GBASE®

GBase 8s SQL 指南：教程

www.gbase.cn

GBase 8s SQL 指南：教程，南大通用数据技术股份有限公司

版权声明

免责声明

通讯方式

南大通用数据技术股份有限公司

天津市高新区开华道22号普天创新产业园东塔20-23层

电话：400-013-9696　　邮箱：info@gbase.cn

商标声明

目　录

1 简介

1.1 本简介内容

本出版物显示如何使用基本和高级结构化查询语言（SQL）访问和处理数据库中的数据。它讨论数据操纵语言（DML）语句以及 DML 语句经常使用的触发器和存储过程语言（SPL）例程。

本出版物针对以下用户编写：

- 数据库用户
- 数据库管理员
- 数据库应用程序程序员

本手册假定您具备以下背景：

- 对于计算机、操作系统和操作系统提供的实用程序的工作知识
- 使用关系数据库经验或熟悉数据库概念
- 一些计算机编程经验

本出版物是讨论 SQL 的 GBase 8s 实现的一系列出版物中的一本。GBase 8s SQL 指南：语法包含 SQL 和 SPL 的所有语法描述。 GBase 8s SQL 参考指南提供对 SQL 的各个方面（语言语句除外）的参考信息。

1.2 符合行业标准

GBase 8s 产品符合各种标准。

基于 GBase 8s SQL 的产品完全兼容 SQL-92 入门级（发布为 ANSI X3.135-1992），这与 ISO 9075:1992 完全相同。另外， GBase 8s 数据库服务器的许多功能都遵守 SQL-92 中级和完全级别以及 X/Open SQL 公共应用程序环境 (CAE) 标准。

1.3 演示数据库

DB-Access 实用程序随 GBase 8s 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。GBase 8s 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定义的例程的示例。

有关如何创建和填充演示数据库的信息，请参阅《GBase 8s DB-Access 用户指南》。有关数据库及其内容的描述，请参阅《GBase 8s SQL 参考指南》。

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

1.4 示例代码约定

SQL 代码的示例在整个出版物中出现。除非另有说明，代码不特定于任何单个的 GBase 8s 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将添加更多的代码，但是不必显示它以描述正在讨论的概念。

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。

2 数据库概念

本章描述基本数据库概念并着重讨论一下主题：

- 数据模型

- 多用户
- 数据库术语
- SQL（结构化查询语言）

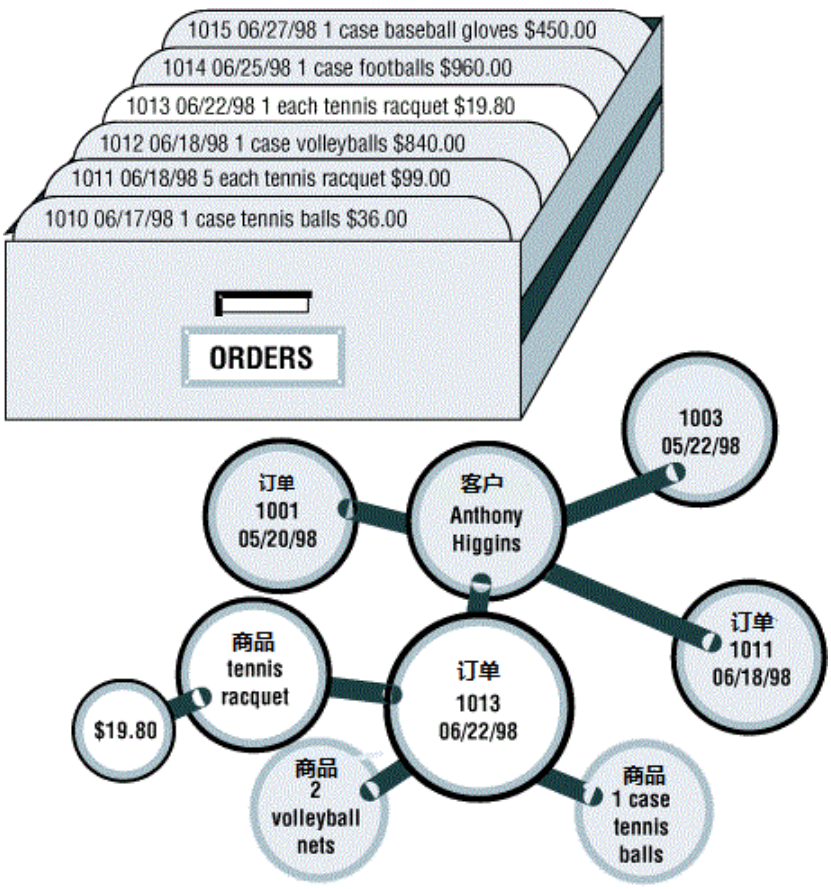
数据库的实际使用从 SELECT 语句开始，在编写 SELECT 语句中进行了描述。

2.1 数据模型的说明

在数据库中收集的信息与在文件中收集的信息的主要差异是组织数据的方式。平面文件是以物理方式组织的；一些项置于其他项的前面或后面。但数据库的内容时根据数据模型组织的。数据模型是一个方案或一个图，它定义数据单元并指定每个单元如何与其它单元相关联。

例如，某个数字可以出现在文件或数据库中。在文件中，它指示出现在文件中某个位置的数字。但是，数据库中的数字具有数据模型指定给它的角色。该角色可能是一个价格，与作为客户预订订单中某一商品销售的产品相关联。价格、产品、商品、订单和客户等组件中的每一个也具有数据模型指定的角色。有关数据模型的说明，请参阅下图。

图: 使用数据模型的优点



当创建数据库时就会设计数据模型。然后根据模型布局的规划插入数据单元。有些书籍使用术语模式而不是数据模型。

2.1.1 存储数据

数据库和文件之间的另一个差异是数据库的组织方式是与数据库一起存储的。

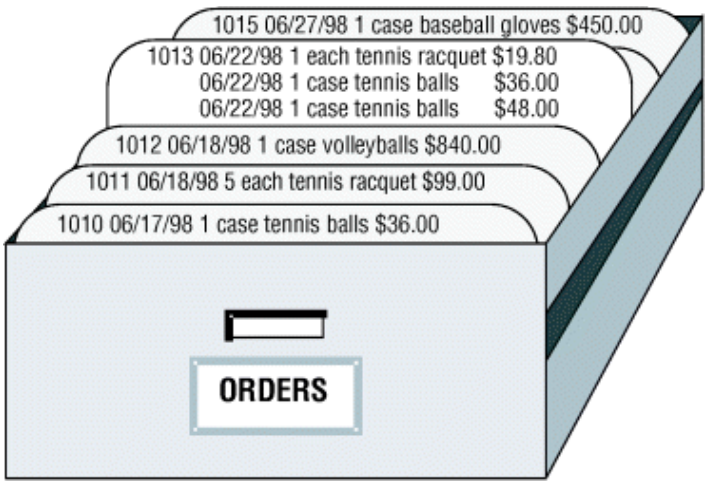
文件可能具有复杂的内部结构，但该结构的定义不在该文件中；结构的定义在创建或使用文件的程序中。例如：字处理程序存储的文档文件可能包含描述文档格式的详细结构。但是，只有字处理程序能够译解该文件的内容，因为结构是在程序而不是文件中定义的。

然而，数据模型包含在它描述的数据库中。它与数据库融为一体。并且可用于使用该数据库的任何程序。模型不但定义数据项的名称，而且定义数据项的数据类型，因此程序可以使它自己适应该数据库。例如：某个程序可发现在当前数据库中，价格项是八位数的十进制数，小数点右边有两位数；于是它可为该类型的数分配存储器。在 SQL 编程和通过 SQL 程序修改数据中讨论了程序如何使用数据库的主题。

