以拥有 Insert 或 Update 权限。

例如：应用程序可能有一个使用表。每次启动特定的程序时，都会插入一行到使用表中，以记载使用了该程序。在程序终止之前，它会更新该行以显示其运行的时长，并可能记录其用户执行的工作计数。

如果要让程序的任何用户能够在此使用表中插入和更新行，那么请向 PUBLIC 授予对该表的 Insert 和 Update 权限。但是，您也可以仅授予少数用户 Select 权限。

系统目录表中的权限
数据库级别和表级别权限记录在系统目录表中。任何拥有 Connect 权限的用户都可以查询系统目录表以确定向哪些用户授予哪些权限。

数据库级别权限和角色记录在 sysusers 系统目录表中，在这个表中，主键是 username 列，而 usertype 列包含单个字符 C（表示 Connect）、R（表示 Resource），或者 D（表示 DBA）来指定 username 拥有的最高数据库级别权限，或者 G（如果 username 是某个角色的授权标识符）。如果 username 拥有缺省角色，则最后一列 defrole 会存储缺省角色。（无法将缺省角色或数据库级别权限授予角色，但角色可以拥有其他访问权限，如表级别权限，且可以将非缺省角色授予角色。）行中显示 PUBLIC 组所拥有的最高数据库级别权限的 username 是 public。

表级别权限记录在 systabauth 系统目录表中，该表使用包含表号、授权者和被授权者的复合主键。在 tabauth 列中，权限的编码如以下列表所示。

s
   无条件的 select
u
   update
-
   未授权
i
   insert
d
   delete
x
   index
a
   alter
r
   references

连字符表示未授予的权限，因此授予所有权限表示为 su-idxar，-u------ 表示只授予 Update 权限。代码字母通常是小写的，但当 GRANT 语句中使用了 WITH GRANT OPTION 关键字时，代码字母是大写的。

当第三个位置中出现星号 (*) 时，表示该表和被授权者有一些列级别的权限。特定的权限记录在 syscolauth 中。其主键是表号、授权者、被授权者和列号的组合。唯一的属性是一个包含三个字母的列表，其显示权限的类型：s、u 或 r。

Index、Alter 和 References 权限

Index 权限允许其拥有者对表创建和变更索引。与 Select、Insert、Update 和 Delete 权限相似，当创建表时，会对 PUBLIC 自动授予 Index 权限。

您可以将 Index 权限授予任何人，但为了行使该权限，用户还必须拥有 Resource 数据库权限。因此，尽管 Index 权限是自动授予的（在符合 ANSI 标准的数据库中除外），但是只拥有 Connect 数据库权限的用户仍然无法行使其 Index 权限。由于索引会占用大量的磁盘空间，所以这样的限制是合理的。

Alter 权限允许其拥有者对表使用 ALTER TABLE 语句，包括添加和删除列以及复位 SERIAL 列的起始点等权力。您应该只对熟悉数据模型的用户以及您信任的会谨慎行使其权力的用户授予 Alter 权限。

References 权限允许您对表施加引用约束。与 Alter 权限相同，您应该只对熟悉数据模型的用户授予 References 权限。

类型表的 Under 权限

您可以授予或撤销 Under 权限来控制用户是否可以将类型表用作继承层次结构中的超表。创建表时，将自动把 Under 权限授予 PUBLIC（在符合 ANSI 标准的数据库中除外）。在符合 ANSI 标准的数据库中，将把对表的 Under 权限授予表的所有者。要限制哪些用户可以将某个表定义为继承层次结构中的超表，首先必须撤销 PUBLIC 的 Under 权限，然后指定要授予 Under 权限的用户。例如：只指定一组有限的用户将 employee 表用作继承层次结构中的超表，可执行下列语句：

```
REVOKE UNDER ON employee
   FROM PUBLIC;

GRANT UNDER ON employee
   TO johns, cmiles, paulz
```

有关在继承层次结构中如何使用 UNDER 子句来创建表的信息，请参阅表继承 在第98页。

对表分段的权限

使用 GRANT FRAGMENT 语句可授予对分段表的单个分段的 Insert、Update 和 Delete 权限。GRANT FRAGMENT 语句仅对采用基于表达式的分布方案进行分段的表有效。

假设您创建一个 customer 表，该表通过表达式分为三个分段，这些分段位于数据库空间 dbsp1、dbsp2 和 dbsp3 中。以下语句显示如何只将对前两个分段（dbsp1 和 dbsp2）的 Insert 权限授予用户 jones、reed 和 mathews。

```
GRANT FRAGMENT INSERT ON customer (dbsp1, dbsp2)
   TO jones, reed, mathews
```

要授予对表所有分段的权限，请使用 GRANT 语句或 GRANT FRAGMENT 语句。

有关 GRANT FRAGMENT 和 REVOKE FRAGMENT 语句的信息，请参阅《*SinoDB® SQL* 指南:语法》。

## 列级别权限

可使用特定列的名称对 Select、Update 和 References 权限进行限定。通过命名特定的列，就可以授予用户对表的特定访问权。可以允许用户只查看特定的列、只更新特定的列或只对特定的列施加引用约束。

可使用 GRANT 和 REVOKE 语句来授予或限制用户对表数据的访问权。此功能解决了只有特定用户才可以了解某个雇员的薪水、工作表现评价或其他敏感属性这一问题。假设雇员数据表是按以下示例所示的方式定义的：

```
CREATE TABLE hr_data
   (
   emp_key INTEGER,
   emp_name CHAR(40),
   hire_date DATE,
   dept_num SMALLINT,
   user-id CHAR(18),
```

```
    salary DECIMAL(8,2)
    performance_level CHAR(1),
    performance_notes TEXT
    )
```

由于这个表包含敏感数据，因此在创建后立即执行以下语句：

```
REVOKE ALL ON hr_data FROM PUBLIC
```

对于人力资源部门中的所选人员以及所有经理，请执行以下语句：

```
GRANT SELECT ON hr_data TO harold_r
```

这样就可以允许特定用户查看所有的列。（本章的最后一节包含有关如何将经理限制为仅供其员工查看的信息。）对于执行业绩考核的一线经理，可执行如下所示的语句：

```
GRANT UPDATE (performance_level, performance_notes)
   ON hr_data TO wallace_s, margot_t
```

此语句允许经理输入对其雇员的评价。您将只对人力资源部的经理或被委托改变薪水等级的人员执行以下语句：

```
GRANT UPDATE (salary) ON hr_data to willard_b
```

对于人力资源部的职员，可执行如下所示的语句：

```
GRANT UPDATE (emp_key, emp_name, hire_date, dept_num)
   ON hr_data TO marvin_t
```

此语句使特定用户能够维护不敏感的列，但拒绝他们更改工作表现等级或薪水的权限。对于 MIS 部门中指定计算机用户标识符的人员，可执行如下所示的语句：

```
GRANT UPDATE (user_id) ON hr_data TO eudora_b
```

对于允许连接至数据库但未授权查看薪水或工作表现评价的所有用户，执行如下语句以允许他们查看不敏感的数据：

```
GRANT SELECT (emp_key, emp_name, hire_date, dept_num, user-id)
   ON hr_data TO george_b, john_s
```

这些用户可以执行以下查询：

```
SELECT COUNT(*) FROM hr_data WHERE dept_num IN (32,33,34)
```

然而，尝试执行如下查询将生成错误消息并且不返回数据：

```
SELECT performance_level FROM hr_data
   WHERE emp_name LIKE '*Smythe'
```

## 类型级别权限

SinoDB® 支持用户定义的数据类型 (UDT)。创建用户定义的数据类型时，只有 DBA 或该数据类型的所有者才能授予或撤销类型级别权限（这些权限控制哪些用户可以使用该 UDT）。SinoDB® 支持两个类型级别的权限。

对用户定义的类型的 Usage 权限

要控制哪些用户可使用不透明类型、单值类型或命名行类型，请指定对该数据类型的 Usage 权限。Usage 权限允许 DBA 或类型所有者限制用户对列或程序变量（或者对命名行类型的表或视图）指定数据类型或者

对该数据类型指定强制转型的能力。创建数据类型时将把 Usage 权限自动授予 PUBLIC（在符合 ANSI 标准的数据库中除外）。在符合 ANSI 标准的数据库中，将把对数据类型的 Usage 权限授予该数据类型的所有者。

要限制哪些用户可以使用不透明类型、单值类型或命名行类型，首先必须撤销 PUBLIC 的 Usage 权限，然后指定要授予 Usage 权限的用户的名称。例如：要限制只有一组用户可以使用名为 circle 的数据类型，可执行下列语句：

```
REVOKE USAGE ON circle
   FROM PUBLIC;

GRANT USAGE ON circle
   TO dawns, stevep, terryk, camber;
```

相关链接

选择命名行类型的名称 在第91页

对命名行类型的 Under 权限

对于命名行类型，可授予或撤销 Under 权限，此权限控制用户是否可以将命名行类型指定为继承层次结构中的另一个命名行类型的超类型。创建命名行类型时，将 Under 权限自动授予 PUBLIC（在符合 ANSI 标准的数据库中除外）。在符合 ANSI 标准的数据库中，将对命名行类型的 Under 权限授予该类型的所有者。

要限制特定用户将命名行类型定义为继承层次结构中的超类型的能力，首先必须撤销 PUBLIC 的 Under 权限，然后指定要授予 Under 权限的用户的名称。例如：只指定一组有限的用户可将命名行类型 person_t 用作继承层次结构中的超类型，可执行下列语句：

```
REVOKE UNDER ON person_t
   FROM PUBLIC;

GRANT UNDER ON person_t
   TO howie, jhana, alison
```

有关在继承层次结构中如何使用 UNDER 子句来创建命名行类型的信息，请参阅类型继承 在第95页。

相关链接

定义类型层次结构 在第96页

## 例程级别权限

可将 Execute 权限应用于用户自定义例程 (UDR) 以授权非所有者执行 UDR。如果在不符合 ANSI 标准的数据库中创建 UDR，那么缺省例程级别权限是 PUBLIC；除非先撤销了 Execute 权限，否则不需要将此权限授予特定用户。如果在符合 ANSI 标准的数据库中创建例程，那么缺省情况下其他用户不具有 Execute 权限；必须将 Execute 权限授予特定用户。以下示例将 Execute 权限授予用户 orion，以便 orion 可使用名为 read_address 的 UDR：

```
GRANT EXECUTE ON ROUTINE read_address TO orion;
```

sysprocauth 系统目录表记录例程级别权限。sysprocauth 系统目录表使用由例程号、授权者和被授权者组成的主键。在 procauth 列中，执行权限由小写的“e”表示。如果执行权限是使用 WITH GRANT 选项授予的，那么该权限由大写的“E”表示。

有关例程级别权限的更多信息，请参阅《*SinoDB® SQL* 指南: 教程》。

## 语言级别权限

SinoDB® 支持用内置存储过程语言 (SPL) 编写的 UDR，也支持用 C 语言和 Java™ 语言编写的 UDR（称为外部例程）。要创建任何 UDR，用户必须拥有数据库的 Resource 权限（或 DBA 权限）。此外，要创建 UDR，用户也必须通过相应的 GRANT 语句获得编程语言的 Usage 权限：

- GRANT USAGE ON LANGUAGE C（用于 C 例程）

- GRANT USAGE ON LANGUAGE JAVA（用于 Java™ 1 例程）
- GRANT USAGE ON LANGUAGE SPL（用于 SPL 例程）

用户不仅需要拥有所需语言级别权限，而且如果 IFX_EXTEND_ROLE 配置参数已启用（缺省情况下，或通过设置为 1 或 ON），那么只有 DBSA 授予内置 EXTEND 角色的用户才能创建、变更或删除外部例程。

SPL 例程

在缺省情况下，SPL 的语言使用权限会授予用户 informix 和拥有 DBA 权限的用户。然而，只有用户 informix 才可以将语言使用权限授予其他用户。具有 DBA 权限的用户拥有语言使用权限，但无法将这些权限授予其他用户。在缺省情况下，用来创建 SPL 例程的 Usage 权限会授予 PUBLIC。

以下语句显示用户 informix 如何撤销 PUBLIC 使用 SPL 创建 UDR 的许可权，并将该许可权授予用户 mays、jones 和 freeman：

```
REVOKE USAGE ON LANGUAGE SPL FROM PUBLIC
GRANT USAGE ON LANGUAGE SPL TO mays, jones, freeman
```

假设取消了 PUBLIC 对 SPL 例程的缺省 Usage 权限，以下语句显示具有 DBA 权限的用户如何将用来注册 SPL 例程的 Usage 权限授予用户 franklin、reeves 和 wilson：

```
GRANT USAGE ON LANGUAGE SPL TO franklin, reeves, wilson
```

外部例程

此版本的 SinoDB® 不支持对外部例程（由 C 语言或 Java™ 语言编写）的语言级别权限。然而，当 IFX_EXTEND_ROLE 配置参数为 ON 时，将会通过内置的 EXTEND 角色提供同样的功能，任何用户注册、删除或替换用 C 语言或 Java™ 语言编写的 UDR 或 DataBlade® 模块时都需要该角色。

只有数据库服务器管理员 (DBSA)（缺省情况下为用户 informix）可以授予 EXTEND 角色。与用户定义的角色名称相反，内置角色（比如 EXTEND 和 DBSECADM）是自动激活的，并且不能修改由该角色授予的权限。当启用 EXTEND 角色时，只有授予了 EXTEND 角色的用户才能创建或删除 DataBlade® 模块或外部 UDR。

DBSA 也可以选择通过将 IFX_EXTEND_ROLE 配置参数设为 OFF 或将该参数保留为未设置来禁用该限制。在这种情况下，拥有数据库的 Resource 权限的任何用户都可以创建以 C 语言或 Java™ 语言编写的 UDR。

## 自动维护权限

这种设计似乎迫使您在最初设置数据库时执行大量的 GRANT 语句。此外，当人们改变工作时，需要不断维护权限。例如：如果人力资源部门的某个雇员被解雇，那么您可能要尽快撤销 Update 权限，否则这个满心不愉快的雇员可能会执行以下语句：

```
UPDATE hr_data
   SET (emp_name, hire_date, dept_num) = (NULL, NULL, 0)
```

虽然不会那么戏剧性，但还是具有同等的必要性，在包含敏感数据的任何模型中，每天甚至每小时都需要更改权限。如果您预期有这种需要，那么可以准备一些自动化工具来帮助维护权限。

第一步应该是指定基于用户的工作（而不是基于表的结构）的权限级。例如，一线经理需要以下权限：

- 对虚构的 hr_data 表的 Select 权限和有限的 Update 权限
- 对这个以及其他数据库的 Connect 权限
- 对那些数据库中数个表的一定程度的权限

当某位经理升职或调到现场办公室时，必须撤销所有这些权限并授予一组新的权限。

定义所支持的权限级，并对每一级指定必须授予访问权的数据库、表和列。然后，为每一级设计两个自动化例程，一个用于将该权限级授予用户，另一个用于撤销该权限级。

### 通过命令脚本实现自动化

操作系统可能支持自动执行命令脚本。在大多数操作环境中，交互式 SQL 工具（如 DB-Access）接受从命令行执行命令和 SQL 语句。可以将这两项功能组合起来以自动进行权限维护。

详细信息取决于您的操作系统以及使用的交互式 SQL 工具的版本。必须创建执行下列功能的命令脚本：

- 将权限要进行更改的用户标识符作为其参数
- 准备自定义为包含该用户标识符的 GRANT 或 REVOKE 语句的文件
- 调用交互式 SQL 工具（如 DB-Access），并指定参数以通知此工具选择该数据库并执行已准备好的 GRANT 或 REVOKE 语句文件

这样就可以将用户权限级的更改减少为一两个命令。

### 角色

另一种避免逐个实例地更改用户权限的方法是使用角色。数据库环境中的角色概念与操作系统中的组概念类似。角色是一项数据库功能，它允许 DBA 通过将许多用户视为某个类的成员，从而标准化并更改这些用户的访问权限。（无法为用户定义的角色授予数据库级别的权限 Connect、Resource 或 DBA，但角色可拥有数据库对象的自主访问权限，包括对表对象、表分段、用户定义的数据类型、用户自定义例程和编程语言的权限。）

例如，如果您向 PUBLIC 组授予每个数据库的 Connect 权限来处理公司新闻和消息，那么可以创建名为 news_mes 的角色，并为该角色授予对表的 Insert 和 Delete 权限，授予该角色的员工可在这些表中添加或删除行。有新到员工时，只能将该人员添加至 news_mes 角色。通过发出 SET ROLE news_mes 语句来启用该角色，则新员工将获取 news_mes 角色的访问权限。（或者，也可以在需要这些权限的每个数据库中定义 ***user***.sysdbopen 过程（其中 ***user*** 是新员工的权限标识），以便在用户连接数据库时自动执行 SET ROLE news_mes 语句。）

此过程反过来也能起作用。要更改授予了 news_mes 角色的每个人员的自主访问权限，请使用 GRANT 或 REVOKE 语句在定义 news_mes 角色的每个数据库中更改该角色的权限。

> 注: 然而，在从个别用户或属于 PUBLIC 组的用户所拥有的用户定义角色中撤销授予这些用户的访问权限、从这些用户撤销该角色或删除该角色时，授予这些用户的访问权限将不受影响。

### 创建角色

要开始角色创建过程，请确定角色的名称以及要授予拥有该角色的用户的连接和权限。尽管连接和权限的确在您的域中，但您在声明新角色的名称时必须考虑某些因素。对于用户定义的角色名称，请勿使用以下任何 SQL 关键字、访问权限或内置角色：

- ALTER
- C
- CONNECT
- DBA
- DBSECADM
- DEFAULT
- DELETE
- EXECUTE
- EXTEND
- INDEX
- INSERT
- NONE
- NULL
- PUBLIC
- REFERENCES
- RESOURCE
- SELECT
- SETSESSIONAUTH

- SPL
- UPDATE

由于角色名称是权限标识符而非 SQL 标识符，因此角色名称的最大长度为 32 个字节。

角色名称必须与同一数据库中的现有角色名称不同。角色名也必须与操作系统已知的用户名（包括服务器计算机已知的网络用户）不同。要确保新角色名称是唯一的，请检查共享内存结构中当前正在使用数据库的用户名称以及以下系统目录表中的用户名称：

- sysusers
- systabauth
- syscolauth
- sysfragauth
- sysprocauth
- sysroleauth
- syssecpolicyexemptions
- sysxtdtypeauth

在相反情况下，将用户添加至数据库时，请检查用户名是否与任何现有角色名不同。

在确认角色名之后，请使用 CREATE ROLE 语句来创建新角色。创建角色后，缺省情况下将把角色管理的所有权限授予 DBA。

> 重要： 角色的作用域仅为当前数据库。执行 SET ROLE 语句时，指定的角色仅在当前数据库中生效。为安全起见，仅通过角色拥有访问权限的用户无法通过视图、触发器或过程来访问远程数据库中的表。

处理用户权限以及将角色权限授予其他角色
作为 DBA，您可以使用 GRANT 语句将角色权限授予用户，也可以使用户将权限授予其他用户。使用 GRANT 语句的 WITH GRANT OPTION 子句可执行此操作。

将权限授予角色时，也可以如本例所示使用 WITH GRANT OPTION 子句：

```
GRANT rol1 TO usr1 WITH GRANT OPTION;
```

WITH GRANT OPTION 子句仅对您授予用户的角色（和访问权限）有效。当 TO 子句指定角色时或指定 PUBLIC 组时，如果加入 WITH GRANT OPTION 关键字，那么数据库服务器会发出错误。

在授予角色权限时，可使用角色名来替代 GRANT 语句中的用户名。可以将角色权限授予另一个角色。例如：假设将角色 A 的权限授予角色 B。当用户启用角色 B 时，用户将同时获得角色 A 和角色 B 的权限。

但是，角色授权是不能循环的。如果将角色 A 的权限授予角色 B，并将角色 B 的权限授予角色 C，那么将角色 C 的权限授予 A 会返回错误。

如果必须更改权限，请使用 REVOKE 语句来删除现有权限，然后使用 GRANT 语句来添加新权限。

启用缺省角色和非缺省角色

在 DBA 授予权限并将用户添加至角色之后，有两种方式启用角色。

- DBSA 可以使用 GRANT DEFAULT ROLE 语句为 PUBLIC 或个别用户指定缺省角色。在用户连接到数据库时，该角色会自动激活为初始角色设置。
- 当用户在 SET ROLE 语句中指定该角色时，用户拥有的任何角色也都可以激活。

启用角色时，授予该角色的所有权限将变为可用的，并且所有权限将显式授予您或 PUBLIC。

先将权限分配给角色，然后将该角色授权为指定用户的缺省角色，这样做的话，对于那些用户运行应用程序（需要一组特定访问权限）的会话来说是非常方便的。如果对应用程序进行重新编译（使其包含将必要的访问权限具体分配给用户的 GRANT 和 SET ROLE 语句）是不现实的，那么请使用缺省角色。

确认角色中的成员资格和删除角色

您会发现在某些情况下不确定某个角色中包括哪个用户，可能是您没有创建该角色，也可能是创建该角色的人员不可用。对 sysroleauth 和 sysusers 系统目录表进行查询以了解谁对哪个表拥有权限以及存在多少个角色。

在确定哪些用户拥有哪些角色之后，您可能会发现一些角色不再需要。要删除角色，请使用 DROP ROLE 语句。在删除角色之前，必须遵循下列要求：

- 只能删除 sysusers 系统目录表中列为角色的角色，不能删除内置角色（例如 NONE 或 EXTEND）。
- 您必须拥有 DBA 权限，否则您必须获得角色中的可授予选项才能删除角色。

### 在运行时确定当前角色

如果在授予适当访问权限的角色中遇到意外错误，请确保在运行时启用了该角色。要在连接到数据库时获取该信息，可以使用 onstat -g sql 或 onstat -g ses 命令。仅查看您当前的角色时，请使用 SQL 的 CURRENT_ROLE 操作符。要查看您的缺省角色，请使用 SQL 的 DEFAULT_ROLE 操作符。

## 使用 SPL 例程来控制对数据的访问

可以使用 SPL 例程来控制对数据库中各个表和列的访问权。使用例程来进行各种程度的访问控制。SPL 的一项强大功能是能够将 SPL 例程指定为具有 DBA 权限的例程。在编写具有 DBA 权限的例程时，您可以允许具有很少或没有表权限的用户在执行该例程时具有 DBA 权限。在该例程中，用户可以使用其临时 DBA 权限来执行特定任务。具有 DBA 权限的例程可以完成下列任务：

- 限制各个用户从表中读取的信息量。
- 限制对数据库所作的所有更改并确保不会意外地清空或更改整个表。
- 监视对表所作的整类更改，如删除或插入。
- 将所有对象创建（数据定义）操作限制为只在 SPL 例程中发生，以便对表、索引和视图的构建方式进行全面控制。

有关使用 SPL 编写的例程的信息，请参阅《*SinoDB® SQL* 指南: 教程》。

### 限制数据读取

以下示例中的例程对用户隐藏 SQL 语法，但要求用户对 customer 表具有 Select 权限。如果要限制用户可以选择的内容，那么通过编写例程以能够在下列环境中工作：

- 您是数据库的 DBA。
- 用户对数据库具有 Connect 权限。他们对表不具有 Select 权限。
- 使用 DBA 关键字来创建 SPL 例程（或一组 SPL 例程）。
- SPL 例程（或一组 SPL 例程）读取用户的表。

如果只让用户读取客户的名称、地址和电话号码，那么可按如下示例所示修改此过程：

```
CREATE DBA PROCEDURE read_customer(cnum INT)
RETURNING CHAR(15), CHAR(15), CHAR(18);

DEFINE p_lname,p_fname CHAR(15);
DEFINE p_phone CHAR(18);

SELECT fname, lname, phone
   INTO p_fname, p_lname, p_phone
   FROM customer
   WHERE customer_num = cnum;

RETURN p_fname, p_lname, p_phone;

END PROCEDURE;
```

## 限制数据更改

使用 SPL 例程时，可以限制用户对表所作的更改。通过 SPL 例程来引导所有更改。数据由 SPL 例程进行更改，而不是由用户直接进行更改。如果要限制用户每次只能删除一行以确保他们不会意外删除表中所有的行，那么使用下列权限来设置数据库：

- 您是数据库的 DBA。
- 所有用户都拥有数据库的 Connect 权限。他们可能拥有 Resource 权限。对于本示例，他们对受保护的表不具有 Delete 权限。
- 使用 DBA 关键字来创建 SPL 例程。
- SPL 例程执行删除操作。

编写与以下过程（使用带有用户提供的 customer_num 的 WHERE 子句）类似的 SPL 过程从 customer 表中删除行：

```
CREATE DBA PROCEDURE delete_customer(cnum INT)

DELETE FROM customer
   WHERE customer_num = cnum;

END PROCEDURE;
```

## 监视数据更改

使用 SPL 例程时，可以创建对数据库所作更改的记录。可以记录由特定用户所作的更改，也可以在每次进行更改时作记录。

可以监视单个用户对数据库所作的所有更改。通过用于跟踪每个用户所作更改的 SPL 例程来引导所有更改。如果要在用户 acctclrk 每次修改数据库时作记录，请使用下列权限来设置数据库：

- 您是数据库的 DBA。
- 所有其他用户都拥有数据库的 Connect 权限。他们可能拥有 Resource 权限。对于本示例，他们对受保护的表不具有 Delete 权限。
- 使用 DBA 关键字来创建 SPL 例程。
- SPL 例程执行删除操作并记录特定用户所作的更改。

编写类似于以下示例（针对 UNIX™ 平台）的 SPL 例程，该示例使用用户提供的客户号来更新表。如果用户刚好是 acctclrk，那么将在文件 updates 中放置删除记录。

```
CREATE DBA PROCEDURE delete_customer(cnum INT)

DEFINE username CHAR(8);

DELETE FROM customer
   WHERE customer_num = cnum;

IF username = 'acctclrk' THEN
   SYSTEM 'echo Delete from customer by acctclrk >>
/mis/records/updates' ;
END IF
END PROCEDURE;
```

要监视通过此过程进行的所有删除操作，请删除 IF 语句并使 SYSTEM 语句更具通用性。以下过程更改上面的例程以记录所有删除操作：

```
CREATE DBA PROCEDURE delete_customer(cnum INT)

DEFINE username CHAR(8);
LET username = USER ;
DELETE FROM tbname WHERE customer_num = cnum;
```

```
SYSTEM
 'echo Deletion made from customer table, by '||username
||'>>/hr/records/deletes';

END PROCEDURE;
```

### 限制对象创建

要对构建哪些对象以及如何构建加以约束，请在符合下列设置的条件下使用 SPL 例程：

- 您是数据库的 DBA。
- 所有其他用户都拥有数据库的 Connect 权限。他们不具有 Resource 权限。
- 使用 DBA 关键字来创建 SPL 例程（或一组 SPL 例程）。
- SPL 例程（或一组 SPL 例程）按照您定义的方式来创建表、索引和视图。可使用这样的例程来设置培训数据库环境。

SPL 例程可以包括创建一个或多个表和相关联的索引，如以下示例所示：

```
CREATE DBA PROCEDURE all_objects()

CREATE TABLE learn1 (intone SERIAL, inttwo INT NOT NULL,
   charcol CHAR(10) );
CREATE INDEX learn_ix ON learn1 (inttwo);
CREATE TABLE toys (name CHAR(15) NOT NULL UNIQUE,
   description CHAR(30), on_hand INT);
END PROCEDURE;
```

要使用 all_objects 过程来控制将列添加至表的操作，请撤销所有用户对数据库的 Resource 权限。当用户尝试在此过程外通过 SQL 语句来创建表、索引或视图时，他们将无法完成该操作。当用户执行该过程时，他们具有临时的 DBA 权限，因此，举例来说，CREATE TABLE 语句会成功，并且可以保证对添加的每个列都加以约束。另外，用户创建的对象由这些用户拥有。对于 all_objects 过程，任何执行该过程的用户都拥有两个表和索引。

## 视图

视图是合成的表。您可以将其视为表来进行查询，在某些情况下，还可以更新它。但是，它并不是表。它是存在于真实的表以及其他视图中的数据的合成。

视图的基础是 SELECT 语句。创建视图时，请定义一个 SELECT 语句，用于在您访问视图时生成视图的内容。用户也可以使用 SELECT 语句来查询视图。在某些情况下，数据库服务器将用户的 SELECT 语句和对视图定义的 SELECT 语句合并，然后实际执行组合语句。有关视图性能的信息，请参阅《*SinoDB®* 性能指南》。

由于您编写确定视图内容的 SELECT 语句，因此可以使用视图来实现以下任何目的：

- 限制用户只能访问表的特定列

  您在视图的选择列表中只声明允许的列。
- 限制用户只能访问表的特定行

  指定只返回允许行的 WHERE 子句。
- 将插入和更新的值约束为具有特定范围

  可使用 WITH CHECK OPTION（在使用 *WITH CHECK OPTION* 关键字 在第72页中进行了说明）来强制实施约束。
- 提供对派生数据的访问权，而不在数据库中存储冗余数据

  编写表达式来派生数据到视图的选择列表中。每次查询视图时都重新派生数据。派生数据总是最新的，还不会在数据模型中引入冗余。
- 隐藏复杂 SELECT 语句的详细信息

将多表连接的复杂细节隐藏在视图中，这样用户和应用程序员都不必重复这些细节。

## 创建视图

以下示例创建一个基于 stores_demo 数据库中的表的视图：

```
CREATE VIEW name_only AS
SELECT customer_num, fname, lname FROM customer
```

此视图只显示该表的三个列。由于视图不包含 WHERE 子句，因此不会限制可出现的行。

以下示例基于两个表的连接：

```
CREATE VIEW full_addr AS
SELECT address1, address2, city, state.sname,
   zipcode, customer_num
   FROM customer, state
   WHERE customer.state = state.code
```

州名表降低了数据库的冗余度；它只允许将完整的州名存储一次，这对于 Minnesota 之类的较长州名可能非常有用。这个 full_addr 视图允许用户检索地址，就像每一行都存储了完整的州名一样。下列两个查询是等效的：

```
SELECT * FROM full_addr WHERE customer_num = 105

SELECT address1, address2, city, state.sname,
   zipcode, customer_num
   FROM customer, state
   WHERE customer.state = state.code AND customer_num = 105
```

然而，定义基于连接的视图时务必小心谨慎。这样的视图是不可修改的；即，不能对其使用 UPDATE、DELETE 或 INSERT 语句。有关如何使用视图进行修改的说明，请参阅使用视图进行修改 在第71页。

以下示例限制视图中可以查看的行：

```
CREATE VIEW no_cal_cust AS
   SELECT * FROM customer WHERE NOT state = 'CA'
```

此视图显示 customer 表的所有列，但只显示特定的行。以下示例是一个视图，用于限制用户只能查看与其相关的行：

```
CREATE VIEW my_calls AS
   SELECT * FROM cust_calls WHERE user_id = USER
```

cust_calls 表的所有列都可用，但仅限于包含用户（该用户可执行查询）的标识符的那些行中。

### 类型视图

当要区分显示同一数据类型的数据的两个视图时，可创建带类型视图。例如：假设要对下面这个表创建两个视图：

```
CREATE TABLE emp
(  name    VARCHAR(30),
   age     INTEGER,
   salary  INTEGER);
```

下列语句对 emp 表创建两个带类型视图 name_age 和 name_salary：

```
CREATE ROW TYPE name_age_t
```

```
(  name    VARCHAR(20),
   age     INTEGER);

CREATE VIEW name_age OF TYPE name_age_t AS
   SELECT name, age FROM emp;

CREATE ROW TYPE name_salary_t
(  name    VARCHAR(20),
   salary  INTEGER);

CREATE VIEW name_salary OF TYPE name_salary_t AS
   SELECT name, salary FROM emp
```

创建带类型视图时，该视图显示的数据具有命名行类型。例如：name_age 和 name_salary 视图包含 VARCHAR 和 INTEGER 数据。由于视图带有类型，因此对 name_age 视图执行的查询返回类型为 name_age 的列视图，而对 name_salary 视图执行的查询返回类型为 name_salary 的列视图。因此，数据库服务器能够对 name_age 和 name_salary 视图返回的行加以区分。

在某些情况下，带类型视图具有无类型视图所不具有的优点。例如：假设按以下方式重载 myfunc() 函数：

```
CREATE FUNCTION myfunc(aa name_age_t) ......;
CREATE FUNCTION myfunc(aa name_salary_t) .....;
```

因为 name_age 和 name_salary 视图是带类型视图，所以下列语句将解析为适当的 myfunc() 函数：

```
SELECT myfunc(name_age) FROM name_age;
SELECT myfunc(name_salary) FROM name_salary;
```

您也可以使用表名的别名来编写上述 SELECT 语句：

```
SELECT myfunc(p) FROM name_age p;
SELECT myfunc(p) FROM name_salary p;
```

如果不将包含同一数据类型的两个视图创建为带类型视图，那么数据库服务器无法区分这两个视图显示的行。有关函数重载的更多信息，请参阅《*SinoDB*® 用户自定义例程和数据类型开发者指南》。

视图中的重复行

即使基础表只包含唯一的行，视图也可能会生成重复的行。如果视图的 SELECT 语句可以返回重复的行，那么视图本身可显示为包含重复的行。

可通过两种方法防止此问题。一种方法是在视图的投影列表中指定 DISTINCT。但是，当指定 DISTINCT 时，无法使用视图进行修改。另一种方法是始终查询限制为唯一的列或一组列。（如果查询主键或候选键的列，那么可以确保只返回唯一的行。构建关系数据模型 在第14页包含有关主键和候选键的信息。）

## 对视图的限制

因为视图实际上不是表，所以不能对其建立索引，并且也不能是 ALTER TABLE 和 RENAME TABLE 之类语句的对象。不能使用 RENAME COLUMN 来重新命名视图的列。要更改任何关于视图定义的内容，必须删除该视图并重新创建它。

由于必须与用户的查询合并，因此视图基于的 SELECT 语句不能包含下列子句或关键字：

INTO TEMP

用户的查询可包含 INTO TEMP；如果视图也包含它，那么无法确定数据将放于何处。

ORDER BY

用户的查询可包含 ORDER BY。如果视图也包含它，那么对列或排序方向的选择可能会有冲突。

视图基于的 SELECT 语句可包含 UNION 关键字。在此类情况下，数据库服务器将视图存储在隐式临时表中，在该处，根据需要对联合进行求值。用户的查询将此临时表用作基本表。

### 当视图基础更改时

可以通过数种方法来更改视图所基于的表和视图。视图将自动反映大部分的更改。

删除表或视图时，会自动删除同一数据库中依赖于该表或视图的任何视图。

唯一一种变更视图定义的方法是删除并重新创建该视图。因此，如果更改其他视图所依赖的视图的定义，那么还必须重新创建那些视图（因为它们将被全部删除）。

重命名表时，会修改同一数据库中依赖该表的任何视图以使用新名称。重命名列时，会更新同一数据库中依赖该表的视图以查询正确的列。然而，视图本身的列名称不会更改。例如，请再调用基于 customer 表的以下视图：

```
CREATE VIEW name_only AS
   SELECT customer_num, fname, lname FROM customer
```

现在，假设按以下方式更改 customer 表：

```
RENAME COLUMN customer.lname TO surname
```

要直接选择客户的姓氏，您现在必须选择新列名。但是，从视图中看来，列的名称没有更改。下列两个查询是等效的：

```
SELECT fname, surname FROM customer

SELECT fname, lname FROM name_only
```

当删除列以变更表时，将不会修改视图。如果使用那些视图，那么会发生错误 -217（在查询的任何表中都找不到列）。不修改视图的原因是可以通过删除列并接着添加同名的列来更改表中列的顺序。基于该表的视图会继续工作，并保留其原始列顺序。

您可以使视图基于外部数据库中的表和视图。对其他数据库中的表和视图所作的更改不会反映在视图中。在某人查询视图并且因外部表更改而出错之前，这样的更改可能不明显。例如，如果视图包含的某个远程对象中的某个列变更为具有其他类型，那么必须删除该视图并重新创建。

## 使用视图进行修改

您可以将视图视为表来进行修改。某些视图可以修改，而另一些则不能修改，这取决于 SELECT 语句。限制是不同的，取决于您是否使用 DELETE、UPDATE 或 INSERT 语句。

如果定义视图的 SELECT 语句没有包含下列任何一项，那么可以修改该视图：

- 两个或更多表的连接
- 聚合函数或 GROUP BY 子句
- DISTINCT 关键字或其同义词，UNIQUE
- UNION 关键字
- 计算值或文字值

当视图避免了所有这些限制项时，视图的每一行刚好都与一个表的一行相对应。通过使用 INSTEAD OF 触发器，如果触发器操作修改了基本表，那么可以规避视图上的这些限制。

### 使用视图进行删除

可以对可修改视图使用 DELETE 语句，就像该视图是表一样。数据库服务器将删除底层表的正确行。

### 更新视图

可以对可修改视图使用 UPDATE 语句。但是，数据库服务器不支持更新任何派生列。派生列是由 CREATE VIEW 语句的选择列表中的表达式（例如 order_date + 30）生成的。

以下示例显示一个可修改的视图，其中包含一个派生列和一个可接受的 UPDATE 语句：

```
CREATE    VIEW   response(user_id, received, resolved, duration) AS
```

```
   SELECT user_id, call_dtime, res_dtime,  res_dtime - call_dtime
      FROM cust_calls
      WHERE user_id = USER;

UPDATE response SET resolved = TODAY
   WHERE resolved IS NULL;
```

因为视图的 duration 列表示一个表达式，所以不能更新该列（数据库服务器无法决定如何在表达式命名的两个列之间分布更新值，即使在理论上也无法这样做）。但如果没有在 SET 子句中命名派生列，那么可以将视图视为表来执行更新。

即使基础表的行是唯一的，视图也可能会返回重复的行。您无法对各重复行加以区分。如果更新一组重复行的其中一行（例如：如果使用游标来更新 WHERE CURRENT），那么无法确定要更新基础表中的哪一行。

插入到视图中

只有当视图为可修改的并且不包含派生列时才可将行插入到视图中。第二项限制的原因是插入的行必须为所有的列提供值，但数据库服务器无法指出如何通过表达式来对插入的值进行分布。如上述示例所示，尝试插入到 response 视图中将失败。

当可修改的视图不包含派生列时，可以将视图视为表来执行插入。然而，对于视图未显示的任何列，数据库服务器使用 NULL 作为那些列的值。如果这样的列不允许 NULL 值，那么会出错，并且插入失败。

在 SinoDB® 视图（包括复杂视图）中插入行（或执行 UPDATE 或 DELETE 操作）的另一种机制是创建 INSTEAD OF 触发器，如《*SinoDB® SQL* 指南*:* 语法》中所述。

使用 WITH CHECK OPTION 关键字

可以将不满足视图条件的行（也就是无法通过视图进行查看的行）插入到视图中。也可以更新视图的行，以使其不再满足视图的条件。

为了避免更新某个视图的某一行，使其不再满足视图的条件，请在创建视图时添加 WITH CHECK OPTION 关键字。该子句要求数据库服务器对插入或更新的每一行进行测试，以确保其符合视图的 WHERE 子句设置的条件。如果不符合条件，那么数据库服务器将拒绝该操作，并显示错误。

限制: 当视图定义中包含 UNION 运算符时，不能包含 WITH CHECK OPTION 关键字。

在上述示例中，名为 response 的视图是按以下示例所示的方式定义的：

```
CREATE   VIEW   response (user_id, received,  resolved,  duration) AS
   SELECT user_id,call_dtime,res_dtime,res_dtime  - call_dtime
      FROM cust_calls
      WHERE user_id = USER
```

可以更新视图的 user_id 列，如以下示例所示：

```
UPDATE response SET user_id = 'lenora'
   WHERE received BETWEEN TODAY AND TODAY - 7
```

视图需要 user_id 等于 USER 的行。如果用户 tony 执行此更新，那么更新后的行将从视图中消失。但是，可以按如下示例所示创建视图：

```
CREATE   VIEW    response  (user_id, received, resolved,duration) AS
   SELECT user_id, call_dtime, res_dtime, res_dtime - call_dtime
      FROM cust_calls
      WHERE user_id = USER
WITH CHECK OPTION
```