2.1.2 查询数据

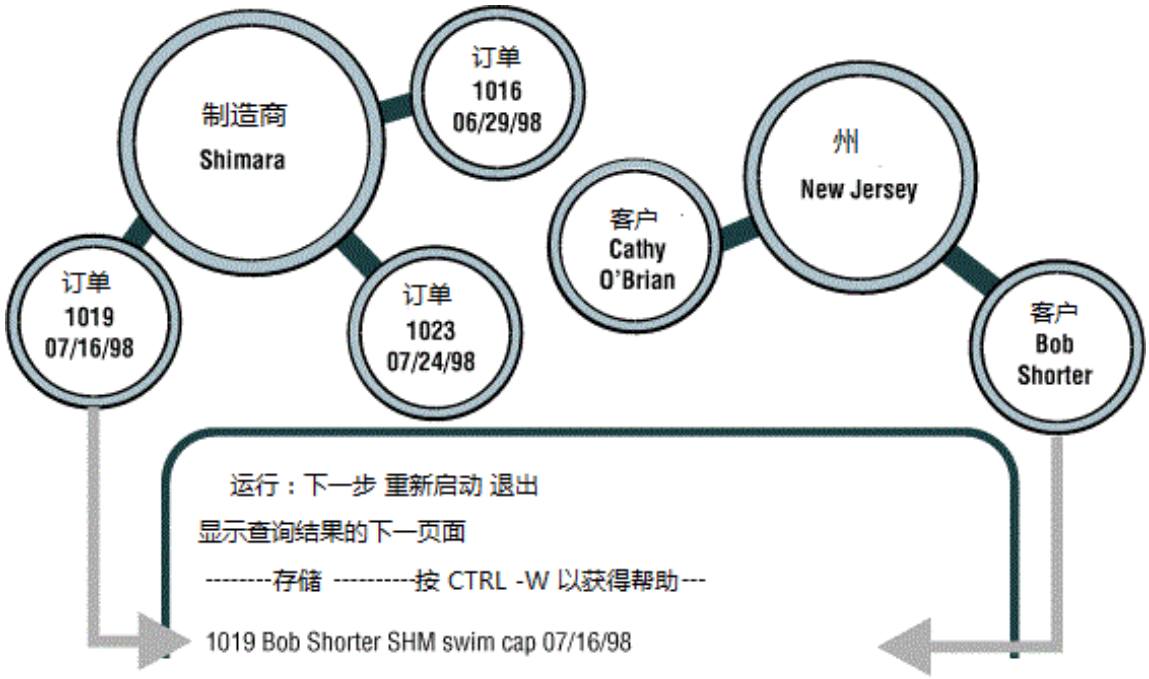
数据库与文件的另一个差异是访问它们的方法。您可以按顺序搜索文件。在每一行或每一个记录的特定物理位置查找特定值。即，可询问“哪个记录在第一个字段中具有数字 1013？”，下图显示了此搜索类型。

图: 按顺序搜索文件



相反，当查询数据库时，使用模型定义的条件，可以带着如下问题来查询数据库，“New Jersey 的客户 Shimara 公司生产的产品下了哪些装运日期在第三季度的订单？” 下图显示了此类查询。

图: 查询数据库



换言之，当范围存储在文件中的数据时，必须以该文件的物理布局的形式陈述您的问题。当查询数据库时，可以忽略计算机存储器的繁琐细节并以反映现实世界的方式（至少是数据模型反映现实世界的方式）陈述您的查询。

编写 SELECT 语句和编写高级 SELECT 语句讨论了用于进行查询的语言。

2.1.3 修改数据

数据模型还使修改数据库内容而少出错成为可能。可以使用如下语句查询数据库，“查询制造商为 Presta 或 Schraeder 的每个库存商品，并将其价格提高 13%”。以反映数据意义的形式说明更改。不必浪费时间与精力来考虑文件中记录内字段的详细信息，因此出错的几率也减小。

在修改数据中描述了用来修改存储数据的语句。

2.2 并发使用和安全性

数据库可以是许多用户的公共资源。多个用户可以同时查询和修改一个数据库。数据库服务器（管理所有数据库的内容的程序）确保查询和修改顺序完成而不会产生冲突。

数据库并发用户具有许多优点，但也引入了新的安全性和隐私问题。某些数据库是专用的，是个人建立以供自己使用的。其它一些数据库包含必须共享（但仅在受限组中共享）的机密材料；还有其它数据库提供公用访问权。

2.2.1 控制数据库使用

GBase 8s 数据库软件提供控制数据库使用的方法。当您设计数据库时，可以执行以下任何功能：

- 使数据库完全专用
- 对所有用户或选择的用户开放其全部内容
- 限制某些用户可以查看的数据选择（不同的数据选择适用于不同的用户组）
- 允许指定的用户查看特定项，但不能修改它们
- 允许指定的用户添加新数据，但不能修改旧数据
- 允许指定的用户更修改全部现有数据或现有数据的指定项
- 确保添加或修改的数据符合数据模型

访问管理策略

GBase 8s 支持两种访问管理系统：

基于标签的访问控制（LBAC）

基于标签的访问控制是强制访问控制的实现。它通常在存储高度敏感数据的存储库中，如由军队或保安服务公司保卫的系统。与 LBAC 相关的 GBase 8s 的主要文档是 GBase 8s 安全指南 。GBase 8s SQL 指南：语法 描述了数据库管理员（DBSA）如何创建和维护 LBAC 安全对象，此类管理员必须被授授予（DBSECADM）角色，并且该角色只能由数据库系统管理员（DBSA）授予。

自主访问控制（DAC）

自主访问控制是更为简单的系统，涉及的开销少于 LABC 。根据访问特权和角色，DAC 在所有的 GBase 8s 数据库中都启用，包括实现 LBAC 的数据库。

创建和授予角色

为了支持 DAC，数据库管理员（DBA）可以定义角色并将角色指定给用户。从而对需要访问相同数据库对象的用户组的访问特权进行标准化。当 DBA 将特权指定给角色时，一旦激活该角色，被授予该角色的每个用户都具有这些特权。为了激活特定角色，用户必须发出 SET ROLE 语句。用于定义和操纵角色的 SQL 语句包括：CREATE ROLE 、DROP ROLE 、GRANT 、REVOKE 和 SET ROLE。

有关定义和操作角色的 SQL 语法语句，请参阅《GBase 8s SQL 指南：语法》。

创建和授予角色：

1. 使用 CREATE ROLE 语句在当前数据库中创建新的角色。
2. 使用 GRANT 语句将访问特权授予该角色。
3. 使用 GRANT 语句将角色授予用户或 PUBLIC （所有用户）。
4. 用户必须发出 SET ROLE 语句来启用该角色。

为缺省角色定义和授予特权

DBA 还可以定义一个缺省角色以将该角色分配给特定数据库的单个用户或 PUBLIC 组。当用户与该数据库建立连接后，不需要用户发出 SET ROLE 语句，该角色将自动激活。在连接时，拥有缺省角色的每个用户都具有单独为该用户授予的访问特权以及缺省角色的特权。

对于给定用户，在给定时间内只有一个 CREATE ROLE 语句定义的角色可以生效。如果同时拥有缺省角色和一个或多个其它角色的用户使用 SET ROLE 语句使非缺省角色成为活动角色，那么仅授予缺省角色（不单独对用户，对 PUBLIC 或对新活动角色授予）的任何访问特权该用户将不再有效。同一用户可以发出 SET ROLE DEFAULT语句重新激活缺省角色，但该操作会禁用用户仅通过先前启用的非缺省角色而拥有的任何特权。