用户 tony 所作的上述 UPDATE 操作会被当作错误而遭到拒绝。

可使用 WITH CHECK OPTION 功能来强制实施任何类型的可作为 Boolean 表达式来表示的数据约束。在以下示例中，您可以创建表的视图，对于此视图，可以根据 WHERE 子句的条件来表达对数据的所有逻辑约束。然后可以要求通过该视图对该表进行所有修改。

```
CREATE VIEW order_insert AS
   SELECT * FROM orders O
      WHERE order_date = TODAY -- no back-dated entries
         AND EXISTS -- ensure valid foreign key
            (SELECT * FROM customer C
               WHERE O.customer_num = C.customer_num)
         AND ship_weight < 1000 -- reasonableness checks
         AND ship_charge < 1000
WITH CHECK OPTION
```

由于需要执行 EXISTS 和其他测试（当数据库服务器检索现有行时，这些测试应该能成功），此视图显示 orders 中数据的效率不高。但是，如果对 orders 执行的插入操作只通过此视图进行（并且还没有使用完整性约束来约束数据），那么用户无法插入延期订单、无效的客户号或过高的装运重量和运费。

在视图定义更改时重新执行预编译语句

数据库服务器使用您在对某个视图准备 SELECT 语句时存在的视图定义。如果在对某个视图准备 SELECT 语句后，该视图的定义发生变化，那么由于预编译语句没有反映新的视图定义，因此执行它将会生成不正确的结果。不会生成 SQL 错误。

## 权限和视图

创建视图时，数据库服务器将检测您对基础表和视图拥有的自主访问权限。但是，使用视图时，只检测您对该视图拥有的权限。

修改对表的访问权限（在视图定义中引用的）之后，无论对基础表的访问权限作何更改，数据库服务器都不会自动将其应用到现有视图中。要将对派生视图的任何表的修改权限强制应用到视图中，请使用 SQL 的 DROP VIEW 和 CREATE VIEW 语句来删除和重新创建视图。

删除视图也会毁坏基于该视图定义的任何其他视图和 INSTEAD OF 触发器。在删除并重新创建一个视图来同步其访问权限及其基础表的访问权限之后，可以使用 CREATE TRIGGER 和 CREATE VIEW 语句来分别重新创建任何 INSTEAD OF 触发器和从属视图（在删除视图时数据库服务器毁坏的）。

### 创建视图时的权限

数据库服务器将进行检测以确保您具有执行视图定义中的 SELECT 语句所需的所有权限。如果您不具有那些权限，那么数据库服务器不会创建视图。

此项检测确保用户无法通过对某个表创建视图并查询该视图来进行对该表的未经授权的访问。

创建视图后，数据库服务器至少会授予您（您是该视图的创建者和所有者）对它的 Select 权限。不会对 PUBLIC 进行自动授权，这与新创建表的情况相同。

数据库服务器将检测视图定义以了解视图是否是可修改的。如果是的话，数据库服务器将授予您对该视图的 Insert、Delete 和 Update 权限（前提是您对基础表或视图也具有该权限）。换言之，如果新视图是可修改的，那么数据库服务器将从基础表或视图复制您的 Insert、Delete 和 Update 权限并对新视图授予该权限。如果您对基础表只拥有 Insert 权限，那么将只授予您对视图的 Insert 权限。

此项检测确保用户无法通过使用视图来获取对他们尚未拥有的任何权限的访问权。

由于无法变更视图或对视图建立索引，因此永远不会授予对视图的 Alter 和 Index 权限。

此部分不适用于远程表上的视图。远程表上的许可权不能自动传播到那些表上的视图。例如，要向 PUBLIC 提供对包含远程表上一个或多个列的视图的 Select 存取权，必须对该视图明确执行 REVOKE ALL FROM PUBLIC，然后向 PUBLIC 明确授予对该视图的 Select 权限。

## 使用视图时的权限

当您尝试使用视图时，数据库服务器将只检测授予您对该视图的权限，且并不会检测您是否具有访问基础表的权限。

如果视图由您创建，那么您拥有上一节记载的权限。如果您不是创建者，那么您拥有创建者（或某个拥有 WITH GRANT OPTION 权限的人）授予您的权限。

因此，您可以创建一个表并撤销 PUBLIC 对它的访问权；然后可通过视图对该表授予有限的访问权限。假设要授予用户对以下表的访问权限：

```
CREATE TABLE hr_data
   (
   emp_key INTEGER,
   emp_name CHAR(40),
   hire_date DATE,
   dept_num SMALLINT,
   user-id CHAR(18),
   salary DECIMAL(8,2),
   performance_level CHAR(1),
   performance_notes TEXT
   )
```

列级别权限 在第60页部分显示了如何直接授予对 hr_data 表的访问权限。下列示例采用另一种方法。假设创建该表时执行了以下语句：

```
REVOKE ALL ON hr_data FROM PUBLIC
```

（在符合 ANSI 标准的数据库中，这样的语句不是必需的。）现在，为不同的用户类创建一系列视图。对于应该对非敏感列具有只读访问权的用户，您可以创建以下视图：

```
CREATE VIEW hr_public AS
   SELECT emp_key, emp_name, hire_date, dept_num, user_id
      FROM hr_data
```

对此视图拥有 Select 权限的用户可以查看不敏感的数据，但不能进行更新。对于必须输入新行的人力资源部人员，您可以创建另一个视图，如以下示例所示：

```
CREATE VIEW hr_enter AS
   SELECT emp_key, emp_name, hire_date, dept_num
      FROM hr_data
```

将对此视图的 Select 和 Insert 权限授予这些用户。由于您（您同时是表和视图的创建者）对表和视图具有 Insert 权限，因此您可以将对该视图的 Insert 权限授予其他对该表不具有权限的人。

对于 MIS 部门中输入或更新新用户标识符的人员，您还要创建另一个视图，如以下示例所示：

```
CREATE VIEW hr_MIS AS
   SELECT emp_key, emp_name, user_id
      FROM hr_data
```

此视图与前一视图的不同点在于它不显示部门号和聘用日期。

最后，经理需要访问所有列，并且他们需要能够只更新其员工的业绩考核数据。可通过创建 hr_data 表来满足这些需求，该表包含每个雇员的部门号和计算机用户标识符。让经理作为他们所管理部门的成员成为一条规则。于是，以下视图将限制经理只能访问反映其雇员的行：

```
CREATE VIEW hr_mgr_data AS
   SELECT * FROM hr_data
      WHERE dept_num =
         (SELECT dept_num FROM hr_data
```

```
            WHERE user_id = USER)
      AND NOT user_id = USER
```

最后一个条件是必需的，这样经理才不具有对表中其自己的行的更新访问权。因此，可以安全地将此视图的 Update 权限授予经理，但只授予对所选列的 Update 权限，如以下语句所示：

```
GRANT SELECT, UPDATE (performance_level, performance_notes)
   ON hr_mgr_data TO peter_m
```

# 分布式查询

本章概述了分布式查询。分布式查询允许对 SinoDB® 数据库服务器网络内多个数据库中的数据进行共享访问。不同的数据库服务器可以管理多个数据库，可以在单独的分布式查询中引用这些数据库。

## 分布式查询概述

SinoDB® 数据库服务器允许查询同一个数据库服务器或多个数据库服务器内的多个数据库。这类查询称为分布式查询。数据库服务器可位于单个主机上、同一网络中的不同计算机上或网关上。（通常，本章描述的分布式查询的大多数功能和限制都适用于函数调用，也适用于在多个数据库中引用对象或数据的分布式 INSERT、DELETE 或 UPDATE 操作。）

### 在单个 SinoDB® 实例的多个数据库中的分布式查询

在同一 SinoDB® 实例的多个数据库中进行分布式操作时，对返回的数据类型存在以下限制：

- 查询、DML 操作或函数调用可以返回任何内置数据类型，包括 BLOB、BOOLEAN、CLOB 和 LVARCHAR 内置不透明类型。
- 查询、DML 操作或函数调用不能返回 DISTINCT 或 OPAQUE 数据类型，除非这些类型是对内置数据类型的显式强制转型，并且所有 DISTINCT 和 OPAQUE 数据类型及所有显式强制转型都在存储或接收这些数据类型的每个参与数据库中进行了定义。

### 在两个或两个以上 SinoDB® 实例的多个数据库中的分布式查询

在两个或两个以上 SinoDB® 实例的多个数据库中进行分布式操作时，对返回的数据类型存在以下限制：

- 查询、DML 操作或函数调用可以返回任何透明的内置数据类型、BOOLEAN 数据类型 和 LVARCHAR 数据类型。
- 查询、DML 操作或函数调用可以返回 DISTINCT 数据类型，该数据类型显式强制转型为内置数据类型，其基本类型为透明的内置数据类型 BOOLEAN 或 LVARCHAR 数据类型。此外，基本类型也可以为 DISTINCT 数据类型，其基本数据类型为透明的内置类型 BOOLEAN、LVARCHAR 或基于这些类型之一的其他 DISTINCT 数据类型。

必须在参与分布式操作的每个数据库中定义这些显式强制转型、函数和 DISTINCT 数据类型。如果任何参与的数据库服务器是较早的版本而不能在跨服务器操作中支持这些数据类型，那么这些服务器只能返回它们支持的数据类型。如果分布式操作指定了不受支持的数据类型，那么该操作将会失败。就像在同一 SinoDB® 服务器实例的各数据库之间进行的分布式操作一样，跨服务器分布式操作要求所有数据库均为兼容的事务日志记录模式（如分布式查询的日志记录模式限制 在第80页中所述）。

### 分布式查询中的协调者服务器和参与者服务器

访问多个数据库服务器的查询称为跨服务器查询。为了支持跨多个数据库服务器的分布式 DML 操作，SinoDB® 服务器会保持由一个协调者服务器和一个或多个参与者服务器构成的层次结构关系。

协调者和参与者的定义如下：

- 协调者对查询的解决方案提供指导。它也决定是应该提交查询还是取消查询。

- 参与者（也称为下级服务器）指导一条分支上分布式查询的执行。该分支是分布式查询的一部分，只涉及该参与者数据库服务器。

通常，启动任何跨服务器分布式 DML 操作的本地数据库服务器上的会话称为协调者，而执行同一分布式事务的各个分支的下级服务器称为参与者服务器。分布式查询有时适用于任何跨服务器 DML 操作，包括 DELETE、INSERT、MERGE 和 UPDATE 语句。

跨服务器分布式操作可包含多个数据库服务器实例作为下级参与者，但只能包含一个协调者。该协调者必须与每个下级建立一个连接。但是，在单个分布式操作中，如果下级服务器尝试连接到另一个下级服务器或与协调者建立一个新连接，那么该分布式操作将失败并返回错误。

以下分布式查询示例引用了多服务器环境中的对象：

- db 是本地数据库
- db2 是位于同一服务器上的另一个数据库
- master_db 是远程服务器 new_york 上的外部数据库

以下示例显示了访问远程服务器 new_york 上的数据的跨服务器分布式查询，将数据库 db 的本地服务器用作事务协调者。

```
DATABASE db;
SELECT col2 FROM db2:tab1, master_db@newyork:tab2;
```

在给定的时间内，会话只能有一个本地数据库，但可以打开多个外部数据库。分布式查询必须始终来自作为协调者的数据库服务器实例。

在跨服务器操作中调用远程例程

跨服务器连接拓扑限制为两层（协调者与参与者），限制跨服务器操作中的远程 UDR 的调用为协调者。

例如，在上述示例的多服务器环境中，假设本地数据库包含以下 CREATE PROCEDURE 语句定义的名为 bad_example1() 的 UDR：

```
CREATE PROCEDURE bad_example1()
   EXECUTE PROCEDURE master_db@new_york:bad_example2()
END PROCEDURE;
```

假设 new_york 数据库服务器实例的 master_db 数据库中名为 bad_example2() 的远程 UDR 有以下定义来调用 new_buffalo 数据库服务器实例上名为 bad_example3() 的远程 UDR：

```
CREATE PROCEDURE bad_example2()
   EXECUTE PROCEDURE examples_db@new_buffalo:bad_example3()
END PROCEDURE;
```

假设 new_buffalo 数据库服务器的 examples_db 数据库中存在远程 UDR bad_example3()，并且您拥有所有必需的访问权限来调用本地实例、new_york 实例和 bad_example3() 例程的 new_buffalo 实例。

在此环境中，当您从本地数据库执行以下语句时：

```
EXECUTE PROCEDURE bad_example1();
```

本地数据库服务器（作为此跨服务器分布式操作的协调者）会调用 new_york 数据库服务器实例上的远程 UDR bad_example2()。但是，当远程 bad_example2() 例程尝试建立 new_york 和 new_buffalo 数据库服务器之间的连接时会返回错误 -556。因为该连接违反了在分布式事务中只有协调者可以连接到参与者服务器的规则，所以该尝试连接会失败。

但是，如果会话未调用 bad_example1()，而 new_york 数据库服务器实例的本地数据库上的会话则通过执行以下语句来启动跨服务器分布式操作，那么调用同一远程 UDR 将成功。

```
EXECUTE PROCEDURE bad_example2();
```

在这种情况下，new_york 数据库服务器是跨服务器操作的协调者，其中 new_buffalo 数据库服务器是执行 bad_example3() 的分布式操作分支上的下级参与者，并且没有第三级跨服务器连接。

切记:

跨服务器操作只需要一个协调者来连接作为下级参与者的一个或多个数据库服务器实例。如果参与者尝试访问任何远程对象或调用任何其他服务器实例的数据库中的远程 UDR，那么该调用会失败并返回错误 -556，因为只有协调者可以与其他数据库服务器建立连接。

## 配置数据库服务器以使用分布式查询

要将多个 SinoDB® 服务器用于分布式查询，必须确保所有涉及的数据库服务器都配置为支持通过网络进行服务器到服务器的通信。

编辑以下配置文件以允许进行分布式查询：

- sqlhosts 文件 - 用于保存有关其他服务器的连接信息
- onconfig 文件 - 用于设置 DBSERVERALIASES、NETTYPE、REMOTE_USERS_CFG 和 REMOTE_SERVER_CFG 配置参数
- 由 REMOTE_USERS_CFG 或 REMOTE_SERVER_CFG 配置参数指定的文件 - 用于配置网络安全性
- /etc/services 和 /etc/hosts 文件或通过网络系统（如 NIS+）管理的等价文件 - 用于 TCP/IP 网络配置

  注: 要配置网络安全性，请使用您通过 REMOTE_USERS_CFG 或 REMOTE_SERVER_CFG 配置参数指定的文件，而不是 hosts.equiv 或受信任用户的 rhosts 文件。

要将几台数据库服务器设置为可使用分布式查询，请使用以下其中一种方法来存储所有数据库的 sqlhosts 信息：

- 存储在一个 sqlhosts 文件中，由 INFORMIXSQLHOSTS 环境变量指向该文件
- 存储在多个 sqlhosts 文件中，每个文件位于每个数据库服务器目录下
- 存储在一个集中管理的文件中，该文件位于网络安装的只读文件系统上，通过使用每个数据库服务器目录中的 sqlhosts 文件作为符号链接来链接到该集中管理的文件

  注: 要使用数据库服务器的非 root 安装来进行分布式查询，必须在 onconfig 文件中设置以下其中一个配置参数：

  - REMOTE_USERS_CFG - 指定使用 rhosts 文件列出远程服务器上的受信任用户的备用方法。
  - REMOTE_SERVERS_CFG - 指定使用 etc/hosts.equiv 文件列出受信任远程主机的备用方法。

相关链接

*《SinoDB 管理员参考》: REMOTE_USERS_CFG 配置参数*
*《SinoDB 管理员参考》: REMOTE_SERVER_CFG 配置参数*
*《SinoDB 管理员指南》: sqlhosts 文件和 SQLHOSTS 注册表键*
*《SinoDB SQL 指南: 参考》: INFORMIXSQLHOSTS 环境变量*
*《SinoDB 管理员参考》: DBSERVERALIASES 配置参数*
*《SinoDB 管理员参考》: NETTYPE 配置参数*
*《SinoDB 管理员指南》: TCP/IP 连接文件*

## 分布式查询的语法

本节描述了如何在分布式查询中指定远程服务器、数据库和数据库对象。

### 访问远程服务器和数据库

分布式查询中任何语句的核心元素都是数据库段。使用这些段的语法可以指定远程数据库服务器、数据库或数据库对象。

数据库名称段

数据库名称段用来指定数据库的名称。以下示例显示了指定远程数据库的不同方法：

```
empinfo@personnel '//personnel/empinfo'
```

数据库对象名称段

数据库对象名称段用来指定数据库对象的名称，包括约束、索引、触发器和任何同义词。以下示例显示了如何访问远程对象：

```
empinfo@personnel:markg.emp_names empinfo@personnel:emp_names
```

## 访问远程对象的有效语句

以下语句支持远程对象作为数据库和数据库对象段的一部分并且可以在分布式查询中使用：

- INSERT
- SELECT
- UPDATE
- DELETE
- CREATE VIEW
- CREATE SYNONYM
- CREATE DATABASE
- DATABASE
- LOAD
- UNLOAD
- LOCK
- UNLOCK
- INFO

## 访问远程表

远程表是当前服务器以外的数据库服务器上的表。您可以从当前服务器连接到远程服务器。

在任何时候都只能有一个从本地服务器到远程服务器的活动连接。SinoDB® 不支持使用不同服务器别名的两个相同数据库服务器之间的多个活动连接。因此，如果您使用不同的服务器别名连接相同的远程服务器，则会复用初始连接。

在另一服务器上访问表的通用语法是：

```
database@server:[owner.]table
```

此处 table 可以是表名、视图名称或同义词。您可以选择指定表所有者。有关完整的语法选项，请参阅《*SinoDB® SQL* 指南*:* 语法》中关于 Database 和 Database Object 的两部分文档。

以下示例显示了访问远程表的查询：

```
DATABASE locdb; SELECT l.name, r.assignment FROM rdb@rsys:rtab r,
loctab l WHERE l.empid = r.empid;
```

该查询从本地表 loctab 中访问 name 和 empid 列，从远程表 rtab 中访问 assignment 和 empid 列。将 empid 用作连接列来连接数据。

以下示例显示了访问远程表上的数据并将数据插入本地表的查询：

```
DATABASE locdb; INSERT INTO loctab SELECT * FROM rdb@rsys:rtab;
```

该查询从远程表 rtab 中选择所有数据，然后将数据插入本地表 loctab。

以下示例使用远程数据库 rdb 的 empid 和 priority 列在本地数据库中创建了一个视图。

```
DATABASE locdb; CREATE VIEW myview (empid, empprty)
AS SELECT empid, priority FROM rdb@rsys:rtab;
```

表许可权

访问其他数据库中的表以及远程表的许可权是在表所在的位置进行控制的。访问远程服务器时会使用执行该查询的用户的登录名和密码来进行连接。要访问远程数据，用户必须具有远程表的正确许可权。

处理分布式查询时，数据库服务器在访问远程对象时会忽略当前本地数据库的活动角色。在远程服务器上会使用应用于每个远程数据库的缺省角色。如果没有定义缺省角色，那么用户的权限会定义对每个远程数据库中对象的访问许可权。

限定表引用

可以使用当前数据库和服务器名称来限定对表的引用。如果没有指定限定，那么默认使用当前的数据库和服务器上下文。例如：如果当前数据库是 locdb，当前服务器是 currsys，那么对 loctab 的以下引用，效果是相同的：

```
locdb@currsys:loctab
locdb:loctab
loctab
```

## 其他远程操作

除了查询和更新数据之外，您还可以使用分布式查询框架执行其他远程操作。

打开远程数据库

通过在 DATABASE 语句中指定远程对象，可以打开远程数据库，如下例所示：

```
DATABASE dbname@servername;
DATABASE "//servernam/database";
```

创建远程数据库

在使用 CREATE DATABASE 语句时利用服务器名称来限定数据库名称，可以创建远程数据库。

```
CREATE DATABASE remfoo@rsys;
```

创建远程表的同义词

您可以在 CREATE SYNONYM 语句中使用限定名在另一个数据库或远程表中创建远程表的同义词。例如：以下语句创建了 rdb@srsys:rtab 的同义词：

```
CREATE SYNONYM myrtab FOR rdb@rsys:rtab;
```

同义词可以同时存在于本地和远程服务器上。在上述示例中，rtab 本身有可能是 rdb2@rsys2:rtab2 的同义词。检索目录信息时会沿着同义词链查询，直至找到表所在的物理数据库和服务器。如果同义词最终指回其自身，那么会返回一个错误。