如果为用户和 PUBLIC 指定了不同的缺省角色，那么用户的缺省角色优先。

为缺省角色定义和授予特权：

1. 使用 CREATE ROLE 语句在当前数据库中创建一个新角色。
2. 使用 GRANT 语句将特权授予该角色。
3. 使用以下语法将角色授予一个用户，并将该角色设置为缺省用户或 PUBLIC 角色：
   - GRANT DEFAULT ROLE rolename TO username;
   - GRANT DEFAULT ROLE rolename TO PUBLIC;
4. 使用 REVOKE DEFAULT ROLE 语句取消缺省角色到用户的关联。

   限制： 只有 DBA 或数据库所有者才能移除该缺省角色。
5. 使用 SET ROLE DEFAULT 语句将当前角色复位为缺省角色。

内置角色

出于安全原因，GBase 8s 支持内置角色，这些角色对于被授予该角色且连接到数据库的任何用户都生效，而与任何其他角色是否也处于活动状态无关。

例如，在 IFX_EXTEND_ROLE 配置参数设置为 ON 的数据库中，只有数据库服务器管理员（DBSA）或 DBSA 已授予内置 EXTEND 角色的用户才可以创建或删除使用 EXTERNAL 关键字定义的 UDR 。

同样，在实现 LBAC 安全策略的数据库中，DBSA 可以授予内置 DBSECADM 角色。该角色的被授予者成为数据库安全管理员，可以定义并实现 LBAC 安全策略并可以为数据和用户指定安全标签。

与用户定义的角色不同，内置角色无法被 DROP ROLE 语句删除。SET ROLE 语句对内置角色无效，因为在用户连接到它们已被授予内置角色的数据库期间，该角色始终处于活动状态。

有关定义和操纵角色的外部例程引用段或 SQL 语句的更多信息，请参阅《GBase 8s SQL 指南：语法》。

有关定义和操纵 LBAC 安全对象的 DBSECADM 角色或 SQL 语句的更多信息，请参阅《GBase 8s 安全指南》。

有关缺省角色的更多信息，请参阅《GBase 8s 管理员指南》。

2.2.2 集中管理

许多人使用的数据库都很有价值，必须将它们作为重要的企业资源来保护。当编译由价值数据的存储并同时允许许多职员访问这些数据时，会产生严重问题。通过在维护性能时保护数据来解决此问题，数据库服务器允许您将这些任务集中。

必须保护数据库，以免数据库丢失或遭到破坏。很多情况都可以对数据库构成威胁：软件和硬件故障以及火灾、水灾和其它自然灾害。丢失重要的数据库可能会带来巨大的破坏。破坏可能不仅包括重新创建丢失数据的支出和困难，而且包括数据库用户的生产时间损失以及用户不能工作时失去的业务和信誉。定期备份的计划可能帮助避免或减轻这些可能的灾难。

必须维护和调整许多人使用的大型数据库。必须有人监视系统资源的使用状况、列出其增长图表、预计瓶颈并计划数据库的扩展。用户将报告应用程序中的问题；必须有人诊断这些问题并更正它们。如果快速响应很重要，那么必须有人分析系统的性能并找出响应慢的原因。

2.3 重要的数据库术语

在开始下一章之前应该了解一些术语。根据您使用的数据库服务器，不同的术语组描述相应的数据库和数据模型。

2.3.1 关系数据库模型

使用 GBase 8s 数据库服务器厂家的数据库是对象关系数据库。在实际应用的术语中，这意味着所有数据库都以具有行和列的表的格式显示。其中具有以下简单的对应关系：

关系

描述

table = entity

一个表不是数据库对某一主题或一类事务已知的全部内容。

column = attribute

一列表示一个特征、特性或表主题适用的事实。

row = instance

一行表示表主题的个别实例。

对于如何选择实体和属性有一些规则，但仅当您设计数据库时这些规则才显得重要。已经设置现有数据库中的数据模型。要使用数据库，只需要知道表和列的名称以及表和列如何与现实对应。

2.3.2 表

数据库是分组为一个或多个表的信息集合。表是组织为行和列的数据项的数组。每个 GBase 8s 数据库服务器产品中都提供了演示数据库。下面是该演示数据库中的部分表。

stock_num	manu_code	description	unit_price	unit	unit_descr
. . .	. . .	. . .	. . .	. . .	. . .
1	HRO	baseball gloves	250.00	case	10 gloves/case
1	HSK	baseball gloves	800.00	case	10 gloves/case
1	SMT	baseball gloves	450.00	case	10 gloves/case
2	HRO	baseball	126.00	case	24/case
3	HSK	baseball bat	240.00	case	12/case
4	HSK	football	960.00	case	24/case

stock_num	manu_code	description	unit_price	unit	unit_descr
4	HRO	football	480.00	case	24/case
5	NRG	tennis racquet	28.00	each	each
. . .	. . .	. . .	. . .	. . .	. . .
313	ANZ	swim cap	60.00	case	12/box

表表示数据库管理员（DBA）想要存储的有关实体（数据库描述的某类型事物）的全部内容。示例表 stock 表示 DBA 想要存储的有关体育用品商店的全部内容。演示数据库中的其他表表示诸如 customer 和 orders 之类的实体。

2.3.3 列

表的每一列包含一个属性，它就是一个特性、特征或描述表的主题的事实。stock 表具有有关商品的下列事实的列：库存编号、制造商代码、描述、价格和计量单位。

2.3.4 行

表的每一行就是表主题的一个实例，就是实体的一个特定示例，stock 表的每一行表示体育用品商店销售的一种商品。

2.3.5 视图

视图是基于指定的 SELECT 语句的虚拟表。视图是数据库中内容的动态控制图片，它允许程序员确定用户查看额处理哪些信息。可为不同用户提供数据库内容的不同视图，还可使用若干种方法限制它们对那些内容的访问。

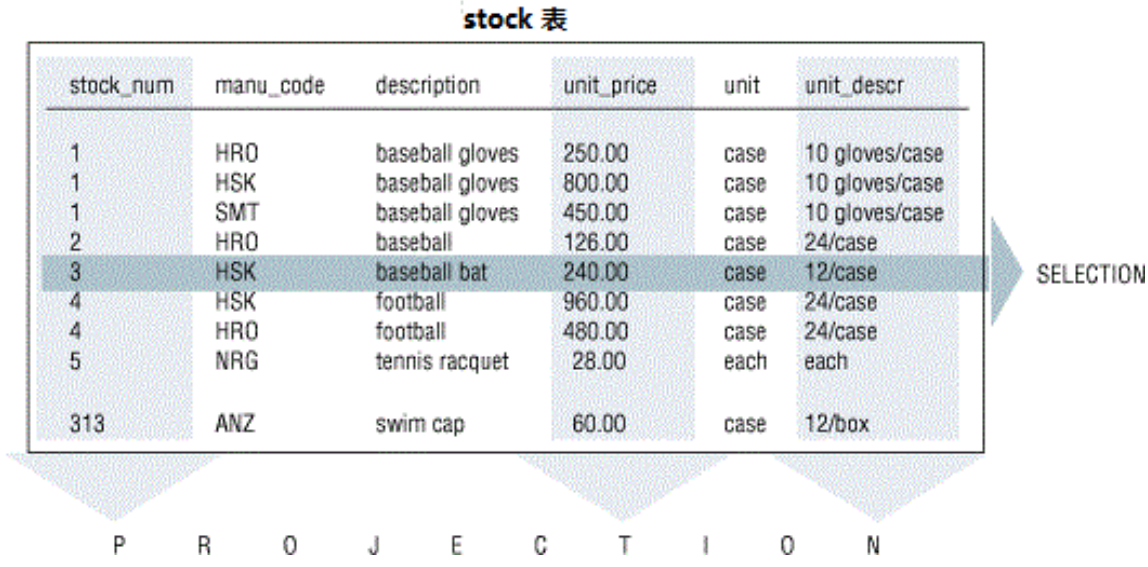
2.3.6 序列

序列是数据库对象，它生成已定义范围内的整个数字序列。数字序列可以按升序或降序运行，并且是单调的。有关序列的更多信息，请参阅《GBase 8s SQL 指南：语法》。

2.3.7 针对表的操作

因为数据库实际上是表的集合，使用数据库操作就是对表的操作。对象关系模型支持三种基本操作：选择、投影和连接。下图描述了选择和投影操作（在后面的章节中详细定义了这三种操作，并提供了许多示例。）

图: 选择和投影的说明



当从表选择数据时，您就选择了某些行而忽略其他行。例如：您可以通过要求数据库管理系统“选择制造商代码为 HSK 并且单价在 200.00 和 300.00 之间的所有行”来查询 stock 表。

当从表中进行投影时，您就选择了某些列而忽略其他列。例如：您可以通过要求数据库管理系统“投影 stock_num 、unit_descr 和 unit_price列”来查询 stock 表。

表只包含有关一个实体的信息；当想要有关多个实体的信息时，就必须连接它们的表。可使用多种方法来连接表。有关连接操作的更多信息，请参阅编写高级 SELECT 语句。

2.3.8 对象关系模型

GBase 8s 允许您构建对象关系数据库。除了支持字母数字数据（如字符串、整型、日期和小数）之外，对象关系数据库还扩展了关系模型的功能，使它具有以下面向对象的功能：

可扩展性

可以通过定义新数据类型（以及支持它们的访问方法和函数）和用户定义的例程（UDR、允许您存储和管理图像、音频、视频和大型文本文档等等）来扩展数据库服务器的功能。

GBase 及第三方供应商将某些数据类型和访问方法封装到 DataBlade 模块或共享类库中，可根据您的需要，将它们添加到数据库服务器中。DataBlade 模块使您能够存储费传统数据类型（如两维空间对象：线、多边形、椭圆和圆）并通过 R-tree 索引访问它

们。DataBlade 模块还可能提供对大型文本文档的新访问类型，包括词组匹配、模糊搜索和同义词匹配。

还可以使用允许您添加数据类型和访问方法的 GBase 8s 的功能来自己扩展数据库服务器。有关更多信息，请参阅《GBase 8s 用户定义的例程和数据类型开发者指南》 。

可以使用 SPL 和 C 编程语言来创建 UDR ，以便封装应用程序逻辑或增强 GBase 8s 的功能。有关更多信息，请参阅创建和使用 SPL 例程。

复杂类型

可以定义包含一个或多个现有数据类型的新数据类型。复杂类型在组织列和表级别的数据方面允许更大的灵活性。例如：使用复杂类型，您可以定义包含单个类型的值集合的列和包含多个组件类型的列。

继承

可以定义获取其它对象的属性的对象（类型和表）并添加特定于所定义对象的新属性。

GBase 8s 提供对象的面向对象的功能优于关系模型的功能，但以具有行和列的表的形式表示所有数据。虽然对象关系模型扩展了关系模型的功能，但您可以将数据模型作为传统关系数据库实现（如果您选择这样做的话）。

对于如何选择实体和属性的规则仅在您设计新的数据库时才显得非常重要。

2.4 结构化查询语言

大多数计算机软件还做不到以文字方式询问数据库“New Jersey 的客户下了哪些装运日期在第三季度的订单？”。您还必须使用软件能够容易分析的限制语法来表述问题。可以使用以下术语对演示数据库提出相同问题：

```
SELECT * FROM customer, orders
        WHERE customer.customer_num = orders.customer_num
        AND customer.state = 'NJ'
        AND orders.ship_date
        BETWEEN DATE('7/1/98') AND DATE('9/30/98');
```

此问题是“结构化查询语言（SQL）”的一个样本。您将使用这种语言来发出对数据库的所有操作。SQL 由语句组成，每个语句都以指定函数的一个或两个关键字开头。SQL 的 GBase 8s 实现包含从 ALLOCATE DESCRIPTOR 到 WHENEVER 的大量 SQL 语句。

仅当设置或调整数据库时才将使用大多数语句。通常将使用三个或四个语句来查询或更新数据库。有关 SQL 语句的详细信息，请参阅《GBase 8s SQL 指南：语法》 。

最经常使用的语句是 SELECT 语句。SELECT 是唯一可用来从数据库检索数据的语句。它还是最复杂的语句，本书中的后面两章将讨论它的许多用法。

2.4.1 标准 SQL

由于性能或竞争优势等原因，或者为了利用本地硬件或软件功能，每个 SQL 实现都与其它实现以及 GBase 版本的语言有些小的区别。为了确保这些差异不会增大，在二十世纪八十年代早期成立了标准委员会。

由美国国家标准学会（ANSI）资助的委员会 X3H2 在 1986 年发布了 SQL1 标准。此标准定义了一组核心的 SQL 功能和诸如 SELECT 等语句的语法。

2.4.2 GBase 8s SQL 和 ANSI SQL

SQL 的 GBase 8s 实现与标准 SQL 兼容。GBase 8s SQL 还与 GBase 版本的语言兼容。然而，GBase 8s SQL 包含对标准的扩展；即，某些语句的额外选项或功能及其他语句的更宽松的规则，大多数差异出现在不经常使用的语句中。例如：在 SELECT 语句方面很少出现差异，该语句占 SQL 使用率的 90%。

GBase 8s SQL 和 ANSI 标准之间的一个差异是，GBase 8s SQL 指南：语法会将 GBase 8s 语法标识为对 SQL 的 ANSI 标准的扩展。

2.4.3 交互式 SQL

要实施本书中的示例以尝试 SQL 和数据库设计，您需要一个允许您以交互方式执行 SQL 语句的程序。DB-Access 就是一个这样的程序。它帮助您编写 SQL 语句，然后将 SQL 语句传递至数据库服务器以供执行，并向您显示结果。

2.4.4 一般编程

可用编写合并 SQL 语句并与数据库服务器交换数据的程序。即，可以编写从数据库中检索数据并对选择的任何内容进行格式化的程序。还可以编写从任何格式的任何源中取出数据。准备数据并将数据插入到数据库中的程序。

还可以编写称为存储例程的程序来使用数据库数据和对象。编写的存储例程直接存储在数据库中的表中。然后，可以从 DB-Access 或 SQL 应用编程接口（API，例如 GBase 8s ESQL/C）执行存储例程。

SQL 编程和通过 SQL 程序修改数据提供了如何在程序中使用 SQL 的概述。

2.4.5 符合 ANSI 的数据库

当创建数据库时使用 MODE ANSI 关键字来将数据库指定为符合 ANSI 。在此类数据库中，ANSI/ISO 标准的某些特性适用。例如：修改数据的所有操作在事务中自动发生。这意味着更改要么作为一个整体进行，要么根本就不进行。在 GBase 8s SQL 指南：语法 中的语句描述中。在适当的位置说明了符合 ANSI 的数据库在行为方面的差异。

2.4.6 Global Language Support

GBase 8s 数据库服务器产品提供了 Global Language Support （GLS）功能部件。除了 U.S. ASCII 英语之外，GLS 允许您在其它语言环境中工作并在 SQL 数据和标识中使用非 ASCII 字符。可以使用 GLS 功能来与特定语言环境定制保持一致。语言环境文件包括特定于文化的信息。如货币和日期格式以及整理顺序。

2.5 总结

数据库包含一系列相关信息，但与存储数据的其他方法在基本方式上有所不同。数据库不仅包含数据，还包含数据模型，数据模型定义每个数据项并指定数据项相对于其他项和现实世界的意义。