# 监视分布式查询

使用 onstat -x 命令可以显示来自分布式查询协调者的事务信息。

在位置 5 处的以下标志代码用于分布式查询：

C
分布式查询协调者
S
分布式查询参与者
B
分布式查询协调者和参与者

有关使用 onstat -x 的更多信息，请参阅《*SinoDB*® 管理员参考》。

相关链接
《*SinoDB* 管理员参考》: *onstat -x* 命令: 显示数据库服务器事务信息

## 服务器环境和分布式查询

设置 DEADLOCK_TIMEOUT 配置参数和 PDQPRIORITY 环境变量可以指定分布式查询的信息。

DEADLOCK_TIMEOUT 配置参数会指定数据库服务器线程等待获取锁定的最大秒数。如果强制分布式事务等待的时间超过了指定的秒数，那么拥有该事务的线程会假定存在多个服务器死锁，并会返回以下错误消息：

```
-143 ISAM error: deadlock detected.
```

建立连接时，会话的 PDQPRIORITY 有效值会发送到远程站点。协调者中该参数的后续变化不会反映在远程站点上。然而，该环境变量的确切行为取决于分布式查询中数据库服务器的角色（协调者或参与者）。

PDQPRIORITY 对于不同的服务器版本具有不同的语法和语义。有关设置 PDQPRIORITY 的信息，请参阅服务器的《*SinoDB*® 性能指南》。

相关链接
《*SinoDB* 管理员参考》: *DEADLOCK_TIMEOUT* 配置参数
《*SinoDB SQL* 指南: 参考》: *PDQPRIORITY* 环境变量

## 分布式查询的日志记录模式限制

要在 SinoDB® 数据库服务器环境中执行分布式查询，所有参与的数据库必须符合事务日志记录模式：

- 只有当所有参与的数据库都符合 ANSI 标准时，符合 ANSI 标准的数据库才支持分布式查询。
- 只有当所有参与的数据库不使用事务日志记录时，才能在不支持事务日志记录的数据库上执行分布式查询。
- 如果所有其他数据库使用显示事务日志记录，那么在不符合 ANSI 标准但使用显示事务日志记录的数据库上支持分布式查询。

在最后一种情况下，参与的数据库使用缓冲日志记录还是无缓冲日志记录并不影响其支持分布式操作。在 X/Open 分布式事务处理 (DTP) 环境中，所有数据库必须使用无缓冲日志记录。请参阅《*SinoDB*® 管理员指南》以获取有关数据库日志记录模式和 X/Open DTP 的更多信息。

# 事务处理

在事务处理环境中使用分布式查询时，请注意事务隔离级别的影响、SET LOCK MODE 语句结合 DEADLOCK_TIMEOUT 配置参数一起使用时的影响以及两阶段提交协议。

## 隔离级别

在远程站点的事务开始时，事务的隔离级别会发送到远程服务器。如果在事务过程中隔离级别发生了变化，那么会将新值发送到远程站点。

## DEADLOCK_TIMEOUT 和 SET LOCK MODE

使用分布式查询时，可以结合 DEADLOCK_TIMEOUT 配置参数使用 SET LOCK MODE 语句帮助防止服务器死锁。

请求 SET LOCK MODE 的 WAIT 选项时，数据库服务器会针对可能的死锁进行保护。但是，如果数据库服务器发现可能会出现死锁，则会终止操作并返回错误。

DEADLOCK_TIMEOUT 配置参数指定了数据库服务器线程等待获取锁定的最大秒数。该值是 SET LOCK MODE WAIT 语句使用的缺省值。只有当在同一事务内获取当前数据库服务器和远程数据库服务器上的锁定时该值才适用。

相关链接

*《SinoDB 管理员参考》: DEADLOCK_TIMEOUT 配置参数*

*《SinoDB 管理员指南》: 多阶段提交协议*

*《SinoDB SQL 指南: 语法》: SET LOCK MODE 语句*

## 两阶段提交和恢复

两阶段提交协议用于确保分布式查询在多个数据库服务器之间统一提交或回滚。数据库服务器对在多个数据库服务器上修改数据的任何事务自动使用两阶段提交协议。

相关链接

*《SinoDB 管理员指南》: 多阶段提交协议*

# 第 III 部分

# 对象关系数据库

## 在 SinoDB® 中创建和使用扩展数据类型

本章描述可以用来构建对象关系数据库的扩展数据类型。对象关系与数据库设计的特定方法或模型不相关，相反，它指的是任何使用 SinoDB® 功能来扩展数据库功能的数据库。

对象关系数据库不是与关系数据库相对立的，相反，它是对关系数据库中已存在的功能的扩展。通常，使用 SinoDB® 中功能的某种组合来扩展数据库可以存储和处理的数据种类。这些功能包括扩展数据类型、智能大对象、类型和表继承、用户定义的强制转型和用户自定义例程 (UDR)。手册这一部分中的各章节描述了许多这些功能。有关 UDR 的信息，请参阅《*SinoDB® 用户自定义例程和数据类型开发者指南*》和《*SinoDB® SQL 指南: 教程*》。

要获取对象关系数据库的示例，可以创建 superstores_demo 数据库，该数据库包含 SinoDB® 提供的一些功能的示例。有关如何创建 superstores_demo 数据库的信息，请参阅《*SinoDB® DB-Access 用户指南*》。

### SinoDB® 数据类型

图 *22: 选择数据类型* 在第30页提供了根据将存储的数据类型来为表列选择正确数据类型的图表。下图显示了数据类型的层次结构，此层次结构反映了数据库服务器管理数据类型的方式。

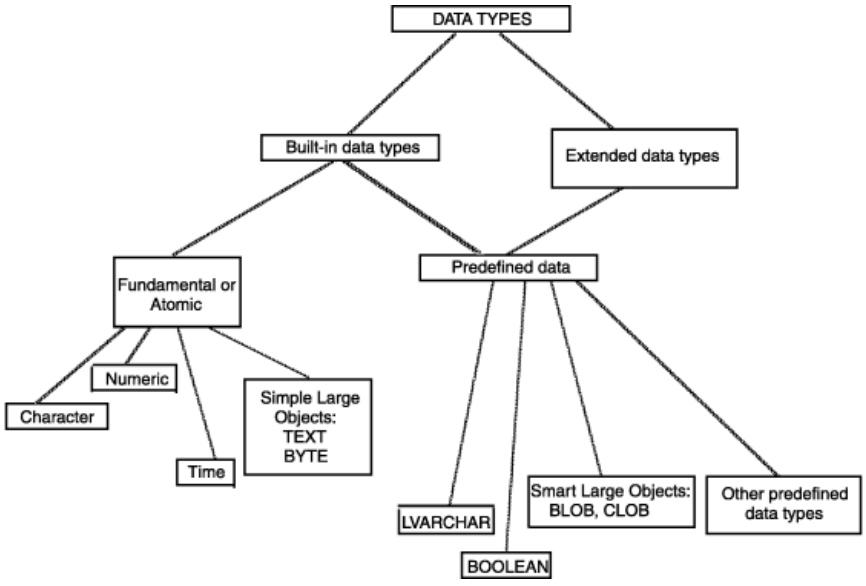


图 25: SinoDB® 数据类型

## 基本或原子数据类型

所有 SinoDB® 数据库服务器都支持基本或原子数据类型。因为这些类型是可以在 SELECT 语句中指定的最小单位，所以它们是基本类型。只有 SinoDB® 才支持扩展数据类型和预定义数据类型。因为预定义数据类型与扩展数据类型共享某些特征，但它们是由数据库服务器提供的，所以它们位于独立的类别中。

有关基本数据类型的说明，请参阅选择数据类型 在第29页。

## 预定义数据类型

就像提供基本数据类型一样，数据库服务器还提供预定义数据类型。但是，预定义数据类型与扩展数据类型具有某些共同特征。

BOOLEAN 和 LVARCHAR 数据类型

BOOLEAN 和 LVARCHAR 数据类型的行为与内置数据类型相似，不过系统目录表将他们定义为扩展数据类型。

有关更多信息，请参阅选择数据类型 在第29页以及《*SinoDB® SQL* 指南*:* 参考》中的系统目录表。

IDSSECURITYLABEL 数据类型

IDSSECURITYLABEL 数据类型将一个安全标签存储在通过安全策略保护的表中。只有持有 DBSECADM 角色的用户可以创建、变更或删除此数据类型的列。这是内置 DISTINCT OF VARCHAR(128) 数据类型，但是并没有被归类为字符数据，因为其用途仅限于基于标签的访问控制。

有关更多信息，请参阅《*SinoDB® SQL* 指南*:* 参考》中的系统目录表以及《*SinoDB®* 安全指南》。

BLOB 和 CLOB 数据类型

由于您可以随机访问 BLOB 或 CLOB 中的数据，因此 BLOB 和 CLOB 数据类型不是基本数据类型。您可以创建带有 BLOB 和 CLOB 列的表，但不能直接将数据插入这样的列中。必须使用函数才能插入和处理数据。

有关更多信息，请参阅智能大对象 在第85页。

## 其他预定义数据类型

除 BLOB、BOOLEAN、CLOB 和 LVARCHAR 以外，预定义数据类型通常不作为表列的数据类型出现。相反，以下预定义数据类型用于与复杂、用户定义的数据类型和用户自定义例程关联的函数：

- clientbinval
- ifx_lo_spec
- ifx_lo_stat
- impexp
- impexpbin
- indexkeyarray
- lolist
- pointer
- rtnparamtypes
- selfuncargs
- sendrecv
- stat
- stream

有关这些预定义数据类型的更多信息，请参阅《*SinoDB®* 用户自定义例程和数据类型开发者指南》。

## 扩展数据类型

扩展数据类型允许创建数据类型来描述那些无法简单使用内置数据类型表示的数据特征。但是，不能在分布式事务中使用扩展数据类型。下图显示了扩展数据类型。

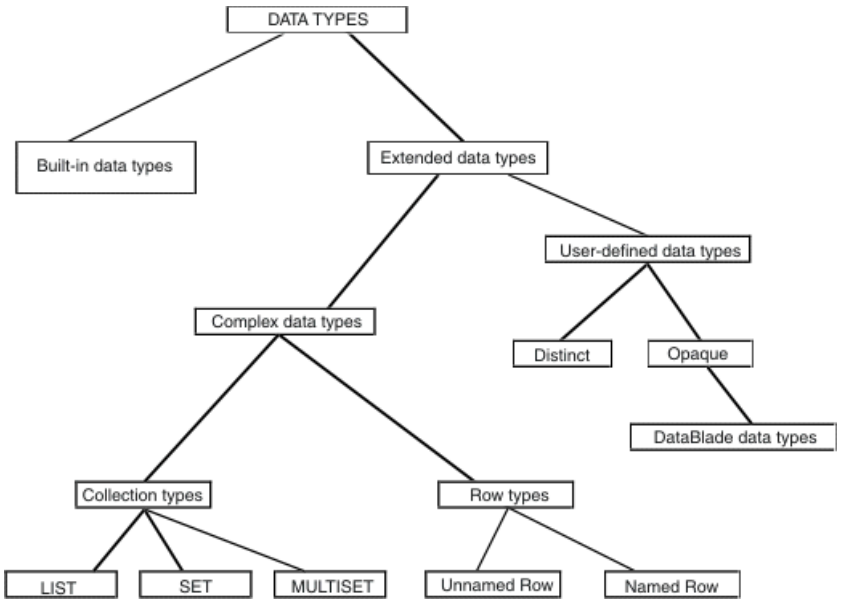


图 26: 扩展数据类型

复杂数据类型

复杂数据类型描述全部属于一种类型（LIST、SET 和 MULTISET）的数据对象的集合或不同类型（命名行和未命名行）的对象组。

用户定义的数据类型
用户定义的数据类型 (UDT) 并不是由数据库服务器提供的数据类型。必须提供数据库服务器管理不透明数据类型或单值数据类型所需的所有信息。

SinoDB® 支持两个类别的用户定义的数据类型：单值数据类型和不透明数据类型。

相关链接

《*SinoDB SQL* 指南: 语法》: 用户自定义数据类型

单值数据类型

单值数据类型是使用 CREATE DISTINCT TYPE 语句创建的封装数据类型。单值数据类型与其基于的数据类型具有相同的表示法，但却不同于该数据类型。可以根据内置类型、不透明类型、命名行类型或其他单值类型创建单值数据类型。不能根据下列任何数据类型创建单值数据类型：

- BIGSERIAL、SERIAL 和 SERIAL8
- 集合类型
- 未命名行类型

因为单值数据类型继承其源数据类型的结构，所以创建单值数据类型时，将隐式定义数据类型的结构。也可以定义对单值数据类型操作的函数、运算符和聚合。

有关单值数据类型的信息，请参阅强制转型单值数据类型 在第110页、《*SinoDB® SQL* 指南: 语法》和《*SinoDB® SQL* 指南: 参考》。

不透明数据类型

不透明数据类型是使用 CREATE OPAQUE TYPE 语句创建的封装数据类型。创建不透明数据类型时，必须显式定义该数据类型的结构以及对该不透明数据类型进行操作的函数、运算符和聚合。可以像使用内置类型那样使用不透明数据类型来定义列和程序变量。

有关创建不透明数据类型的信息，请参阅《*SinoDB®* 用户自定义例程和数据类型开发者指南》和《*SinoDB® SQL* 指南: 语法》。

DataBlade® 数据类型

图 *26: 扩展数据类型* 在第84页中的图包括 DataBlade® 数据类型。DataBlade® 是一组用户定义的数据类型和用户自定义例程，为专用应用程序提供工具。例如，不同的 DataBlade® 数据类型提供了用于管理图像、视频和地理信息的工具。此类应用程序经常需要不透明数据类型以及其他用户定义的数据类型。有关开发 DataBlade® 模块的信息，请参阅《*SinoDB® DataBlade® API* 程序员指南》。有关星瑞格®提供的 DataBlade® 模块的信息，请联系客户代表。

# 智能大对象

智能大对象是基于 BLOB 或 CLOB 数据类型定义的对象。智能大对象允许应用程序随机访问列数据，这表示可以按任意顺序读取或写入 BLOB 或 CLOB 列的任何部分。您可以创建 BLOB 或 CLOB 列来存储二进制数据或字符数据。

## BLOB 数据类型

可以使用 BLOB 数据类型来存储程序可以生成的任何数据：图形图像、卫星图像、视频剪辑、音频剪辑或者由任何文字处理器或电子表格保存的格式化文档。在 BLOB 列中，数据库服务器允许任何类型的、具有任何长度的数据。

与 CLOB 对象相似，BLOB 对象存储在与普通行数据不同的磁盘区域的完整磁盘页中。

与 CLOB 相比，BLOB 数据类型的优点是它接受任何数据。在其他方面，BLOB 数据类型的优点和缺点与 CLOB 数据类型的优点和缺点相同。

## CLOB 数据类型

您可以使用 CLOB 数据类型来存储文本块。此数据类型用来存储 ASCII 文本数据，包括诸如 HTML 或 PostScript ™之类的格式化文本。虽然可以在 CLOB 对象中存储任何数据，但是 SinoDB® 工具期望 CLOB 对象可打印，所以将此数据类型限制为可打印的 ASCII 文本。

CLOB 值不与其所在的行一起存储。在完整的磁盘页（通常是与行分开的区域）中分配它们。（有关更多信息，请参阅《*SinoDB®* 管理员指南》。）

CLOB 数据类型与 TEXT 数据类型类似，但 CLOB 数据类型具有下列优点：

- 应用程序可以读写 CLOB 对象的任何部分。
- 由于应用程序可以访问 CLOB 对象的任何部分，因此访问速度可以快很多。
- 缺省特征相对易于覆盖。数据库管理员能够在列级别覆盖智能大对象空间的缺省特征。应用程序员可以在创建 CLOB 对象时覆盖列的某些缺省特征。
- 可以使用等于运算符 (=) 来测试两个 CLOB 值是否相等。
- CLOB 对象在发生系统故障时是可恢复的，并且在 DBA 或应用程序员指定了事务隔离方式的情况下遵守该方式。（恢复 CLOB 对象要求数据库系统有必需的资源以提供足够大的缓冲区来处理 CLOB 对象。）
- 可以使用 CLOB 数据类型来为用户定义的数据类型提供大存储空间。
- DataBlade® 开发者可以对 CLOB 数据类型创建索引。

CLOB 数据类型的缺点如下：

- 在完整的磁盘页中分配，因此 CLOB 长度较短时将浪费空间。
- 对您在 SQL 语句中可以如何使用 CLOB 列有限制。（请参阅使用智能大对象 在第85页。）
- 对所有 SinoDB® 数据库服务器都不可用。

## 使用智能大对象

要存储数据类型为 BLOB 或 CLOB 的列，必须分配智能大对象空间。智能大对象空间是以最有效的方式存储 BLOB 和 CLOB 数据的逻辑存储器单元格。可以编写允许用户访存和存储 BLOB 或 CLOB 数据的 SinoDB® ESQL/C 程序。应用程序员想要直接访问和处理智能大对象可以查阅《*SinoDB® ESQL/C* 程序员指南》。

在任何 SQL 语句中（无论是交互式的还是编程的），都不能在以下上下文中使用 BLOB 或 CLOB 列：

- 在算术或 Boolean 表达式中
- 在 GROUP BY 或 ORDER BY 子句中
- 在 LIKE 或 MATCHES 条件中
- 在 UNIQUE 测试中
- 用于建立索引，作为 SinoDB® B 型树索引的一部分

  但是，DataBlade® 开发者可以对 CLOB 列创建索引。

在以交互方式输入的 SELECT 语句中，BLOB 或 CLOB 列可以：

- 在创建表时通过 DEFAULT NULL 子句将 NULL 值指定为缺省值
- 在创建表时使用 NOT NULL 约束来拒绝 NULL 值
- 使用 IS [NOT] NULL 谓词进行测试

在 SinoDB® ESQL/C 程序中，可以使用 ifx_lo_stat() 函数来确定 BLOB 或 CLOB 数据的长度。

### 复制智能大对象

SinoDB® 提供了可以从 SQL 语句中调用的函数，以导入和导出智能大对象。下表显示了智能大对象函数。

表 3: 智能大对象的 SQL 函数

| 函数名 | 用途 |
|---|---|
| FILETOBLOB() | 将文件复制到 BLOB 列中 |
| FILETOCLOB() | 将文件复制到 CLOB 列中 |
| LOCOPY() | 将 BLOB 或 CLOB 数据复制到另一个 BLOB 或 CLOB 列中 |
| LOTOFILE() | 将 BLOB 或 CLOB 数据复制到文件中 |

有关智能大对象函数的详细信息和语法，请参阅《*SinoDB® SQL* 指南: 语法》中的表达式部分。

限制: 不允许在 BLOB 和 CLOB 数据类型之间进行强制转型。

## 复杂数据类型

复杂数据类型通常是其他现有数据类型的组合。例如：可以创建其组件包括内置类型、不透明类型、单值类型或其他复杂类型的复杂数据类型。与用户定义的类型相比，复杂数据类型的一项重要优点是用户可以访问和处理复杂数据类型的单个组件。

相反，内置类型和用户定义的类型是独立（封装）数据类型。因此，访问不透明数据类型的组件值的唯一方法是通过对不透明类型定义的函数进行。

下图显示了 SinoDB® 支持的复杂数据类型以及用于创建复杂数据类型的语法。

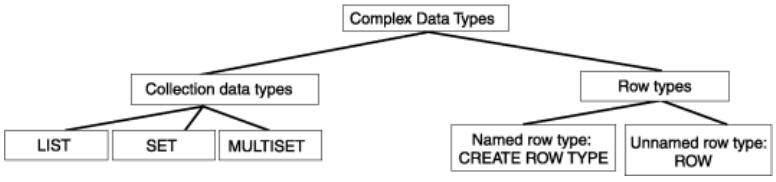


图 27: 复杂数据类型

上图说明的复杂数据类型支持以下扩展数据类型：

集合类型

每当必须在表单元格中存储和处理数据集合时，可以使用集合类型。您可以对列指定集合类型。

行类型

行类型通常包含多个字段。当要在列或变量中存储多种类型的数据时，可以创建行类型。行类型具有两类：命名行类型和未命名行类型。可以对列和变量指定未命名行类型。可以对列、变量、表或视图指定命名行类型。当对表指定命名行类型时，表是类型表。类型表的主要优点是可用来定义继承层次结构。