多个用户可同时访问和修改数据库。每个用户具有数据库内容的不同视图，并且可使用若干方法限制每个用户对那些内容的访问。

关系数据库由表组成，而表由列和行组成。关系模型支持对表的三种基本操作：选择、投影和连接。

对象关系数据库扩展了关系数据库的功能。可定义新的数据类型来存储和管理音频、视频和大型文本文档等等。可以定义组合一个或多个现有数据类型的复杂类型，为在列和表中组织数据提供了更大的灵活性。可以定义继承其它数据库对象的属性的类型和表并添加特定于所定义对象的新属性。

要使用和查询数据库，使用 SQL 。ANSI 对 SQL 进行了标准化。您可用于提高性能的一些 GBase 8s 扩展补充了 ANSI 定义的语言。GBase 8s 工具还是得有可能与 ANSI 标准严格保持一致。

软件的两层结构将您的所有工作与数据库联系起来。底层总是执行 SQL 语句并管理磁盘和计算机内存中的数据的数据库服务器。上层是许多应用程序（有些来自 GBase ，有些由您、其他供应商或您的同事编写）之一。中间件是将数据库服务器与应用程序进行链接的组件，由数据库供应商提供来将客户机程序与数据库服务器绑定在一起。GBase 8s 存储过程语言（SPL）就是此类工具的一个示例。

3 编写 SELECT 语句

SELECT 语句是最重要且最复杂的 SQL 语句。可使用它和 SQL 语句 INSERT 、UPDATE 和 DELETE 操纵数据。可以使用 SELECT 语句从数据库检索数据。将它用作 INSERT 语句的一部分来生成新行或将它作为 UPDATE 语句的一部分来更新信息。

SELECT 语句是查询数据库中信息的主要方法。它是检索程序、报告、表单或电子表格中的数据的关键。可以将 SELECT 语句与查询工具 DB-Access 配合使用或在应用程序中嵌入 SELECT 语句。

本章介绍了使用 SELECT 语句查询和检索关系数据库数据的基本方法。本章讨论如何调整语句以从一个或多个表中选择信息行和列，如何在 SELECT 语句中包含表达式和函数以及如何创建数据库表之间的各种连接条件。SELECT 语句的语法和使用方法在 GBase 8s SQL 指南：语法中有详细描述。

本出版物中的大部分示例来自 stores_demo 数据库中的各表，该数据库随 GBase 8s SQL API 或数据库实用程序的软件提供。为了简便起见，示例只显示了每个 SELECT 语句检索的数据的一部分。有关演示数据库的结构和内容的信息，请参阅《GBase 8s SQL 参考指南》。为了着重强调，虽然 SQL 不区分大小写，但是在示例中用大写字母显示关键字。

3.1 介绍 SELECT 语句

SELECT 语句允许您查看关系数据库中的数据的子句构成。这些子句允许您从一个或多个表或视图中选择列和行、指定一个或多个条件、对数据进行排序和总结以及将选择的数据放置在临时表中。

本章介绍了如何使用五个 SELECT 语句子句。如果包含全部五个子句，那么它们必须按照下列顺序出现在 SELECT 语句中：

1. Projection 子句
2. FROM 子句
3. WHERE 子句
4. ORDER BY 子句
5. INTO TEMP 子句

只有 Projection 子句和 FROM 子句是必需的。这两个子句构成每个数据库查询的基础，原因是它们指定要检索的列值，以及包含这些列的表。使用以下列表中的一个或多个其它子句：

- 添加 WHERE 子句以选择特定行或创建连接条件。
- 添加 ORDER BY主键以更改生成数据的顺序。
- 添加 INTO TEMP 子句以将结果保存为表以供进一步查询。

还有两个 SELECT 语句子句 GROUP BY 和 HAVING，使您可以执行更复杂的数据检索。编写高级 SELECT 语句中对它们进行了描述。另一个子句 INTO 指定要从应用程序中的 SELECT 语句中接收数据的程序或主变量。关于使用 SELECT 语句的完整语法和规则在 GBase 8s SQL 指南：语法中有所描述。

3.1.1 SELECT 语句的输出

虽然在所有GBase 8s 产品中语法相同，但是结果输出的格式和显示取决于应用程序。本章和编写高级 SELECT 语句中的示例如同您在 DB-Access 中使用“交互式查询语言”选项时那样显示 SELECT 语句及输出。

大对象数据类型的输出

当发出包含大对象的 SELECT 语句时，DB-Access 按如下所示显示结果：

- 对于 TEXT 列或 CLOB 列，显示列的内容。
- 对于 BYTE 列，显示词 <BYTE value> 而不是实际值。
- 对于 BLOB 列，显示词 <SBlob data> 而不是实际值。

用户定义的数据类型的输出

DB-Access 使用特殊约定来显示包含复杂或不透明数据类型的列的输出。

非缺省代码集的输出

可以发出查询 NCHAR 列而不是 CHAR 列，或者 NVARCHAR 列而不是 VARCHAR 列的 SELECT 语句。

3.1.2 一些基本概念

SELECT 语句不同于 INSERT 、UPDATE 和 DELETE 语句，它不修改数据库中的数据。一次只能有一个用户修改数据，而多个用户可同时查询或选择数据。有关修改数据的语句的更多信息，请参阅修改数据。INSERT 、UPDATE 和 DELETE 语句的语法描述位于《GBase 8s SQL 指南：语法》中。

在关系数据库中，列是包含出现在表中的每一行中的特定信息类型的数据元素。行是在数据库表中的所有列上有关单个实体的信息的一组相关项。

可以从数据库表、系统目录表（包含有关数据库的信息的特殊表）、或视图（创建来包含一组定制数据的虚拟表）中选择列和行。有关系统目录表的信息在 GBase 8s SQL 参考指南中有所描述。

特权

在查询数据之前，确保您对数据库具有 Connect 特权和表的 Select 特权。通常这些特权授予所有用户。在 GBase 8s SQL 指南：语法的 GRANT 和 REVOKE 语句中描述了数据库访问权。

关系操作

关系操作涉及处理一个或多个表或者关系以产生另一个表，三种关系操作为选择、投影和连接。本章包括选择、投影和连接操作的一些示例。

选择和投影

在关系术语中，选择被定义为取得满足特定条件的单个表的行的水平子集。此类 SELECT 语句返回表中的某些行和所有列。选择是通过 SELECT 语句的 WHERE 子句实现的，如下图所示：

图: 查询

```
SELECT * FROM customer WHERE state = 'NJ';
```

该结果包含的列数与 customer 表相同，但只是后者的行的子集。在此示例中，DB-Access 显示单独行上来自每列的数据。

图: 查询结果

```
customer_num    119
        fname           Bob
        lname           Shorter
        company          The Triathletes Club
        address1        2405 Kings Highway
        address2
        city            Cherry Hill
        state           NJ
        zipcode         08002
        phone           609-663-6079

        customer_num    122
        fname           Cathy
        lname           O'Brian
        company          The Sporting Life
        address1        543d Nassau
```

```
        address2
        city            Princeton
        state           NJ
        zipcode         08540
        phone           609-342-0054
```

在关系术语中，投影被定义为从保留唯一行的单个表的列中获取垂直子集。此类SELECT 语句返回表中的某些行和某些列。

投影是通过 SELECT 语句的 Projection 子句中的投影列表实现的。如下图所示。

图: 查询

```
SELECT city, state, zipcode FROM customer;
```

此结果包含的列数与 customer 表相同，但只是投影表中列的子集。因为只从每行中选择一小部分的数据，所以 DB-Access 能够在一行上显示行的所有数据。

图: 查询结果

```
city            state zipcode

        Sunnyvale       CA      94086
        San Francisco   CA      94117
        Palo Alto       CA      94303
        Redwood City    CA      94026
        Los Altos       CA      94022
        Mountain View   CA      94063
        Palo Alto       CA      94304
        Redwood City    CA      94063
        Sunnyvale       CA      94086
        Redwood City    CA      94062
        Sunnyvale       CA      94085
        :
        Oakland         CA      94609
        Cherry Hill     NJ      08002
        Phoenix         AZ      85016
        Wilmington      DE      19898
        Princeton       NJ      08540
        Jacksonville    FL      32256
        Bartlesville    OK      74006
```

最常见的 SELECT 语句都同时使用选择和投影。此类查询返回表的某些行和某些列。如下图所示。

图: 查询

```
SELECT UNIQUE city, state, zipcode
        FROM customer
        WHERE state = 'NJ';
```

图 6包含 customer 表的列的子集和行的子集。

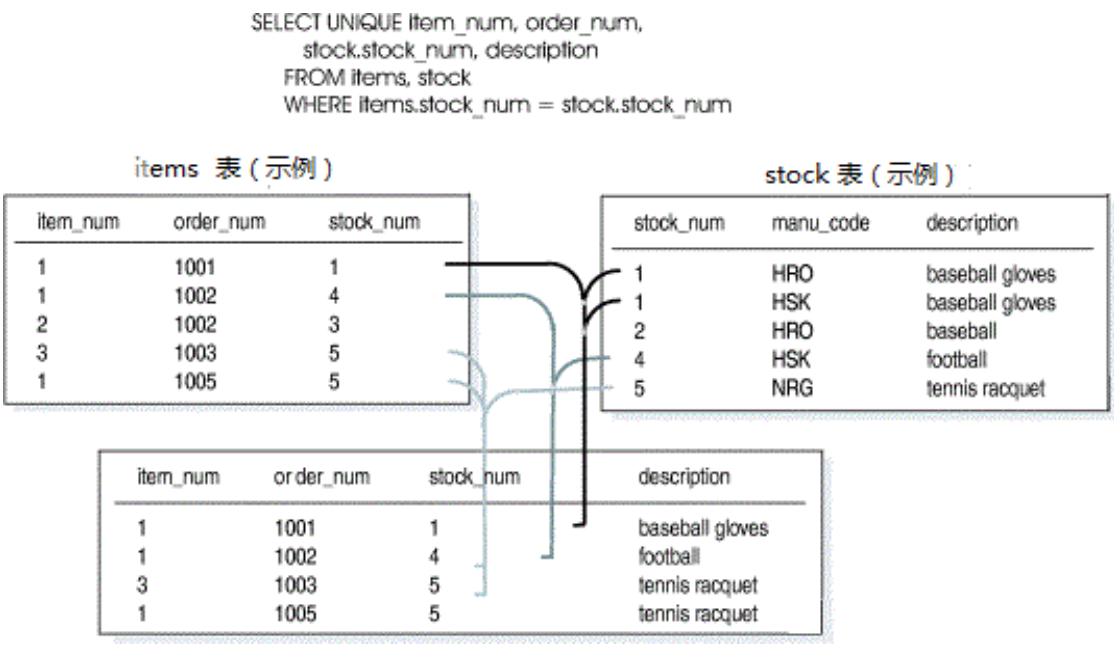
图: 查询结果

```
city              state zipcode

        Cherry Hill       NJ      08002
        Princeton         NJ      08540
```

连接

当两个或多个表被同一列或多个列连接时发生连接，它创建新的结果表。下图显示了一个查询，该查询使用 items 和 stock 表的子集来说明连接的概念。

图： 两个表之间的连接



下列查询将 customer 和 state 表连接起来。

图: 查询

```
SELECT UNIQUE city, state, zipcode, sname
        FROM customer, state
        WHERE customer.state = state.code;
```

该结果包含 customer 和 state 表的指定行和列。

图: 查询结果

```
city                 state zipcode sname


        Bartlesville       OK       74006     Oklahoma
        Blue Island         NY       60406     New York
        Brighton             MA       02135     Massachusetts
        Cherry Hill         NJ       08002     New Jersey
        Denver               CO       80219     Colorado
        Jacksonville       FL       32256     Florida
        Los Altos           CA       94022     California
        Menlo Park          CA       94025     California
        Mountain View     CA       94040     California
        Mountain View     CA       94063     California
        Oakland              CA       94609     California
        Palo Alto           CA       94303     California
        Palo Alto           CA       94304     California
        Phoenix              AZ       85008     Arizona
        Phoenix              AZ       85016     Arizona
        Princeton           NJ       08540     New Jersey
        Redwood City       CA       94026     California
        Redwood City       CA       94062     California
        Redwood City       CA       94063     California
        San Francisco     CA       94117     California
        Sunnyvale           CA       94085     California
        Sunnyvale           CA       94086     California
        Wilmington          DE       19898     Delaware
```

3.2 单个表的 SELECT 语句

可用多种方法查询数据库中的单个表。可用调整 SELECT 语句以执行以下操作：

- 检索所有或指定的列
- 检索所有或指定的行
- 对检索到的数据执行计算或其它功能
- 用各种方法对数据进行排序

最基本的 SELECT 语句只包含两个必需的子句，即 Projection 子句和 FROM。

3.2.1 使用星号（*）

下列查询在投影列表中指定 manufact 表中所有的列。显式投影列表是想要从表投影的列名或表达式的列表。

图: 查询

```
SELECT manu_code, manu_name, lead_time FROM manufact;
```

以下查询使用通配符星号（*）作为选择列表中的简写来表示表中所有名称相同的列。当想要所有列按其定义的顺序排列时，可使用星号（*），隐式选择列表使用星号。

图: 查询

```
SELECT * FROM manufact;
```

因为 manufact 表只含有三列，图 1和图 2是等价的，并且显示相同的结果。即，manufact 表中每个列和行的列表。下图显示了结果。

图: 查询结果

```
manu_code manu_name      lead_time

   SMT       Smith            3
   ANZ       Anza             5
   NRG       Norge            7
   HSK       Husky            5
   HRO       Hero             4
   SHM       Shimara         30
   KAR       Karsten         21
   NKL       Nikolus          8
   PRC       ProCycle         9
```

队列进行重新排序

下列查询显示了如何通过更改列在投影列表中的顺序。

图: 查询

```
SELECT manu_name, manu_code, lead_time FROM manufact;
```

该查询结果包含与前一查询结果相同的列，但因为用不同的顺序指定了列，所以显示也不同。

图: 查询结果

```
manu_name       manu_code lead_time

        Smith           SMT          3
        Anza            ANZ          5
        Norge           NRG          7
        Husky           HSK          5
        Hero            HRO          4
        Shimara         SHM         30
        Karsten         KAR         21
        Nikolus         NKL          8
        ProCycle        PRC          9
```

3.2.2 使用 ORDER BY 子句存储行

不以任何特定顺序排列查询的结果。例如：图 4和图 2以随机顺序显示。

可以将 ORDER BY 子句添加到您的 SELECT 语句里指导系统以特定顺序对数据进行排序。ORDER BY 子句是任何远程或本地表或视图中的列名的列表。投影列表中允许的所有表达式在 ORDER BY 列表中也允许。如果在 ORDER BY 列表中使用的列具有选择触发器，那么将不会激活该触发器。

以下查询返回 manufact 表中 manu_code 、manu_name 和 lead_time 列的每一行。并根据 lead_time 进行排序。

图: 查询

```
SELECT manu_code, manu_name, lead_time
        FROM manufact
        ORDER BY lead_time;
```

对于 GBase 8s，不需要在投影列表中包括您想要在 ORDER BY 子句中使用的列。即，可以根据不在投影列表中检索的列对数据进行排序。以下查询返回 manufact 表中 manu_code 、manu_name 列的每一行，并根据 lead_time 进行排序。lead_time 列位于 ORDER BY 子句中（尽管未包含在投影列表中）。