有关如何对本章描述的复杂数据类型执行 SELECT、INSERT、UPDATE 和 DELETE 操作的更多信息，请参阅《*SinoDB® SQL* 指南: 教程》。

## 集合数据类型

集合数据类型使您能够在表的单个行内存储和处理数据集合。集合数据类型有两个组件：类型构造函数（它确定集合类型是 SET、MULTISET 还是 LIST）和元素类型（它指定集合可包含的数据类型）。（下列各节详细描述了 SET、MULTISET 和 LIST 集合类型。）

集合的元素几乎可以具有任何数据类型。（要获取例外情况的列表，请参阅对集合的限制 在第90页。）集合的元素就是集合包含的值。在包含下列值的集合中：{'blue', 'green', 'yellow', and 'red'}, 'blue' 表示集合中的单个元素。集合中的每个元素都必须具有相同的类型。例如：元素类型为 INTEGER 的集合只能包含整数值。

集合的元素类型可以表示单个数据类型（列），也可以表示多个数据类型（行）。在以下示例中，col_1 列表示整数的 SET：

```
col_1 SET(INTEGER NOT NULL)
```

要定义包含多个数据类型的集合数据类型，可使用命名行类型或未命名行类型。在以下示例中，col_2 列表示包含 name 和 salary 字段的行的 SET：

```
col_2 SET(ROW(name VARCHAR(20), salary INTEGER) NOT NULL)
```

重要: 定义集合数据类型时，必须将 NOT NULL 约束纳入类型定义的一部分。不允许对集合数据类型施加任何其他列约束。

在将列定义为具有集合数据类型之后，可以对该集合执行下列操作：

- 选择和修改集合的单个元素（仅从 SinoDB® ESQL/C 程序中）。
- 对集合包含的元素计数。
- 确定集合是否包含特定值。

有关用于创建集合数据类型的语法的信息，请参阅《*SinoDB® SQL* 指南: 语法》中的数据类型部分。有关如何将一种集合类型的值转换为另一集合类型的值的信息，请参阅《*SinoDB® SQL* 指南: 教程》。

集合中的 Null 值

集合不能包含 NULL 元素。然而，当集合是行类型时，可以对集合包含的行类型的任何或所有字段插入 NULL 值。假设创建以下带有集合列的表：

```
CREATE TABLE tab1 (col1 INT,
       col2 SET(ROW(a INT, b INT) NOT NULL));
```

因为只有行类型的组成字段指定了 NULL 值，所以允许下列语句：

```
INSERT INTO tab1 VALUES ( 25,"SET{ROW(NULL, NULL)}");

INSERT INTO tab1 VALUES ( 35,"SET{ROW(4, NULL)}");

INSERT INTO tab1 VALUES ( 45,"SET{ROW(14, NULL), ROW(NULL,5)}");

UPDATE tab1 SET col2 = "SET{ROW(NULL, NULL)}" WHERE col1 = 45;
```

然而，由于集合元素指定了 NULL 值，因此下列每个语句都会返回一条错误消息：

```
INSERT INTO tab1 VALUES ( 45, "SET{NULL)}");

UPDATE tab1 SET col2 = "SET{NULL}" WHERE col1 = 55;
```

SET 集合类型

SET 是元素的无序集合，其中每个元素都是唯一的。当要存储具有下列特征的元素的集合时，请将列定义为 SET 集合类型：

- 元素不包含重复值。
- 元素不具有与其相关联的特定顺序。

为了说明如何使用 SET，假设人力资源部门需要有关公司每个员工的家属信息。可使用集合类型来定义 employee 表中的列，用于存储雇员家属的姓名。以下语句创建一个表，其中 dependents 列定义为 SET：

```
CREATE TABLE employee
(
   name        CHAR(30),
   address     CHAR (40),
   salary      INTEGER,
   dependents  SET(VARCHAR(30) NOT NULL)
);
```

对任何给定行的 dependents 列执行的查询都将返回雇员的所有家属姓名。在本例中，由于每个员工的家属集合不应包含任何重复值，所以 SET 是正确的集合类型。定义为 SET 的列确保集合中的每个元素都是唯一的。

为了说明如何定义具有行类型元素的集合类型，假设要让 dependents 列包括雇员家属的姓名和生日。在以下示例中，将 dependents 列定义为 SET，其元素类型是行类型：

```
CREATE TABLE employee
(
   name        CHAR(30),
   address     CHAR (40),
   salary      INTEGER,
   dependents  SET(ROW(name VARCHAR(30), bdate DATE) NOT NULL)
);
```

dependents 列中的集合的每个元素都包含 name 值和 bdate 值。employee 表的每一行都包含有关员工的信息以及该员工家属的姓名和生日的集合。例如：如果某个雇员没有家属，那么 dependents 列的集合是空的。如果某个雇员有 10 个家属，那么该集合应包含 10 个元素。

MULTISET 集合类型

MULTISET 是元素的集合，其中元素可以有重复值。例如：整数的 MULTISET 可包含集合 {1,3,4,3,3}，此集合有重复的元素。当要存储具有下列特征的元素的集合时，可将列定义为 MULTISET 集合类型：

- 元素可能不是唯一的。
- 元素不具有与其相关联的特定顺序。

为了说明如何使用 MULTISET，假设人力资源部门想要跟踪奖励给公司雇员的奖金。要跟踪每个雇员一段时间内的奖金，可使用 MULTISET 定义表中的一个列，该列记录每个雇员得到的所有奖金。在以下示例中，将 bonus 列定义为 MULTISET：

```
CREATE TABLE employee
(
   name      CHAR(30),
   address   CHAR (40),
   salary    INTEGER,
   bonus     MULTISET(MONEY NOT NULL)
```

```
);
```

可使用此语句中的 bonus 列来存储和访问每个雇员的奖金集合。对任何给定行的 bonus 列执行的查询都将返回雇员收到的每笔奖金的美元金额。由于雇员可能会收到多笔相同金额的奖金（导致集合包含的元素并不全是唯一的），因此将 bonus 列定义为允许重复值的 MULTISET。

LIST 集合类型

LIST 是元素的有序集合，其中元素可以有重复值。LIST 与 MULTISET 的不同之处在于 LIST 中的每个元素在集合中都是有序的。列表中元素的顺序与将值插入 LIST 的顺序相对应。当要存储具有下列特征的元素的集合时，可将列定义为 LIST 集合类型：

- 元素具有与其相关联的特定顺序。
- 元素可能不是唯一的。

为了说明如何使用 LIST，假设销售部门要对每个销售人员的销售总额进行月度记录。可使用 LIST 来定义表中的一个列，该列包含每个销售人员的月度销售总额。以下示例创建一个表，其中 month_sales 列定义为 LIST。LIST 中的第一个条目（元素）具有序数位置 1，可以与一月相对应，序数位置为 2 的第二个元素与二月相对应，依此类推：

```
CREATE TABLE sales_person
(
   name         CHAR(30),
   month_sales  LIST(MONEY NOT NULL)
);
```

可使用此语句中的 month_sales 列来存储和访问每个销售人员的月度销售总额。更确切地说，可对 month_sales 列执行查询来了解以下方面：

- 某个销售人员在指定月份内的总销售量
- 每个销售人员在指定月份内的总销售量

嵌套集合类型

嵌套的集合是包含另一集合类型的集合类型。可以将任何集合类型嵌套在另一集合类型中。对集合类型可以具有的嵌套深度没有实际的限制。但是，对已经嵌套了一两层以上的集合执行插入或更新可能比较困难。

以下示例显示了创建定义为嵌套集合类型的列的数种方法：

```
col_1 SET(MULTISET(VARCHAR(20) NOT NULL) NOT NULL);

col_2 MULTISET(ROW(x CHAR(5), y SET(INTEGER NOT NULL))
NOT NULL);

col_3 LIST(MULTISET(ROW(a CHAR(2), b INTEGER) NOT NULL)
NOT NULL);
```

有关如何访问嵌套集合的信息，请参阅《*SinoDB® SQL* 指南*:* 教程》。

将集合类型添加至现有表中

可以使用 ALTER TABLE 语句来添加或删除具有集合类型（或任何其他数据类型）的列。例如：以下语句将已定义为 SET 的 flowers 列添加至 nursery 表：

```
ALTER TABLE nursery ADD flower SET(VARCHAR(30) NOT NULL)
```

不能修改具有集合类型的现有列，也不能将不具有集合类型的列转换为具有集合类型。

有关添加和删除集合类型列的更多信息，请参阅《*SinoDB® SQL* 指南*:* 语法》中的 ALTER TABLE 语句。

对集合的限制

不能使用下列任何数据类型作为集合的元素类型：

- TEXT
- BYTE
- SERIAL
- SERIAL8
- BIGSERIAL

不能使用 CREATE INDEX 语句来对集合创建索引，也不能创建集合列的函数索引。

## 命名行类型

命名行类型是在单个名称下定义的一组字段。字段指的是行类型的组件，不应与列混淆，列只与表相关联。命名行类型的字段与 C 语言结构的字段或面向对象编程中的类的成员相似。在创建命名行类型之后，对行类型指定的名称表示数据库中的唯一类型。要创建命名行类型，请指定行类型的名称并指定其组成字段的名称和数据类型。以下示例显示如何创建名为 person_t 的命名行类型：

```
CREATE ROW TYPE person_t
(
   name     VARCHAR(30) NOT NULL,
   address  VARCHAR(20),
   city     VARCHAR(20),
   state    CHAR(2),
   zip      VARCHAR(9),
   bdate    DATE
);
```

person_t 行类型包含 6 个字段：name、address、city、state、zip 和 bdate。创建命名行类型后，可以像使用任何其他数据类型那样使用它。person_t 可以出现在任何使用其他数据类型的位置。以下 CREATE TABLE 语句使用 person_t 数据类型：

```
CREATE TABLE sport_club
(
   sport     CHAR(20),
   sportnum  INT,
   member    person_t,
   since     DATE,
   paidup    BOOLEAN
)
```

可以使用大多数数据类型来定义行类型的字段。有关在行类型中不受支持的数据类型的信息，请参阅对命名行类型的限制 在第91页。

有关用来创建命名行类型的语法，请参阅《*SinoDB® SQL* 指南: 语法》中的 CREATE ROW TYPE 语句。有关如何对行类型值进行强制转型的信息，请参阅创建和使用用户定义的强制转型 在第104页。

何时使用命名行类型

命名行类型是一种在 SinoDB® 中创建新数据类型的方法。创建命名行类型时，您将定义数据库服务器已知的数据类型的字段模板。因此，行类型的字段定义与表的列定义类似：两者都是根据数据库服务器已知的数据类型构造的。

当需要一个类型来充当用户必须访问的组件值的容器时，可以创建命名行类型。例如，由于用户需要直接访问地址的单个组件值（如街道、城市、省/自治区/直辖市和邮政编码），因此您可以创建命名行类型来支持地址值。在将地址类型创建为命名行类型后，用户始终可以直接访问每个字段。

相反，如果创建不透明数据类型来处理地址值，那么使用 C 语言数据结构来存储所有地址信息。因为不透明类型的组件值是封装的，所以必须定义函数来抽取街道、城市、省/自治区/直辖市和邮政编码的组件值。因此，不透明数据类型是定义和使用起来都更为复杂的类型。

在定义数据类型之前，请确定该类型是否仅仅是用户可以直接访问的一组值的容器。如果该类型符合此描述，那么使用命名行类型。

选择命名行类型的名称

可以对命名行类型指定您喜欢的任何名称，只要该名称不违反为 SQL 标识符建立的约定。《*SinoDB® SQL* 指南:语法》 中的标识符部分描述了有关 SQL 标识符的约定。为了避免类型和表名冲突，本手册中的示例在行类型名末尾使用 _t 字符来指定命名行类型。

您必须拥有 Resource 权限才能创建命名行类型。由于所有数据类型共享同一个名称空间，因此，对命名行类型指定的名称不应该与数据库中存在的任何其他数据类型的名称相同。在符合 ANSI 标准的数据库中，owner.type 组合在数据库中必须是唯一的。在不符合 ANSI 标准的数据库中，名称在数据库中必须是唯一的。

重要： 必须授予其他用户对命名行类型的 USAGE 权限，他们才能使用该类型。

相关链接

对用户定义的类型的 *Usage* 权限 在第61页

对命名行类型的限制

本节描述使用命名行类型时适用的限制。

对数据类型的限制

创建包含大对象列的类型表时，应该使用 BLOB 或 CLOB 数据类型而不是 TEXT 或 BYTE 数据类型。为了与较早版本兼容，您可以创建包含 TEXT 或 BYTE 字段的命名行类型，并使用该类型将现有的（无类型）表重新创建为类型表。然而，虽然可使用包含 TEXT 或 BYTE 字段的命名行类型来创建类型表，但不能将这样的行类型用作列。您可以对类型表或列指定包含 BLOB 或 CLOB 字段的命名行类型。

对约束的限制

在 CREATE ROW TYPE 语句中，只能对命名行类型的字段指定 NOT NULL 约束。必须在 CREATE TABLE 语句中定义所有其他约束。有关更多信息，请参阅《*SinoDB® SQL* 指南:语法》中的 CREATE TABLE 语句。

对索引的限制

不能使用 CREATE INDEX 语句来对命名行类型列创建索引。但是，可使用用户自定义例程来为行类型列创建函数索引。

对串行数据类型的限制
不能将包含 SERIAL、SERIAL8 或 BIGSERIAL 数据类型的命名行类型用作表中的列类型。

当数据库服务器尝试创建表时，下列语句将返回错误：

```
CREATE ROW TYPE row_t (s_col SERIAL)

CREATE TABLE bad_tab (col1 row_t)
```

但是，可使用包含 SERIAL、SERIAL8 或 BIGSERIAL 数据类型的命名行类型来创建类型表。

有关 SERIAL、SERIAL8 和 BIGSERIAL 类型在表层次结构中的使用和行为的信息，请参阅表层次结构中的 *SERIAL* 类型 在第102页。

使用命名行类型创建类型表

可以创建类型表或无类型表。类型表是指定了命名行类型的表。无类型表是没有指定命名行类型的表。CREATE ROW TYPE 语句创建命名行类型，但不为该行类型的实例分配存储空间。要为命名行类型的实例分配存储空间，必须对某个表指定该行类型。以下示例显示如何创建类型表：

```
CREATE ROW TYPE person_t
(
   name     VARCHAR(30),
   address  VARCHAR(20),
   city     VARCHAR(20),
```

```
   state    CHAR(2),
   zip      INTEGER,
   bdate    DATE
);

CREATE TABLE person OF TYPE person_t;
```

第一个语句创建 person_t 类型。第二个语句创建 person 表，该表包含 person_t 类型的实例。更明确地说，类型表中的每一行都包含对该表指定的命名行类型的一个实例。在上述示例中，person_t 类型的字段定义 person 表的列。

重要： 因为在可以使用命名行类型来定义类型表之前该命名行类型必须存在，所以创建命名行类型的顺序十分重要。

将数据插入类型表与将数据插入无类型表没有什么区别。在将数据插入类型表时，将会创建行类型的实例并将其插入到表中。以下示例显示如何将行插入到 person 表中：

```
INSERT INTO person
VALUES ('Brown, James', '13 First St.', 'San Carlos', 'CA', 94070,
'01/04/1940')
```

INSERT 语句创建 person_t 类型的实例并将其插入表中。有关如何插入、更新和删除对命名行类型定义的列的更多信息，请参阅《*SinoDB® SQL* 指南*:* 教程》。

可使用单个命名行类型来创建多个类型表。在此情况下，每个表都具有唯一的名称，但所有表共享同一类型。

限制： 不能创建作为临时表的类型表。

有关在实现数据模型时使用类型表的优点的信息，请参阅类型继承 在第95页。

更改表的类型

与无类型表相比，类型表的主要优点是可以在继承层次结构中使用。通常，继承允许表获取另一个表的表示法和行为。有关更多信息，请参阅什么是继承# 在第95页。

ALTER TABLE 语句的 DROP 和 ADD 子句允许在类型表和无类型表之间进行切换。ADD 和 DROP 操作都不影响表中存储的数据。

将无类型表转换为类型表

如果要将现有的无类型表转换为类型表，可使用 ALTER TABLE 语句。例如：考虑以下无类型表：

```
CREATE TABLE manager
(
   name         VARCHAR(30),
   department   VARCHAR(20),
   salary       INTEGER
);
```

要将无类型表转换为类型表，命名行类型的字段名和字段类型都必须与现有表的列名和列类型相匹配。例如：要让 manager 表成为类型表，首先必须创建与表的列定义相匹配的命名行类型。以下语句创建 manager_t 类型，其中包含与 manager 表的列相匹配的字段名和字段类型：

```
CREATE ROW TYPE manager_t
(
   name         VARCHAR(30),
   department   VARCHAR(20),
   salary       INTEGER
);
```

在创建要指定给现有无类型表的命名行类型之后，请使用 ALTER TABLE 语句来对表指定该类型。以下语句变更 manager 表并使其成为具有 manager_t 类型的类型表：

```
ALTER TABLE manager ADD TYPE manager_t
```

新的 manager 表与旧表包含相同的列和数据类型，但现在新表具有类型表的优点。

将类型表转换为无类型表

也可以使用 ALTER TABLE 语句将类型表更改为无类型表：

```
ALTER TABLE manager DROP TYPE
```

提示： 将列添加至类型表需要三个 ALTER TABLE 语句来删除类型、添加列和将类型添加至表。

使用命名行类型创建列

类型表和无类型表都可以包含对命名行类型定义的列。对命名行类型定义的列无论是出现在类型表中还是出现在无类型表中都具有相同的行为。在以下示例中，第一个语句创建命名行类型 address_t；第二个语句对 employee 表中的 address 列指定 address_t 类型：

```
CREATE ROW TYPE address_t
(
   street  VARCHAR(20),
   city    VARCHAR(20),
   state   CHAR(2),
   zip     VARCHAR(9)
);

CREATE TABLE employee
(
   name     VARCHAR(30),
   address  address_t,
   salary   INTEGER
);
```

在上述 CREATE TABLE 语句中，address 列具有 address_t 类型的 street、city、state 和 zip 字段。因此，employee 表（它只有 3 列）包含 name、street、city、state、zip 和 salary 的值。使用点符号表示法来访问对行类型定义的列的个别字段。有关使用点符号表示法来访问列字段的信息，请参阅《*SinoDB® SQL* 指南*:* 教程》。

将数据插入指定了行类型的列中时，必须使用 ROW 构造函数来指定该行类型的行文字值。以下示例显示如何使用 INSERT 语句将行插入到 employee 表中：

```
INSERT INTO employee
VALUES ('John Bryant',
  ROW('10 Bay Street', 'Madera', 'CA', 95400)::address_t, 55000);
```

对命名行类型执行插入或更新操作时，不强制执行强类型化。要确保行值具有命名行类型，必须显式地强制转型为命名行类型以生成具有命名行类型的值，如上述示例所示。INSERT 语句插入三个值，其中一个是包含四个值的行类型值。更明确地说，此操作对 name 和 salary 列插入单一值，但它创建 address_t 类型的实例并将其插入到 address 列中。

有关如何插入、更新和删除对行类型定义的列的更多信息，请参阅《*SinoDB® SQL* 指南*:* 教程》。

在其他行类型中使用命名行类型

可以使用命名行类型作为另一个行类型中字段的数据类型。嵌套行类型是包含另一个行类型的行类型。可以将任何行类型嵌套在其他行类型中。对行类型可以具有的嵌套深度没有实际的限制。但是，要对深度嵌套的行类型执行插入或更新，需要注意语法。

对于命名行类型，因为使用命名行类型在另一个行类型中定义列或字段之前该命名行类型必须存在，所以创建行类型的顺序十分重要。在以下示例中，第一个语句创建 address_t 类型，该类型在第二个语句中用来定义 employee_t 类型的 address 字段的类型：

```
CREATE ROW TYPE address_t
(
   street  VARCHAR (20),
   city    VARCHAR(20),
   state   CHAR(2),
   zip     VARCHAR(9)
);

CREATE ROW TYPE employee_t
(
   name     VARCHAR(30) NOT NULL,
   address  address_t,
   salary   INTEGER
);
```