图: 查询

```
SELECT manu_code, manu_name
        FROM manufact
```

```
ORDER BY lead_time;
```

升序

缺省情况下，检索到的数据按升序顺序排序。在 ASCII 字符集中，升序是从大写字母 A 到小写字母 z，对于字符数据类型，那么是从最小值到最大值。DATE 和 DATETIME 数据按照从最早到最新排序，INTERVAL 数据按从时间范围最短到最长排序。

降序

降序与升序相反，对于字符类型为从小写 z 到大写 A，对于数字类型为从最大值到最小值。DATE 和 DATETIME 数据按照从最新到最早排序，INTERVAL 数据按从时间范围最长到最短排序。以下查询显示了降序的示例。

图: 查询

```
SELECT * FROM manufact ORDER BY lead_time DESC;
```

列名后跟关键字 DESC 导致以降序对检索数据进行排序，如下图所示：

图: 查询结果

```
manu_code manu_name      lead_time

SHM       Shimara        30
KAR       Karsten        21
PRC       ProCycle       9
NKL       Nikolus        8
NRG       Norge          7
HSK       Husky          5
ANZ       Anza           5
HRO       Hero           4
SMT       Smith          3
```

您可以在 ORDER BY 子句中指定任何内置数据类型的列（TEXT 、BYTE 、BLOB 或 CLOB 除外），数据库服务器根据该列中的值对数据进行排序。

对多个列进行排序

还可以使用 ORDER BY 排序两个或多个列，这会创建嵌套排序。缺省值仍然是升序。在 ORDER BY 子句中最先列出的列优先。

下列查询和图 2及相应的查询结果显示了嵌套排序。要修改显示所选数据的顺序。更改在 ORDER BY 子句中命名的两个列的顺序。

```
SELECT stock_num, manu_code, description, unit_price
        FROM stock
        ORDER BY manu_code, unit_price;
```

查询结果中，manu_code 列数据按字母顺序显示，并且在同一 manu_code（例如：ANZ 、HRO）中，unit_price 以升序列出。

图: 查询结果

```
stock_num manu_code description      unit_price

        5 ANZ         tennis racquet       $19.80
        9 ANZ         volleyball net       $20.00
        6 ANZ         tennis ball          $48.00
        313 ANZ         swim cap             $60.00
        201 ANZ         golf shoes           $75.00
        310 ANZ         kick board           $84.00
        :
        111 SHM         10-spd, assmbld      $499.99
        112 SHM         12-spd, assmbld      $549.00
        113 SHM         18-spd, assmbld      $685.90
        5 SMT         tennis racquet       $25.00
        6 SMT         tennis ball          $36.00
        1 SMT         baseball gloves      $450.00
```

下列查询显示了 ORDER BY 子句中列的相反顺序。

图: 查询

```
SELECT stock_num, manu_code, description, unit_price
        FROM stock
        ORDER BY unit_price, manu_code;
```

在此查询结果中，数据按 unit_price 的升序显示，其中两个或多个行具有相同的 unit_price（例如：$20.00 、$48.00 、$312.00），manu_code 以字母顺序显示。

图: 查询结果

```
stock_num manu_code description      unit_price

        302 HRO         ice pack             $4.50
        302 KAR         ice pack             $5.00
        5 ANZ         tennis racquet       $19.80
        9 ANZ         volleyball net       $20.00
```

```
103 PRC          frnt derailleur       $20.00
⋮
108 SHM          crankset              $45.00
6 ANZ            tennis ball           $48.00
305 HRO          first-aid kit         $48.00
303 PRC          socks                 $48.00
311 SHM          water gloves          $48.00
⋮
113 SHM          18-spd, assmbld       $685.90
1 HSK            baseball gloves       $800.00
8 ANZ            volleyball            $840.00
4 HSK            football              $960.00
```

ORDER BY 子句中的列的顺序十分重要，DESC 关键字的位置也很重要。尽管下列查询中的各语句在 ORDER BY 子句中包含相同的语句，但是每个语句产生的结果并不相同（没有显示）。

图: 查询

```
SELECT * FROM stock ORDER BY manu_code, unit_price DESC;

        SELECT * FROM stock ORDER BY unit_price, manu_code DESC;

        SELECT * FROM stock ORDER BY manu_code DESC, unit_price;

        SELECT * FROM stock ORDER BY unit_price DESC, manu_code;
```

3.2.3 选择特定列

之前的章节显示了如何选择和排序表中所有的数据。然而，您经常希望看到的是一个或多个特定列的数据。并且，公式是使用 Projection 和 FROM 子句指定列和表，并可以使用 ORDER BY 子句按照升序或降序对数据进行排序。

如果想要操作 orders 表中的所有客户号，那么使用以下查询中的语句。

图: 查询

```
SELECT customer_num FROM orders;
```

该结果显示了语句如何只选择 orders 表中 customer_num 列中的所有数据，并列出所有订单上的客户号，包括重复的客户号。

图: 查询结果

```
customer_num

        104
        101
        104
        ⋮
        122
        123
        124
        126
        127
```

输出包括若干重复，原因是某些客户下了多个订单。有时您想要在投影中看到重复的行。而有时您却只想看到特异值，而不是每个值都出的频率。

要抑制重复行，可在选择列表的开头包括关键字 DISTINCT 或其同义词 UNIQUE，每个查询级别一次，如以下查询所示。

图: 查询

```
SELECT DISTINCT customer_num FROM orders;

        SELECT UNIQUE customer_num FROM orders;
```

要生成更可读的表，图 3将显示限制为仅显示一次 orders 表中的每个客户号。如下所示。

图: 查询结果

```
customer_num

        101
        104
        106
        110
        111
        112
        115
        116
        117
        119
        120
        121
```

```
122
123
124
126
127
```

假设您正在处理客户电话，并且想要找到购买订单号 DM354331。要列出 orders 表中的所有购买订单号，使用诸如以下查询所示的语句。

图: 查询

```
SELECT po_num FROM orders;
```

该结果显示了如何检索到 orders 表中 po_num 列的数据。

图: 查询结果

```
po_num

B77836
9270
B77890
8006
2865
Q13557
278693
⋮
```

然而，该列表顺序无用。可以添加 ORDER BY 子句来以升序对列数据进行排序，使得查找特定 po_num 更容易，如下所示。

图: 查询

```
SELECT po_num FROM orders ORDER BY po_num;
```

图: 查询结果

```
po_num

278693
278701
2865
429Q
4745
8006
8052
```

```
        9270
        B77836
        B77890
        ⋮
```

要从表中选择多个列，请在 Projection 子句的投影列表中列出它们。以下查询显示了选择列的顺序就是检索列的顺序，从左到右。

图: 查询

```
SELECT ship_date, order_date, customer_num,
        order_num, po_num
        FROM orders
        ORDER BY order_date, ship_date;
```

如对多个列进行排序所示，可以使用 ORDER BY 子句来以升序或降序对数据进行排序和执行嵌套排序。此结果显示了升序。

图: 查询结果

```
ship_date   order_date customer_num     order_num po_num

        06/01/1998 05/20/1998           104         1001 B77836
        05/26/1998 05/21/1998           101         1002 9270
        05/23/1998 05/22/1998           104         1003 B77890
        05/30/1998 05/22/1998           106         1004 8006
        06/09/1998 05/24/1998           116         1005 2865
        05/30/1998              112          1006 Q13557
        06/05/1998 05/31/1998           117         1007 278693
        07/06/1998 06/07/1998           110         1008 LZ230
        06/21/1998 06/14/1998           111         1009 4745
        06/29/1998 06/17/1998           115         1010 429Q
        06/29/1998 06/18/1998           117         1012 278701
        07/03/1998 06/18/1998           104         1011 B77897
        07/10/1998 06/22/1998           104         1013 B77930
        07/03/1998 06/25/1998           106         1014 8052
        07/16/1998 06/27/1998           110         1015 MA003
        07/12/1998 06/29/1998           119         1016 PC6782
        07/13/1998 07/09/1998           120         1017 DM354331
        07/13/1998 07/10/1998           121         1018 S22942
        07/16/1998 07/11/1998           122         1019 Z55709
        07/16/1998 07/11/1998           123         1020 W2286
```

```
07/25/1998 07/23/1998          124          1021 C3288
07/30/1998 07/24/1998          126          1022 W9925
07/30/1998 07/24/1998          127          1023 KF2961
```