> 重要： 不能递归使用行类型。如果 type_t 是行类型，那么不能使用 type_t 作为包含在 type_t 中的字段的数据类型。

删除命名行类型

要删除命名行类型，请使用 DROP ROW TYPE 语句。只有当类型没有相关性时才能将其删除。如果下列任何条件成立，那么不能删除命名行类型：

- 当前已对表指定该类型。
- 当前已对表中的列指定该类型。
- 当前已对另一个行类型中的字段指定该类型。

以下示例显示如何删除 person_t 类型：

```
DROP ROW TYPE person_t restrict;
```

有关如何从类型层次结构中删除命名行类型的信息，请参阅从类型层次结构中删除命名行类型 在第98页。

## 未命名行类型

未命名行类型是使用 ROW 构造函数创建的一组类型字段。命名行类型与未命名行类型之间的重要区别是不能对表指定未命名行类型。只能使用未命名行类型定义列或字段的类型。另外，未命名行类型只由其结构作为标识符，而命名行类型由其名称作为标识符。行类型的结构由其字段的数目和数据类型组成。

以下语句对 student 表的列指定两个未命名行类型：

```
CREATE TABLE student
(
   s_name  ROW(f_name VARCHAR(20), m_init CHAR(1),
               l_name VARCHAR(20) NOT NULL),
   s_address  ROW(street VARCHAR(20), city VARCHAR(20),
                  state CHAR(2), zip VARCHAR(9))
   );
```

student 表的 s_name 和 s_address 列都包含多个字段。未命名行类型的每个字段都可以具有不同的数据类型。虽然 student 表只有两列，但未命名行类型总共定义了 7 个字段：

- f_name
- m_init
- l_name
- street
- city

- state
- zip

以下示例显示如何使用 INSERT 语句来将数据插入 student 表中：

```
INSERT INTO student
VALUES (ROW('Jim', 'K', 'Johnson'), ROW('10 Grove St.',
'Eldorado', 'CA', 94108))
```

有关如何修改对行类型定义的列的更多信息，请参阅《*SinoDB® SQL* 指南*:* 教程》。

数据库服务器不会区别两个包含相同字段数并且带有相同类型的对应字段的未命名行类型。在未命名行类型的类型检查中，字段名是不相关的。例如：数据库服务器不会对下列未命名行类型加以区别：

```
ROW(a INTEGER, b CHAR(4));
ROW(x INTEGER, y CHAR(4));
```

有关未命名行类型的语法，请参阅《*SinoDB® SQL* 指南*:* 语法》。有关如何对行类型值进行强制转型的信息，请参阅创建和使用用户定义的强制转型 在第104页。

以下数据类型不能是未命名行类型中的字段类型：

- BIGSERIAL
- SERIAL
- SERIAL8
- BYTE
- TEXT

当在未命名行类型的字段定义中指定上述任何类型时，数据库服务器将返回错误。

# 类型和表继承

本章描述类型和表继承并阐述如何创建类型和表层次结构来修改各自层次结构中的类型和表。

## 什么是继承？

继承是允许类型或表获取另一个类型或表的属性的过程。继承属性的类型或表称为子类型或子表。属性被继承的类型或表称为超类型或超表。继承使您能够进行增量修改，以便类型或表可以继承一组一般属性并添加特定于其自身的属性。可使用继承来限制修改程度，以使修改操作不会变更所继承的超类型或超表。

SinoDB® 只对命名行类型和类型表支持继承。SinoDB® 只支持单一继承。对于单一继承，每个子类型或子表都只有一个超类型或超表。

## 类型继承

类型继承仅适用于命名行类型。可使用继承来将命名行类型组成类型层次结构，其中每个子类型都继承超类型（在该超类型下定义了该子类型）的表示法（数据字段）和行为（UDR、聚合和运算符）。类型层次结构具有下列优点：

- 鼓励利用模块来实现数据模型。
- 确保一致地重复使用模式组件。
- 确保不会意外遗漏数据字段。
- 允许类型继承基于另一数据类型定义的 UDR。

## 定义类型层次结构

下图为一个简单类型层次结构的示例，该层次结构包含三个命名行类型。

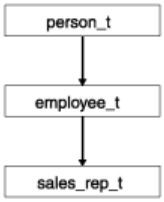


图 28: 类型层次结构的示例

类型层次结构顶部的超类型包含一组字段，所有下层子类型都继承这些字段。在可以创建超类型的子类型之前，该超类型必须存在。以下示例创建图 *28: 类型层次结构的示例* 在第96页显示的类型层次结构的 person_t 超类型：

```
CREATE ROW TYPE person_t
(
   name     VARCHAR(30) NOT NULL,
   address  VARCHAR(20),
   city     VARCHAR(20),
   state    CHAR(2),
   zip      INTEGER,
   bdate    DATE
);
```

要创建子类型，请指定 UNDER 关键字以及超类型（子类型继承其属性）的名称。以下示例说明如何将 employee_t 定义为继承 person_t 所有字段的子类型。此示例添加 person_t 类型中不存在的 salary 和 manager 字段。

```
CREATE ROW TYPE employee_t
(
   salary   INTEGER,
   manager  VARCHAR(30)
)
UNDER person_t;
```

重要： 您必须对超类型具有 UNDER 权限，才能创建继承该超类型属性的子类型。

在图 *28: 类型层次结构的示例* 在第96页的类型层次结构中，sales_rep_t 是 employee_t 的子类型，而 employee_t 是 sales_rep_t 的超类型，就像 person_t 是 employee_t 的超类型一样。以下示例创建 sales_rep_t，它继承 person_t 和 employee_t 的所有字段并添加了四个新字段。由于对子类型所作的修改不会影响其超类型，因此 employee_t 不具有对 sales_rep_t 添加的那四个字段。

```
CREATE ROW TYPE sales_rep_t
(
rep_num      INT8,
region_num   INTEGER,
commission   DECIMAL,
home_office  BOOLEAN
)
UNDER employee_t;
```

sales_rep_t 类型包含 12 个字段：name、address、city、state、zip、bdate、salary、manager、rep_num、region_num、commission 和 home_office。

employee_t 和 sales_rep_t 类型的实例都继承对 person_t 类型定义的所有 UDR。 对 employee_t 定义的其他任何 UDR 都会自动应用于 employee_t 类型的实例，并应用于其子类型 sales_rep_t 的实例，但不会应用于 person_t 的实例。

在上述类型层次结构中，因为每个子类型都是继承单一超类型，所以此类型层次结构是单一继承的一个示例。图 *29: 树结构类型层次结构的示例* 在第97页说明了如何在单个超类型下定义多个子类型。尽管单一继承要求每个子类型都只能继承一个超类型，但是对定义的类型层次结构的深度或宽度没有实际限制。

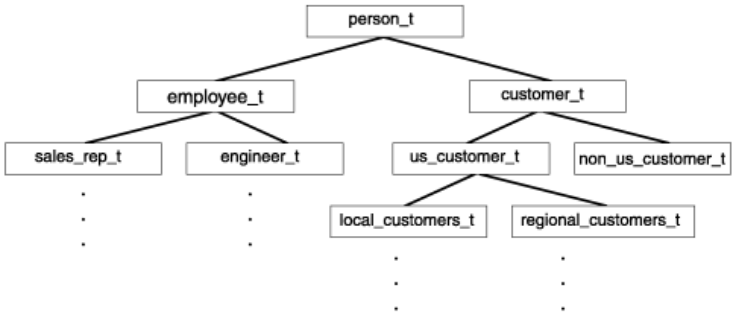


图 29: 树结构类型层次结构的示例

任何层次结构最顶部的类型都称为根超类型。在图 *29: 树结构类型层次结构的示例* 在第97页中，person_t 是层次结构的根超类型。除根超类型以外，层次结构中的任何类型都有可能同时既作为超类型又作为子类型。例如：customer_t 是 person_t 的子类型和 us_customer_t 的超类型。在层次结构中位于较低层的子类型包含根超类型的属性，但不直接从根超类型继承属性。例如：us_customer_t 只有一个超类型 customer_t，但由于 customer_t 本身是 person_t 的子类型，因此 customer_t 从 person_t 继承的字段和例程也由 us_customer_t 继承。

相关链接

对命名行类型的 *Under* 权限 在第62页

## 类型层次结构中类型的例程重载

例程重载是指将一个名称指定给多个例程并指定不同类型的参数来供例程对其进行操作。在类型层次结构中，子类型自动继承对其超类型定义的例程。然而，可以对子类型定义新的例程以覆盖继承的同名例程。例如：假设对类型 person_t 创建了 getinfo() 例程，该例程返回类型为 person_t 的实例的姓氏和生日。您可以对类型 employee_t 注册另一个 getinfo() 例程，该例程返回 employee_t 的实例的姓氏和薪资。这样您就可以重载例程，以便类型层次结构中的每个类型都有自定义的例程，如下图所示。

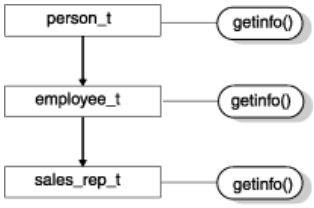


图 30: 类型层次结构中例程重载的示例

当您重载例程以便在类型层次结构中为不同类型使用相同的名称但不同的参数定义例程时，指定的参数将确定执行哪个例程。例如：如果使用类型为 employee_t 的参数来调用 getinfo()，那么对类型 employee_t 定义的 getinfo() 例程将覆盖所继承的同名例程。同样，如果对类型 sales_rep_t 定义另一个 getinfo()，那么使用类型为 sales_rep_t 的参数来调用 getinfo() 将覆盖 sales_rep_t 从 employee_t 继承的例程。

有关如何创建和注册用户自定义例程 (UDR) 的信息，请参阅《*SinoDB*® 用户自定义例程和数据类型开发者指南》。

## 继承和类型可替代性

在类型层次结构中，子类型自动继承对其超类型定义的所有例程。因此，如果使用子类型的参数调用例程，并且没有对该子类型定义例程，那么数据库服务器可调用对超类型定义的例程。类型可替代性指的是在需要超类型实例时以子类型实例替代。例如：假设创建例程 p_info()，该例程接受类型为 person_t 的参数并返回类型为 person_t 的实例的姓氏和生日。如果没有注册其他 p_info() 例程，并且使用类型为

employee_t 的参数来调用 p_info()，那么该例程将返回类型为 employee_t 的实例的姓名和生日字段（从 person_t 继承而来）。由于 employee_t 会继承其超类型 person_t 的函数，因此这是可能发生的。

通常，当数据库服务器尝试对例程进行求值时，数据库服务器将搜索与您调用例程时指定的例程名和参数相匹配的特征符。如果找到这样的例程，那么数据库服务器将使用此例程。如果找不到匹配例程，那么数据库服务器尝试查找以下例程：具有相同的名称，并且其参数类型是调用例程时指定的参数类型的超类型。例如：假设数据库服务器在使用子类型为 sales_rep_t 的参数调用 get() 例程时搜索可以使用的例程。虽然没有对 sales_rep_t 类型定义的 get() 例程，但是数据库服务器也会搜索例程，直到在层次结构中找到对超类型定义的 get() 例程为止。在本例中，没有对 sales_rep_t 及其超类型 employee_t 定义 get() 例程。但是，由于为 person_t 定义了例程，因此将调用此例程来对 sales_rep_t 的实例执行操作。

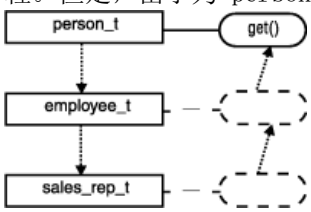


图 31: 数据库服务器如何在类型层次结构中搜索例程的示例

数据库服务器搜索可使用的例程的过程称为例程解析。有关例程解析的更多信息，请参阅《*SinoDB*® 用户自定义例程和数据类型开发者指南》。

### 从类型层次结构中删除命名行类型

要从类型层次结构中删除命名行类型，请使用 DROP ROW TYPE 语句。然而，只有当类型不具有相关性时才可以将其删除。如果下列任何一个条件成立，则不能删除命名行类型：

- 当前已对表指定该类型。
- 该类型是另一类型的超类型。

以下示例显示如何删除 sales_rep_t 类型：

```
DROP ROW TYPE sales_rep_t RESTRICT;
```

要删除超类型，首先必须删除每个从该超类型继承属性的子类型。应该按照类型创建顺序的逆向顺序来删除类型层次结构中的类型。例如：要删除继承和类型可替代性 在第97页显示的 person_t 类型，首先必须按以下顺序删除其子类型：

```
DROP ROW TYPE sale_rep_t RESTRICT;
DROP ROW TYPE employee_t RESTRICT;
DROP ROW TYPE person_t RESTRICT;
```

重要： 要删除类型，您必须是数据库管理员或该类型的所有者。

## 表继承

只有在命名行类型上定义的表才支持表继承。表继承是一项属性，允许表从表层次结构中位于该表之上的超表继承行为（约束、存储选项和触发器）。表层次结构是可以在各个表之间定义的关系，在此关系中，子表继承超表的行为。表继承具有下列优点：

- 鼓励利用模块来实现数据模型。
- 确保一致地重复使用模式组件。
- 允许您构造作用域是表层次结构中的某些表或全部表的查询。

在表层次结构中，子表自动继承其超表的下列属性：

- 所有约束定义（主键、唯一约束和引用约束）
- 存储选项

- 所有触发器
- 索引
- 访问方法

## 类型层次结构与表层次结构之间的关系

必须对表层次结构中的每个表指定相应类型层次结构中的命名行类型。下图显示了类型层次结构与表层次结构之间可能存在的关系示例。

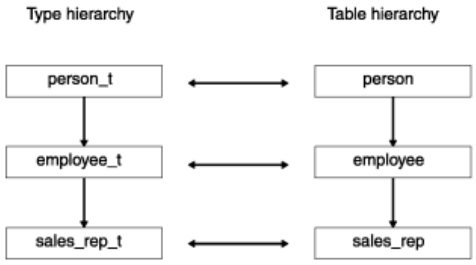


图 32: 类型层次结构与表层次结构之间的关系示例

但是，也可以定义类型层次结构中的命名行类型不必与表层次结构中的表一一对应。下图显示了如何创建只将某些命名行类型指定给表的类型层次结构。

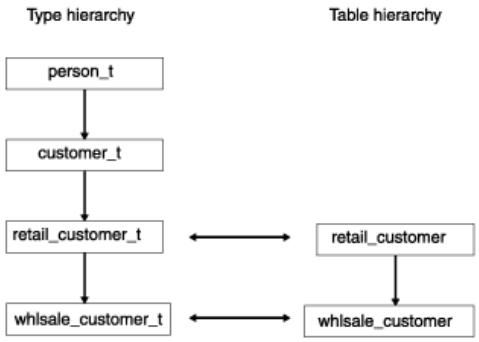


图 33: 只将某些类型指定给表的继承层次结构的示例

## 定义表层次结构

在可以创建表之前，用来定义表的类型必须已存在。同样，在定义相应的表层次结构之前需定义类型层次结构。要在表层次结构中的特定子表和超表之间建立关系，请使用 UNDER 关键字。下列 CREATE TABLE 语句定义图 *32: 类型层次结构与表层次结构之间的关系示例* 在第99页显示的简单表层次结构。本节中的示例假定 person_t、employee_t 和 sales_rep_t 类型已存在。

```
CREATE TABLE person OF TYPE person_t;

CREATE TABLE employee OF TYPE employee_t UNDER person;

CREATE TABLE sales_rep OF TYPE sales_rep_t UNDER employee;
```

person、employee 和 sales_rep 表分别是对 person_t、employee_t 和 sales_rep_t 类型定义的。因此，对于类型层次结构中的每个类型，在表层次结构中都存在相应的表。另外，表层次结构的表之间的关系必须与类型层次结构的类型之间的关系一致。例如：employee 表从 person 表继承的方式与 employee_t 类型从 person_t 类型继承的方式相同，并且，sales_rep 表从 employee 表继承的方式与 sales_rep_t 类型从 employee_t 类型继承的方式相同。

子表会自动继承超表的所有可继承属性。因此，可以随时添加或变更超表的属性，子表将自动继承这些更改。有关更多信息，请参阅在表层次结构中修改表行为 在第101页。

> 重要: 您必须对超表具有 UNDER 权限，才能创建继承该超表属性的子表。有关更多信息，请参阅类型表的 *Under* 权限 在第60页。

## 表层次结构中的表行为继承

当在超表下创建子表时，子表将继承其超表的所有属性，包括下列属性：

- 超表的所有列
- 约束定义
- 存储选项
- 索引
- 引用完整性
- 触发器
- 访问方法

此外，如果表 c 继承表 b，并且表 b 继承表 a，那么表 c 将自动继承表 b 独有的行为以及表 b 从表 a 继承的行为。因此，实际定义行为的超表与继承该行为的子表之间可以相距若干层。例如：请考虑以下表层次结构：

```
CREATE TABLE person OF TYPE person_t
(PRIMARY KEY (name))
FRAGMENT BY EXPRESSION
name < 'n' IN dbspace1,
name >= 'n' IN dbspace2;

CREATE TABLE employee OF TYPE employee_t
(CHECK(salary > 34000))
UNDER person;

CREATE TABLE sales_rep OF TYPE sales_rep_t
LOCK MODE ROW
UNDER employee;
```

在这个表层次结构中，employee 和 sales_rep 表继承 person 表的主键名和分段存储策略。sales_rep 表继承 employee 表的检查约束并添加了 LOCK MODE。下表显示了层次结构中每个表的行为。

person
: PRIMARY KEY 和 FRAGMENT BY EXPRESSION

employee
: PRIMARY KEY、FRAGMENT BY EXPRESSION 和 CHECK 约束

sales_rep
: PRIMARY KEY、FRAGMENT BY EXPRESSION、CHECK 约束和 LOCK MODE ROW

表层次结构也可包含这样的子表：在这些子表中，对子表定义的行为可以覆盖（以别的方式）从其超表继承的行为。考虑以下表层次结构，除了 employee 表添加了新的存储选项之外，这个表层次结构与上述示例完全相同：

```
CREATE TABLE person OF TYPE person_t
(PRIMARY KEY (name))
FRAGMENT BY EXPRESSION
name < 'n' IN person1,
name >= 'n' IN person2;

CREATE TABLE employee OF TYPE employee_t
(CHECK(salary > 34000))
FRAGMENT BY EXPRESSION
name < 'n' IN employ1,
name >= 'n' IN employ2
UNDER person;

CREATE TABLE sales_rep OF TYPE sales_rep_t
LOCK MODE ROW
```

```
UNDER employee;
```

同样，employee 和 sales_rep 表继承 person 表的主键名。然而，employee 表的分段存储策略覆盖 person 表的分段存储策略。因此，employee 和 sales_rep 表都在数据库空间 employ1 和 employ2 中存储数据，而 person 表在数据库空间 person1 和 person2 中存储数据。

## 在表层次结构中修改表行为

定义表层次结构之后，就不能修改现有表的结构（列）。但是，可以在层次结构中修改表的行为。表 4: 可以在表层次结构中进行修改的表行为 在第101页显示了可以在表层次结构中修改的表行为以及用于修改的语法。

表 4: 可以在表层次结构中进行修改的表行为

| 表行为 | 语法 | 注意事项 |
|---|---|---|
| 约束定义 | ALTER TABLE | 要添加或删除约束，请使用 ADD CONSTRAINT 或 DROP CONSTRAINT 子句。有关更多信息，请参阅表层次结构中对表的约束 在第101页。 |
| 索引 | CREATE INDEX, ALTER INDEX | 有关更多信息，请参阅在表层次结构中为表添加索引 在第101页以及《*SinoDB® SQL* 指南: 语法》中的 CREATE INDEX 和 ALTER INDEX 语句。 |
| 触发器 | CREATE/DROP TRIGGER | 不能删除继承的触发器。但是，可以从超表中删除触发器或将触发器添加至子表以覆盖继承的触发器。有关如何对超表和子表修改触发器的信息，请参阅表层次结构中表的触发器 在第101页。有关如何创建触发器的信息，请参阅《*SinoDB® SQL* 指南: 教程》。 |

当您修改层次结构中的超表时，任何现有的子表都将自动继承新的表行为。

重要: 当使用 ALTER TABLE 语句来修改表层次结构中的表时，您只能使用 ADD CONSTRAINT、DROP CONSTRAINT、MODIFY NEXT SIZE 和 LOCK MODE 子句。

### 表层次结构中对表的约束

只能在定义约束的表中变更或删除该约束。当约束是继承的，您不能从子表中删除或变更约束。但是，子表可添加附加的约束。从定义约束的表中继承的任何子表都将继承您对该表定义的任何附加约束。由于约束具有可加性，因此所有继承的约束以及当前（添加的）约束都适用。

### 在表层次结构中为表添加索引

当对层次结构中的超表定义索引时，您在该超表下面定义的任何子表也将继承该索引。假设一个表层次结构包含表 tab_a、tab_b 和 tab_c，其中 tab_a 是 tab_b 的超表，而 tab_b 是 tab_c 的超表。如果对 tab_b 的某个列创建索引，那么该索引在 tab_b 和 tab_c 中的该列上都存在。如果对 tab_a 的列创建索引，那么该索引的范围将包括 tab_a、tab_b 和 tab_c。

重要: 子表从超表中继承的索引不能被删除或修改。但是，可以在子表上添加索引。

索引、唯一约束和主键全都紧密相关。当指定唯一约束或主键时，数据库服务器会自动对该列创建唯一索引。因此，对超表定义的主键或唯一约束将应用于所有子表。例如：假设有两个表（一个超表和一个子表），这两个表都包含 emp_id 列。如果超表指定 emp_id 具有唯一约束，那么子表必须包含子表和超表中都唯一的 emp_id 值。

限制: 即使层次结构中的某些表不继承主键，也不能在表层次结构中定义多个主键。

### 表层次结构中表的触发器

不能删除继承的触发器。但是，可以对子表创建触发器以覆盖该子表从超表继承的触发器。与约束不同，触发器不具有可加性；只有位于层次结构中的超表上最接近的触发器才适用。

如果要禁用子表从其超表继承的触发器，可以对子表创建空触发器以覆盖来自超表的触发器。由于触发器不具有可加性，因此将对该子表以及该子表下的任何子表执行这个空触发器，那些子表不会进行进一步的覆盖。

## 表层次结构中的 SERIAL 类型

表层次结构可以包含具有 SERIAL 和 BIGSERIAL 或 SERIAL8 类型的列。但是，在表层次结构中只允许一个 SERIAL 列和一个 BIGSERIAL 列或一个 SERIAL8 列。假设您创建以下类型和表层次结构：

```
CREATE ROW TYPE parent_t (a INT);
CREATE ROW TYPE child1_t (s_col SERIAL) UNDER parent_t;
CREATE ROW TYPE child2_t (s8_col SERIAL8) UNDER child1_t;
CREATE ROW TYPE child3_t (d FLOAT) UNDER child2_t;

CREATE TABLE parent_tab of type parent_t;
CREATE TABLE child1_tab of type child1_t UNDER parent_tab;
CREATE TABLE child2_tab of type child2_t UNDER child1_tab;
CREATE TABLE child3_tab of type child3_t UNDER child2_tab;
```

parent_tab 表不包含 SERIAL 类型。child1_tab 将 SERIAL 计数器引入到层次结构中。child2_tab 从 child1_tab 继承 SERIAL 列并添加了 SERIAL8 列。child3_tab 继承了 SERIAL 和 SERIAL8 列。

对于层次结构中的任何表，插入到 s_col 或 s8_col 列中的 0 值将单调递增，而不管此插入操作由哪个表执行。

不能在 CREATE ROW TYPE 语句中设置 SERIAL 或 SERIAL8 类型的起始值。要设置表层次结构中的 SERIAL 或 SERIAL8 列的起始值，可使用 ALTER TABLE 语句。以下语句显示如何修改表以修改在表层次结构中任一位置插入的下一个 SERIAL 和 SERIAL8 值：

```
ALTER TABLE child3_tab
MODIFY (s_col SERIAL(100), s8_col SERIAL8 (200))
```

除上述描述的行为以外，所有适用于无类型表中的 SERIAL、BIGSERIAL 和 SERIAL8 类型列的规则也都适用于表层次结构中的 SERIAL、BIGSERIAL 和 SERIAL8 类型列。有关更多信息，请参阅选择数据类型 在第29页和《*SinoDB® SQL* 指南*:* 参考》。

## 将新表添加到表层次结构

定义表层次结构之后，不能使用 ALTER TABLE 语句对层次结构中的表添加、删除或修改列。但是，倘若新的子类型和子表不会干扰现有的继承关系，那么可将新的子类型和子表添加到现有层次结构中。下图说明了一种可将类型和相应的表添加至现有层次结构的方法。虚线表示添加的子类型和子表。

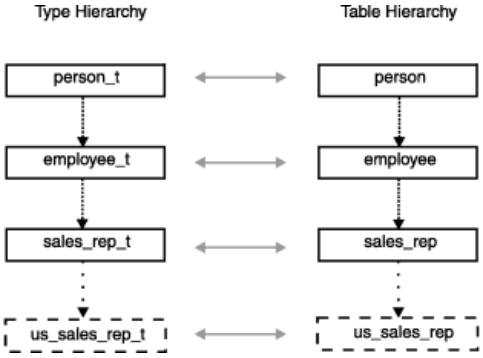


图 34: 如何将子类型和子表添加至现有继承层次结构的示例

下列语句说明如何将类型和表添加至图 *34:* 如何将子类型和子表添加至现有继承层次结构的示例 在第102页显示的继承层次结构：

```
CREATE ROW TYPE us_sales_rep_t (domestic_sales DECIMAL(15,2))
UNDER employee_t;
```

```
CREATE TABLE us_sales_rep OF TYPE us_sales_rep_t
UNDER sales_rep;
```

也可以添加从现有超类型及其并行超表分支出来的子类型和子表。下图显示了如何将 customer_t 类型和 customer 表添加至现有层次结构。在此示例中，customer 表和 employee 表都从 person 表继承属性。

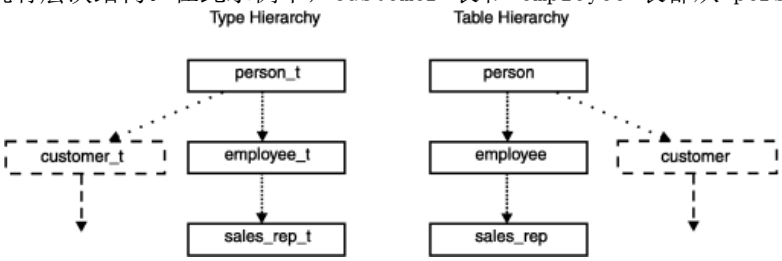


图 35: 在现有超类型和超表下添加类型和表的示例

下列语句分别在 person_t 类型和 person 表下创建 customer_t 类型和 customer 表：

```
CREATE ROW TYPE customer_t (cust_num INTEGER) UNDER person_t;

CREATE TABLE customer OF TYPE customer_t UNDER person;
```

## 在表层次结构中删除表

如果某个表及其相应命名行类型不具有相关性（它们不是超表和超类型），那么可删除该表及其类型。必须先删除表，然后才能删除类型。关于删除表的信息，请参阅《*SinoDB® SQL* 指南*:* 语法》中的 DROP TABLE 语句。有关如何删除命名行类型的信息，请参阅删除命名行类型 在第94页。

## 在表层次结构中变更表的结构

不能使用 ALTER TABLE 语句来对表层次结构中的表添加、删除或修改列。可以使用 ALTER TABLE 语句来添加、删除或修改约束。

对表层次结构中的表添加、删除或修改列（或变更表的结构）的过程是一项相当耗时的任务。

要在表层次结构中变更表的结构：

1. 从所有要修改的子表和超表中下载数据。
2. 删除子表和子类型。
3. 修改卸载的数据文件。
4. 修改超表。
5. 重新创建子类型和子表。
6. 上传数据。

## 在表层次结构中查询表

表层次结构允许您在单个 SQL 命令中构造 SELECT、UPDATE 或 DELETE 语句，这些语句的作用域是超表及其子表。例如：对表层次结构中的任何超表执行的查询将返回该超表的所有列的数据以及子表从该超表继承的列的数据。要将查询结果限制为只返回表层次结构中的一个表，必须在查询中包含 ONLY 关键字。有关如何查询和修改表层次结构中表的数据的更多信息，请参阅《*SinoDB® SQL* 指南*:* 教程》。

## 在表层次结构上创建表视图

可以创建基于表层次结构中任何表的视图。例如：以下语句对 person 表创建视图，该表是图 *32:* 类型层次结构与表层次结构之间的关系示例 在第99页显示的表层次结构的根超表：

```
CREATE VIEW name_view AS SELECT name FROM person
```

由于 person 表是一个超表，因此视图 name_view 显示 person、employee 和 sales_rep 表的 name 列中的数据。要创建只显示 person 表中数据的视图，请使用 ONLY 关键字，如以下示例所示：

```
CREATE VIEW name_view AS SELECT name FROM ONLY(person)
```

限制： 不能对定义在超表上的视图执行插入或更新操作，因为数据库服务器无法知道表层次结构中新行的放置位置。

有关如何创建带类型视图的信息，请参阅类型视图 在第69页。

# 创建和使用用户定义的强制转型

本章描述用户定义的强制转型并阐述如何使用运行时强制转型对扩展数据类型执行数据转换。

## 什么是强制转型？

强制转型是将值从一种数据类型转换为另一种数据类型的机制。强制转型允许在具有不同数据类型的值之间作比较或用一种数据类型的值替代另一数据类型的值。SinoDB® 在下列类型的表达式中支持强制转型：

- 列表达式
- 常量表达式
- 函数表达式
- SPL 变量
- 主变量（ESQL）
- 语句局部变量 (SLV) 表达式

要将一种数据类型的值转换为另一数据类型，那么数据库或数据库服务器必须具有强制转型。SinoDB® 支持下列类型的强制转型：

内置强制转型

内置强制转型是构建到数据库服务器中的强制转型。内置强制转型执行不同内置数据类型之间的自动转换。

用户定义的强制转型

用户定义的强制转型通常需要强制转型函数来处理从一种数据类型到另一数据类型的转换。要注册和使用用户定义的强制转型，必须使用 CREATE CAST 语句。

如果使用 CREATE CAST 语句创建强制转型时包括了 EXPLICIT 关键字，那么用户定义的强制转型是显式的。（缺省选项是显式的。）永远不会自动调用显式强制转型。要调用显式强制转型，必须使用 CAST... AS 关键字或双冒号 (::) 强制转型运算符。

如果使用 CREATE CAST 语句创建强制转型时包括了 IMPLICIT 关键字，那么用户定义的强制转型是隐式的。在运行时，数据库服务器会自动调用隐式强制转型来执行数据转换。

所有强制转型都包括在 syscasts 系统目录表中。有关 syscasts 的更多信息，请参阅《*SinoDB® SQL* 指南*:* 参考》。

### 用户定义的强制转型

当数据库服务器没有提供内置强制转型来执行两种数据类型之间的转换时，您可以创建用户定义的强制转型来处理数据类型转换。通常，用户定义的强制转型用于下列扩展数据类型的转换：

不透明数据类型

不透明数据类型的开发者必须定义强制转型来处理不透明数据类型的内部/外部表示法之间的转换。有关如何为不透明数据类型创建和注册强制转型的信息，请参阅《*SinoDB®* 用户自定义例程和数据类型开发者指南》。

单值数据类型

不能直接将单值数据类型与其源类型作比较。但是，SinoDB® 会自动注册从单值类型到源类型（反之亦然）的显式强制转型。单值类型不会继承对其源类型定义的强制转型。另外，对单值类型定义的用户定义强制转型不可用于其源类型。有关阐述如何对单值类型创建和使用强制转型的更多信息及示例，请参阅为用户定义的强制转型创建强制转型函数 在第112页。

命名行类型

在大多数情况下，可以将命名行类型显式强制转型为另一行类型值，而无需创建强制转型。然而，要在命名行类型的值与其他一些数据类型的值之间进行转换，必须首先创建强制转型来处理该转换。

有关如何创建和使用用户定义的强制转型的示例，请参阅单值数据类型之间强制转型的示例 在第113页。有关 CREATE CAST 语句的语法，请参阅《*SinoDB® SQL* 指南: 语法》。

### 调用强制转型

对于内置强制转型和用户定义的隐式强制转型，数据库服务器会自动（隐式地）调用强制转型来处理数据转换。例如：可以将 INT 类型的值与 SMALLINT、FLOAT 或 CHAR 值作比较，而无需显式地对表达式进行强制转型，因为数据库服务器提供了系统定义的强制转型来透明地处理这些内置数据类型之间的转换。

当定义用户定义的显式强制转型来处理两种数据类型之间的转换时，必须使用 CAST...AS 关键字或双冒号强制转型运算符 (::) 来显式调用强制转型。下列不完整的示例显示了两种调用显式强制转型的方法：

```
... WHERE new_col = CAST(old_col AS newtype)

... WHERE new_col = old_col::newtype
```

### 对用户定义的强制转型的限制

不能在两种内置数据类型之间创建用户定义的强制转型。也不能创建包括下列任何数据类型的用户定义的强制转型：

- 集合数据类型：LIST、MULTISET 或 SET
- 未命名行类型
- 智能大对象数据类型：CLOB 或 BLOB
- 简单大对象数据类型：TEXT 或 BYTE

通常，两种数据类型之间的强制转型要求每种数据类型都表示相同数目的组件值。例如：如果行类型中的每个字段在不透明数据类型中都有对应的字段，则可以在行类型与不透明数据类型之间进行强制转型。当您想要在具有相同存储结构的两种数据类型之间执行转换时，可使用不带强制转型函数的 CREATE CAST 语句。否则，必须创建强制转型函数，然后使用 CREATE CAST 语句进行注册。有关如何使用强制转型函数来创建用户定义的强制转型的示例，请参阅为用户定义的强制转型创建强制转型函数 在第112页。

## 强制转型行类型

只有当两种行类型具有相同数目的字段并且下列其中一个条件也成立时，才可以在任何两种行类型（命名行类型或未命名行类型）的值之间进行比较或替代：

- 两种行类型的所有对应字段都具有相同的数据类型。

  当两种行类型具有相同数目的字段并且对应字段的数据类型相同时，将它们视为在结构上相同。
- 存在用户自定义强制转型，以便可在比较两种命名行类型时执行转型。
- 存在系统定义的强制转型或用户定义的强制转型，以对不具有相同数据类型的对应字段值执行必需的转换。

  当对应的字段不具有相同的数据类型时，可使用系统定义的强制转型或用户定义的强制转型来对这些字段处理数据转换。

如果存在用于对个别字段处理数据转换的内置强制转型，则可将一种行类型的值显式强制转型为其他行类型（除非行类型都是未命名行类型，在这种情况下，显式强制转型不是必需的）。

如果不存在用于处理字段转换的内置强制转型，则可创建用户定义的强制转型来处理字段转换。该强制转型可以是隐式的，也可以是显式的。

通常，当将行类型强制转型为另一行类型时，可使用显式或隐式强制转型来处理个别字段转换。因为数据库服务器不对已进行显式强制转型的值应用任何附加的隐式强制转型，所以当对应字段之间的转换要求进行显式强制转型时，进行强制转型的字段值必须与对应字段值完全匹配。

## 在命名与未命名行类型之间强制转型

要将命名行类型的值与未命名行类型的值作比较，可使用显式强制转型。假设创建下列命名行类型和表：

```
CREATE ROW TYPE info_t (x CHAR(1), y CHAR(20))
CREATE TABLE customer (cust_info info_t)
CREATE TABLE retailer (ret_info ROW (a CHAR(1), b CHAR(20)))
```

下列 INSERT 语句显示如何创建 customer 和 retailer 表的行类型值：

```
INSERT INTO customer VALUES(ROW('t','philips')::info_t)
INSERT INTO retailer VALUES(ROW('f','johns'))
```

要将 customer 表中的数据与 retailer 表中的数据进行比较或替换，必须使用显式强制转型来将一种行类型的值转换为另一行类型。在以下查询中，将 ret_info 列（未命名行类型）显式强制转型为 info_t（命名行类型）：

```
SELECT cust_info
FROM customer, retailer
WHERE cust_info = ret_info::info_t
```

通常，要在命名行类型与未命名行类型之间执行转换，必须将一种行类型显式强制转型为另一行类型。可以在两个方向上执行显式强制转型：可以将命名行类型强制转型为未命名行类型，也可以将未命名行类型强制转型为命名行类型。以下语句返回与上述示例相同的结果。但是，本示例将命名行类型显式强制转型为未命名行类型：

```
SELECT cust_info
FROM customer, retailer
WHERE cust_info::ROW(a CHAR(1), b CHAR(20)) = ret_info
```

## 在未命名行类型之间强制转型

可以在不使用显式强制转型的情况下将结构上相同的两种未命名行类型进行比较。也可以将一种未命名行类型与另一种未命名行类型作比较，前提是这两种行类型具有相同的字段数，并且存在用于转换不具有相同数据类型的对应字段值的强制转型。换言之，如果所有用于处理字段转换的强制转型都是系统定义的或者都是隐式强制转型，则从一种未命名行类型到另一种未命名行类型的强制转型是隐式的。否则，必须对未命名行类型进行显式强制转型才能将其与另一种行类型作比较。

假设创建以下 prices 表：

```
CREATE TABLE prices
(col1  ROW(a SMALLINT, b FLOAT)
 col2  ROW(x INT, y REAL) )
```

当用于在对应字段之间执行转换的内置强制转型存在时，可以将两种未命名行类型的值进行比较（无需使用显式强制转型）。因此，以下查询不需要显式强制转型就可以将 col1 与 col2 值进行比较：

```
SELECT * FROM prices WHERE col1 = col2
```

在本示例中，数据库服务器隐式调用内置强制转型来将 SMALLINT 的字段值转换为 INT 并且将 REAL 的字段值转换为 FLOAT。

如果两种行类型的对应字段不能隐式地进行相互强制转型，则可在这两种行类型之间进行显式强制转型（前提是存在用于处理这两种类型之间的转换的用户定义强制转型）。

## 在命名行类型之间强制转型

命名行类型是强类型化的，这表示即使两种命名行类型在结构上相同，数据库服务器也会将这两种行类型识别成两种独立的类型。因此，在对两种命名行类型进行比较之前，必须创建并注册用户定义的强制转型。有关如何创建和使用强制转型来处理两种命名行类型之间的转换的示例，请参阅在命名行类型之间强制转型的示例 在第112页。

## 对字段进行显式强制转型

在两种行类型（命名行类型或未命名行类型）（其字段包含不同数据类型）之间进行显式强制转型之前，必须先存在一种强制转型（由系统定义或用户定义）来处理对应字段数据类型之间的转换。

当在两种行类型之间进行显式强制转型时，数据库服务器会自动调用显式强制转型来处理字段数据类型之间的转换。换言之，当对行类型值执行显式强制转型时，除非进行多个级别的强制转型来处理字段的数据类型转换，否则不必对行类型的个别字段进行显式强制转型。

在本节中，使用以下示例中的行类型和表来阐述对命名行和未命名行类型进行的显式强制转型的行为：

```
CREATE DISTINCT TYPE d_float AS FLOAT;
CREATE ROW TYPE row_t (a INT, b d_float);

CREATE TABLE tab1 (col1 ROW (a INT, b d_float));
CREATE TABLE tab2 (col2 ROW (a INT, b FLOAT));
CREATE TABLE tab3 (col3 row_t);
```

### 对未命名行类型的字段进行显式强制转型

如果两种行类型之间的转换涉及在特定字段值之间进行转换的显式强制转型，那么您可以对行类型值进行显式强制转型，但不需要单独对某个字段进行显式强制转型。

以下语句显示如何将值插入到 tab1 表中：

```
INSERT INTO tab1 VALUES (ROW( 3, 5.66::FLOAT::d_float))
```

要将 tab1 的 col1 中的值插入到 tab2 的 col2 中，由于数据库服务器不会自动处理 tab1 的 d_float 单值类型与 tab2 表的 FLOAT 类型之间的转换，因此必须对行值进行显式强制转型：

```
INSERT INTO tab2 SELECT col1::ROW(a INT, b FLOAT) FROM tab1
```

在本示例中，由于从 d_float 到 FLOAT 的转换要求进行显式强制转型（将单值类型转换为其源类型要求进行显式强制转型），因此用于转换 b 字段的强制转型是显式的。

通常，要在两个未命名行类型（其中一个或多个字段使用显式强制转型）之间进行强制转型，则必须在行类型级别而不是字段级别进行显式强制转型。

### 对命名行类型的字段进行显式强制转型

当将值显式强制转型为命名行类型时，数据库服务器会自动调用将字段值转换为目标数据类型的隐式或显式强制转型。在以下语句中，col1 到类型 row_t 的显式强制转型将自动调用将 FLOAT 类型的字段值转换为 d_float 的显式强制转型：

```
INSERT INTO tab3 SELECT col2::row_t FROM tab2
```

以下 INSERT 语句包括对 row_t 类型的显式强制转型。对行类型的显式强制转型也会调用显式强制转型来将 row_t 类型的 b 字段从 FLOAT 转换为 d_float。通常，对行类型的显式强制转型也会对行类型包含的个别字段（深度为一层）调用任何显式强制转型以处理转换。

```
INSERT INTO tab3 VALUES (ROW(5, 6.55::FLOAT)::row_t)
```

以下语句也有效，并且与上述语句返回相同的结果。但是，此语句显示了将 row_t 值插入到 tab3 表中执行的所有显式强制转型。

```
INSERT INTO tab3 VALUES (ROW(5, 6.55::float::d_float)::row_t)
```

在上述示例中，行类型的 b 字段之间的转换要求进行两个级别的强制转型。数据库服务器将任何包含小数点的值作为 DECIMAL 类型来处理。另外，DECIMAL 与 d_float 数据类型之间不存在隐式强制转型，因此需要进行两个级别的强制转型：从 DECIMAL 到 FLOAT 的强制转型以及从 FLOAT 到 d_float 的第二个强制转型。

### 对行类型的个别字段进行强制转型

如果对行类型的某个字段执行的操作要求进行显式强制转型，则可对个别字段值进行显式强制转型，而不必考虑与该字段相关联的行类型。以下语句使用对字段值执行的显式强制转型来处理转换：

```
SELECT col1 from tab1, tab2 WHERE col1.b = col2.b::FLOAT::d_float
```

如果对某个行类型的某个字段执行的操作要求进行隐式强制转型，那么可以指定正确的字段值，数据库服务器将自动处理转换。在以下语句中（此语句会将不同数据类型的字段值作比较），内置强制转型自动在 INT 值与 FLOAT 值之间进行转换：

```
SELECT col1 from tab1, tab2 WHERE col1.a = col2.b
```

## 强制转型集合数据类型

在某些情况下，可使用显式强制转型在具有不同元素类型的两个集合之间执行转换。要比较或替换任何两种集合类型的值，两个集合都必须具有 SET、MULTISET 或 LIST 类型。

- 当所有组件类型相同时，两种元素类型是相同的。例如：如果一个集合的元素类型是行类型，则另一集合类型也是行类型，并且具有相同的字段数和字段数据类型。
- 数据库中存在用于在元素类型（不具有相同数据类型）的所有组件之间执行转换的强制转型。

  如果对应的元素类型不具有相同的数据类型，则 SinoDB® 可使用内置强制转型或用户定义的强制转型来对这些元素类型处理数据转换。

当数据库服务器对集合数据类型的值进行插入、更新或比较时，会在元素数据类型级别进行类型检查。因此，在两种集合类型之间的强制转型中，因为存储在集合中的实际数据具有特定的元素类型，所以数据转换在元素类型的级别发生。

在本节的集合强制转型示例中使用下列类型和表：

```
CREATE DISTINCT TYPE my_int AS INT;

CREATE TABLE set_tab1 (col1 SET(my_int NOT NULL));
CREATE TABLE set_tab2 (col2 SET(INT NOT NULL));
CREATE TABLE set_tab3 (col3 SET(FLOAT NOT NULL));
CREATE TABLE list_tab (col4 LIST(INT NOT NULL));
CREATE TABLE m_set_tab(col5 MULTISET(INT NOT NULL));
```

## 对集合类型转换的限制

由于每种集合数据类型（SET、MULTISET 和 LIST）具有不同的特征，因此不允许在不同集合类型的集合之间进行转换。例如：存储在 LIST 集合中的元素有与其相关联的特定顺序。如果插入 LIST 集合中的元素插入 MULTISET 集合中，那么此顺序将丢失。因此，即使两个集合可能共享同一元素类型，也不能使用一个集合类型的集合中的元素来对另一集合插入或更新元素。以下 INSERT 语句将返回错误，因为执行插入操作的列是 MULTISET 集合，而插入的值是 LIST 集合：

```
INSERT INTO m_set_tab SELECT col4 FROM list_tab -- returns error
```

## 具有不同元素类型的集合

如何处理具有相同集合类型但不同元素类型的两个集合之间的转换取决于每个集合的元素类型以及当元素类型不相同时数据库服务器用于将一种元素类型转换为另一元素类型的强制转型的类型，如下所示：

- 如果存在内置强制转型或隐式用户定义的强制转型来处理两种元素类型之间的转换，那么不需要在集合类型之间进行显式强制转型。
- 如果存在显式强制转型来处理元素类型之间的转换，则可对一个集合执行显式强制转型。

### 在元素类型之间进行隐式强制转型

当数据库中存在隐式强制转型来处理两个集合的不同元素类型之间的转换时，不需要使用显式强制转型即可对元素执行从一个集合到另一集合的插入或更新操作。以下 INSERT 语句从 set_tab2 表中检索元素并将元素插入到 set_tab3 表中。虽然 set_tab2 中的集合列具有 INT 元素类型，而 set_tab3 中的集合列具有 FLOAT 元素类型，但一个内置强制转型会隐式地处理 INT 值与 FLOAT 值之间的转换。在此情况下，没有必要进行显式强制转型。

```
INSERT INTO set_tab3 SELECT col2 FROM set_tab2
```

### 在元素类型之间进行显式强制转型

当使用显式强制转型来执行两个集合的不同元素类型之间的转换时，必须将一个集合显式强制转型为另一个集合类型。在以下示例中，元素类型（INT 与 my_int）之间的转换要求进行显式强制转型。（单值类型与其源类型之间的强制转型总是显式的。）

以下 INSERT 语句从 set_tab2 表中检索元素并将元素插入到 set_tab1 表中。set_tab2 中的集合列具有 INT 元素类型，set_tab1 中的集合列具有 my_int 元素类型。因为元素类型（INT 与 my_int）之间的转换要求进行显式强制转型，所以必须显式地对集合类型进行强制转型。

```
INSERT INTO set_tab1 SELECT col2::SET(my_int NOT NULL)
   FROM set_tab2
```

要对集合类型执行显式强制转型，必须包括构造函数（SET、MULTISET 或 LIST）、元素类型和 NOT NULL 关键字。

## 将关系数据转换为 MULTISET 集合

当数据来自关系表时，可使用集合子查询来将行值强制转型为 MULTISET 集合。假设您创建了下列表：

```
CREATE TABLE tab_a ( a_col INTEGER);
CREATE TABLE tab_b (ms_col MULTISET(ROW(a INT) NOT NULL) );
```

以下示例显示如何使用集合子查询来将 tab_a 表中的 INT 值行转换为 MULTISET 集合。将把 tab_a 中的所有行都转换为 MULTISET 集合并插入到 tab_b 表中。

```
INSERT INTO tab_b VALUES (
   (MULTISET (SELECT a_col FROM tab_a)))
```

# 强制转型单值数据类型

单值类型不继承内置类型（该类型可以由某个单值类型用作其源类型）的任何内置强制转型。因此，将内置数据类型隐式转换为其他数据类型的现有内置强制转型不适用于将该内置类型用作其源类型的单值类型。但是，当创建基于内置类型的单值类型时，数据库服务器提供了两个显式强制转型来处理从单值类型到内置类型以及从内置类型到单值类型的转换。

## 对单值类型进行显式强制转型

要在单值类型与其源类型的值之间进行比较或替换，必须将一种类型显式强制转型为另一类型。例如：要使用源类型的值来对单值类型的列进行插入或更新，必须将值显式强制转型为单值类型。

假设创建基于 INTEGER 数据类型的单值类型 int_type 以及带有类型为 int_type 的列的表，如下所示：

```
CREATE DISTINCT TYPE int_type AS INTEGER;
CREATE TABLE tab_z(col1 int_type);
```

要将值插入到 tab_z 表中，必须将 col1 列的值显式强制转型为 int_type，如下所示：

```
INSERT INTO tab_z VALUES (35::int_type)
```

假设创建基于 NUMERIC 数据类型的单值类型 num_type 以及带有类型为 num_type 的列的表，如下所示：

```
CREATE DISTINCT TYPE num_type AS NUMERIC;
CREATE TABLE tab_x (col1 num_type);
```

单值 num_type 不继承对 NUMERIC 数据类型存在的系统定义的强制转型。因此，以下插入操作要求进行两个级别的强制转型。第一个强制转型将值 35 从 INT 转换为 NUMERIC，第二个强制转型从 NUMERIC 转换为 num_type：

```
INSERT INTO tab_x VALUES (35::NUMERIC::num_type)
```

因为不存在用于直接从 INT 类型转换为 num_type 的强制转型，所以对 tab_x 表执行的以下 INSERT 语句将返回错误：

```
INSERT INTO tab_x VALUES (70::num_type) -- returns error
```

## 在单值类型与其源类型之间强制转型

虽然单值类型的数据与其源类型具有相同的表示法，但单值类型不能直接与其源类型进行比较。因此，创建单值数据类型时，SinoDB® 会自动注册下列显式强制转型：

- 从单值类型到其源类型的强制转型
- 从源类型到单值类型的强制转型

假设创建两个单值类型：一个用于处理电影字幕，另一个用于处理音乐录音。可以创建下列基于 VARCHAR 数据类型的单值类型：

```
CREATE DISTINCT TYPE movie_type AS VARCHAR(30);
CREATE DISTINCT TYPE music_type AS VARCHAR(30);
```

然后可以创建 entertainment 表，包含类型为 movie_type、music_type 和 VARCHAR 的列。

```
CREATE TABLE entertainment
(
video         movie_type,
compact_disc  music_type,
laser_disv    VARCHAR(30)
```

```
);
```

要将单值类型与其源类型作比较（反之亦然），必须执行从一种数据类型到另一数据类型的显式强制转型。例如：假设您想要查找在录影带和激光影碟上都可以播放的电影。以下语句要求在 WHERE 子句中指定显式强制转型，以将具有单值类型 (music_type) 的值与具有其源类型 (VARCHAR) 的值作比较。在本示例中，将源类型显式强制转型为单值类型。

```
SELECT video FROM entertainment
   WHERE video = laser_disc::movie_type
```

但是，也可以将单值类型显式强制转型为源类型，如以下语句所示：

```
SELECT video FROM entertainment
   WHERE video::VARCHAR(30) = laser_disc
```

要在对同一源类型定义的两个单值类型之间执行转换，必须先执行中间强制转型以转换回源类型，然后再强制转型为目标单值类型。以下语句将 music_type 的值与 movie_type 的值作比较：

```
SELECT video FROM entertainment
   WHERE video = compact_disc::VARCHAR(30)::movie_type
```

### 添加和删除单值类型的强制转型

为了对单值类型强制执行强类型化，数据库服务器提供了显式强制转型来处理单值类型与其源类型之间的转换。但是，单值类型的创建者可以删除现有的显式强制转型并创建隐式强制转型，以使单值类型与其源类型之间的转换不需要进行显式强制转型。

> 重要： 当删除单值类型与其源类型之间由数据库服务器提供的显式强制转型，然后改为创建隐式强制转型来处理这些数据类型之间的转换时，就无法充分发挥单值类型的独特性。

以下 DROP CAST 语句删除两个自动对 movie_type 定义的显式强制转型：

```
DROP CAST(movie_type AS VARCHAR(30));
DROP CAST(VARCHAR(30) AS movie_type);
```

在删除现有强制转型之后，可以创建两个隐式强制转型来处理 movie_type 与 VARCHAR 之间的转换。下列 CREATE CAST 语句用于创建两个隐式强制转型：

```
CREATE IMPLICIT CAST (movie_type AS VARCHAR(30));
CREATE IMPLICIT CAST (VARCHAR(30) AS movie_type);
```

如果数据库中已存在用于在两种数据类型之间进行转换的强制转型，那么不能创建这样的强制转型。

如果创建隐式强制转型以在单值类型与其源类型之间进行转换，那么可以在不进行显式强制转型的情况下对这两种类型作比较。在以下语句中，video 列与 laser_disc 列之间的比较要求进行转换。由于已创建隐式强制转型，因此 VARCHAR 与 movie_type 之间的转换是隐式的。

```
SELECT video FROM entertainment
   WHERE video = laser_disc
```

## 强制转型为智能大对象

数据库服务器提供了强制转型以允许将 TEXT 和 BYTE 对象转换为 BLOB 和 CLOB 数据类型。此功能允许用户将旧数据库中的 BYTE 和 TEXT 数据迁移到 BLOB 和 CLOB 列中。

以下示例显示如何使用显式强制转型来将 stores_demo 数据库的 catalog 表中的 BYTE 列值转换为 BLOB 列值，并更新 superstores_demo 数据库中的 catalog 表：

```
UPDATE catalog SET advert = ROW (
(SELECT cat_photo::BLOB FROM stores_demo:catalog
   WHERE catalog_num = 10027),
   advert.caption)
      WHERE catalog_num = 10027
```

数据库服务器不会提供将 BLOB 转换为 BYTE 值或 CLOB 转换为 TEXT 值的强制转型。

## 为用户定义的强制转型创建强制转型函数

如果数据库包含不透明数据类型、单值数据类型或命名行类型，那么您可能想创建允许在不同数据类型之间进行转换的用户定义的强制转型。当您想要在具有相同存储结构的两种数据类型之间执行转换时，可使用不带强制转型函数的 CREATE CAST 语句。然而，在某些情况下，您必须创建强制转型函数，然后注册为强制转型。在以下情况下，必须创建强制转型函数：

- 转换操作是在具有不同存储结构的两种数据类型之间进行的
- 转换操作涉及对值进行处理以确保数据转换有意义

下列各部分阐述如何创建和使用需要强制转型函数的用户定义的强制转型。

### 在命名行类型之间强制转型的示例

假设创建以下示例所示的命名行类型和表。尽管命名行类型在结构上是相同的，但是 writer_t 和 editor_t 均是唯一的数据类型。

```
CREATE ROW TYPE writer_t (name VARCHAR(30), depart CHAR(3));
CREATE ROW TYPE editor_t (name VARCHAR(30), depart CHAR(3));

CREATE TABLE projects
(
   book_title  VARCHAR(20),
   writer      writer_t,
   editor      editor_t
);
```

要处理两种命名行类型之间的转换，必须首先创建用户定义的强制转型。以下示例创建一个强制转型函数并将其注册为强制转型，以处理从类型 writer_t 到 editor_t 的转换：

```
CREATE FUNCTION cast_rt (w writer_t)
   RETURNS editor_t
   RETURN (ROW(w.name, w.depart)::editor_t);
END FUNCTION;

CREATE CAST (writer_t as editor_t WITH cast_rt);
```

创建并注册强制转型之后，可以将类型为 writer_t 的值显式强制转型为 editor_t 类型。以下查询在 WHERE 子句中使用显式强制转型将类型为 writer_t 的值转换为 editor_t 类型：

```
SELECT book_title FROM projects
   WHERE CAST(writer AS editor_t) = editor;
```

如果您愿意的话，可使用 :: 强制转型运算符来执行同一强制转型，如以下示例所示：

```
SELECT book_title FROM projects
   WHERE writer::editor_t = editor;
```

## 单值数据类型之间强制转型的示例

假设您想要定义单值类型来表示 dollar、yen 和 sterling 货币。两种货币之间的任何比较都必须考虑汇率。因此，您必须创建的强制转型函数不仅需要处理从一种数据类型到另一种数据类型的强制转型，还需要计算要比较的值的汇率。

以下示例显示如何对同一源类型 DOUBLE PRECISION 定义三种单值类型：

```
CREATE DISTINCT TYPE dollar AS DOUBLE PRECISION;
CREATE DISTINCT TYPE yen AS DOUBLE PRECISION;
CREATE DISTINCT TYPE sterling AS DOUBLE PRECISION;
```

定义单值类型后，您可以创建一个表，这个表提供制造商对可比较产品开出的价格。以下示例创建 manufact_price 表，该表中 dollar、yen 和 sterling 单值类型各占一列：

```
CREATE TABLE manufact_price
(
product_desc  VARCHAR(20),
us_price      dollar,
japan_price   yen,
uk_price      sterling
);
```

将值插入 manufact_price 表时，可以强制转型为 dollar、yen 和 sterling 值的正确单值类型，如下所示：

```
INSERT INTO manufact_price
   VALUES ('baseball', 5.00::DOUBLE PRECISION::dollar,
   510.00::DOUBLE PRECISION::yen,
   3.50::DOUBLE PRECISION::sterling);
```

因为单值类型不继承任何可用于其源类型的内置强制转型，所以上述每个 INSERT 语句都需要进行两次强制转型。对于每个 INSERT 语句，内部强制转型从 DECIMAL 转换为 DOUBLE PRECISION，外部强制转型从 DOUBLE PRECISION 转换为正确单值类型（dollar、yen 或 sterling）。

在可以对 dollar、yen 和 sterling 数据类型进行比较之前，必须创建强制转型函数并将其注册为强制转型。以下示例创建可用于比较 dollar、yen 和 sterling 值的 SPL 函数。每个函数都将输入值乘以一个反映汇率的值。

```
CREATE FUNCTION dollar_to_yen(d dollar)
   RETURN (d::DOUBLE PRECISION * 106)::CHAR(20)::yen;
END FUNCTION;

CREATE FUNCTION sterling_to_dollar(s sterling)
   RETURNS dollar
   RETURN (s::DOUBLE PRECISION * 1.59)::CHAR(20)::dollar;
END FUNCTION;
```

在编写强制转型函数之后，必须使用 CREATE CAST 语句来将函数注册为强制转型。下列语句将 dollar_to_yen() 和 sterling_to_dollar() 函数注册为显式强制转型：

```
CREATE CAST(dollar AS yen WITH dollar_to_yen);
CREATE CAST(sterling AS dollar WITH sterling_to_dollar);
```

在函数注册为强制转型之后，可将其用于需要转型的数据类型之间的转型。有关用来创建强制转型函数并将其注册为强制转型的语法，请参阅《*SinoDB® SQL* 指南*:* 语法》中的 CREATE FUNCTION 和 CREATE CAST 语句。

在以下查询中，WHERE 子句包含一个显式强制转型，该强制转型调用 dollar_to_yen() 函数来比较 dollar 与 yen 值：

```
SELECT * FROM manufact_price
   WHERE CAST(us_price AS yen) < japan_price;
```

以下查询使用强制转型运算符来执行上述查询中显示的同一转换：

```
SELECT * FROM manufact_price
   WHERE us_price::yen < japan_price;
```

您也可以使用显式强制转型来转换查询返回值。以下查询使用强制转型来返回与 dollar 值相等的 yen 值。该查询的 WHERE 子句也使用显式强制转型来将 dollar 与 yen 值进行比较。

```
SELECT us_price::yen, japan_price FROM manufact_price
   WHERE us_price::yen < japan_price;
```

## 多级别强制转型

多级别强制转型指的是在表达式中进行两个或更多级别的强制转型，以将一种数据类型的值转换为目标数据类型。由于 yen 与 sterling 值之间不存在强制转型，因此将两种数据类型作比较的查询要求进行多次强制转型。第一次（内部）强制转型将 sterling 值转换为 dollar 值；第二次（外部）强制转型将 dollar 值转换为 yen 值。

```
SELECT * FROM manufact_price
   WHERE japan_price < uk_price::dollar::yen
```

您可以添加另一个强制转型函数来直接处理 yen 到 sterling 的转换。以下示例创建函数 yen_to_sterling() 并将其注册为强制转型。为了考虑汇率，此函数将 yen 值乘以 .01 以派生出等价的 sterling 值。

```
CREATE FUNCTION yen_to_sterling(y yen)
   RETURNS sterling
   RETURN (y::DOUBLE PRECISION * .01)::CHAR(20)::sterling;
END FUNCTION;

CREATE CAST (yen AS sterling WITH yen_to_sterling);
```

添加了 yen 到 sterling 的强制转型之后，可使用单级别强制转型来比较 yen 和 sterling 值，如以下查询所示：

```
SELECT japan_price::sterling, uk_price FROM manufact_price
   WHERE japan_price::sterling) < uk_price;
```

在 SELECT 语句中，显式强制转型返回 yen 值作为其 sterling 等价值。在 WHERE 子句中，强制转型允许比较 yen 和 sterling 值。