当对表中的若干列使用 SELECT 和 ORDER BY 时，您会发现使用整数来在 ORDER BY 子句中表示列的位置非常有用。当整数是 ORDER BY 列表中的元素时。数据库服务器将它看作是投影列表中的位置。例如，在 ORDER BY 列表中使用 3（ORDER BY 3）表示投影列表中的第三项。以下查询中的语句检索和显示相同数据，如下图 12所示。

图: 查询

```
SELECT customer_num, order_num, po_num, order_date
        FROM orders
        ORDER BY 4, 1;

        SELECT customer_num, order_num, po_num, order_date
        FROM orders
        ORDER BY order_date, customer_num;
```

图: 查询结果

```
customer_num    order_num po_num       order_date

        104          1001 B77836       05/20/1998
        101          1002 9270         05/21/1998
        104          1003 B77890       05/22/1998
        106          1004 8006         05/22/1998
        116          1005 2865         05/24/1998
        112          1006 Q13557       05/30/1998
        117          1007 278693       05/31/1998
        110          1008 LZ230        06/07/1998
        111          1009 4745         06/14/1998
        115          1010 429Q         06/17/1998
        104          1011 B77897       06/18/1998
        117          1012 278701       06/18/1998
        104          1013 B77930       06/22/1998
        106          1014 8052         06/25/1998
        110          1015 MA003        06/27/1998
        119          1016 PC6782       06/29/1998
        120          1017 DM354331     07/09/1998
        121          1018 S22942       07/10/1998
```

```
122         1019 Z55709      07/11/1998
123         1020 W2286       07/11/1998
124         1021 C3288       07/23/1998
126         1022 W9925       07/24/1998
127         1023 KF2961      07/24/1998
```

当将整数指定给列名时，可以在 ORDER BY 子句中包括 DESC 关键字。如下所示。

图: 查询

```
SELECT customer_num, order_num, po_num, order_date
        FROM orders
        ORDER BY 4 DESC, 1;
```

在此示例中，数据先按 order_date 以降序排序再按 customer_num 以升序排序。

选择子串

要选择字符列的部分值，请在投影列表中包含一个子串。假设市场营销部门计划向客户寄邮件并想要客户的基于邮政编码的地理分布。可编写与以下图中显示的查询相似的查询。

图: 查询

```
SELECT zipcode[1,3], customer_num
        FROM customer
        ORDER BY zipcode;
```

该查询使用子串来选择 zipcode 列的前三个字符（它们标识州）和全部 customer_num，并按邮政编码以升序列出它们，如以下结果所示。

图: 查询结果

```
zipcode customer_num

        021         125
        080         119
        085         122
        198         121
        322         123
        ⋮
        943         103
        943         107
        946         118
```

ORDER BY 和非英文数据

缺省情况下，对于数据库数据，GBase 8s 数据库服务器使用美国英语语言环境，称为语言环境。美国英语语言环境指定数据以代码集顺序存储。此缺省语言环境使用 ISO 8859-1 代码集。

如果您的数据库包含非英语数据，那么应在 NCHAR（或 NVARCHAR）列中存储非英语数据，以获取按语言排序的结果。ORDER BY 子句应以适合于语言的顺序返回数据。

3.2.4 使用 WHERE 子句

SELECT 语句返回的行集是其活动集。单个 SELECT 语句返回单个行。如果只想看见特定行，可将 WHERE 子句添加至 SELECT 语句。例如：使用 WHERE 子句来将数据库服务器返回的行限制为特定客户所下的订单或特定客户服务代表输入的电话。

可以使用 WHERE 子句来设置比较条件或连接条件。本节只演示第一种用法。连接条件在后面的节和下一章中描述。

3.2.5 创建比较条件

SELECT 语句的 WHERE 子句指定了您想要看到的行。比较条件使用特定关键字和运算符来定义搜索条件。

例如，可使用 BETWEEN 、IN 、LIKE 或 MATCHES 中的一个来测试相等性。或者使用关键字 IS NULL 来测试空值。可将关键字 NOT 与这些关键字中的任何一个组合来指定相反条件。

下表列出可在 WHERE 子句中用来代替关键字测试相等性的关系运算符。

运算符	**操作**
=	等于
!= 或 <>	不等于
>	大于
>=	大于并或等于
<	

小于

<=

小于或等于

对于 CHAR 表达式，大于在 ASCII 整理顺序中意味着之后，其中小写字母在大写字母之后，而大写字母和小写字母都在数字之后。请参阅《GBase 8s SQL 指南：语法》中的 ASCII 字符集图表。对于 DATE 和 DATETIME 表达式，大于意味着时间上更迟，对于 INTERVAL 表达式，它意味着更长的持续时间。

不能使用 TEXT 或 BYTE 列创建比较条件（使用 IS NULL 或 IS NOT NULL 关键字来测试 NULL 值时除外）。

不能指定 BLOB 或 CLOB 列从而在 GBase 8s 上创建比较条件（用 IS NULL 或 IS NOT NULL 关键字来测试 NULL 值时除外）。

可以在 WHERE 子句中使用上述关键字或运算符来创建执行下列操作的比较条件查询：

- 包括值
- 排除值
- 查找值范围
- 查找值的子集
- 标识 NULL 值

要使用以下条件执行变量文本搜索，在 WHERE 子句中使用上述关键字或运算符来创建比较条件查询：

- 精确文本比较
- 单字符通配符
- 受限单字符通配符
- 可变长通配符
- 下标

下一节包含说明这些查询类型的示例。

包括行

在 WHERE 子句中使用等号（=）关系运算符包括行，如以下查询所示。

图: 查询

```
SELECT customer_num, call_code, call_dtime, res_dtime
        FROM cust_calls
        WHERE user_id = 'maryj';
```

该查询返回以下行集。

图: 查询结果

```
customer_num    call_code    call_dtime          res_dtime

                106    D              1998-06-12    08:20 1998-06-12 08:25
                121    O              1998-07-10 14:05 1998-07-10 14:06
                127    I             1998-07-31 14:30
```

排除行

在 WHERE 子句中使用关系运算符 != 或 <> 排除行。

以下查询假设您从符合 ANSI 的数据库中选择；该语句指定所有者或 customer 表的创建者的登录名。当表的创建者就是当前用户时，或者当数据库不符合 ANSI 时，不需要此限定符。然而，在任一情况下都可以包括该限定符。有关所有者命名的详细讨论，请参阅《GBase 8s SQL 指南：语法》。

图: 查询

```
SELECT customer_num, company, city, state
        FROM odin.customer
        WHERE state != 'CA';

        SELECT customer_num, company, city, state
        FROM odin.customer
        WHERE state <> 'CA';
```

此查询中的两个语句都通过指定在用户 odin 拥有的 customer 表中 state 列中的值不应等于 CA 来排除值，如下所示。

图: 查询结果

```
customer_num    company                    city                state

                119    The Triathletes Club Cherry Hill        NJ
                120    Century Pro Shop        Phoenix             AZ
                121    City Sports             Wilmington          DE
                122    The Sporting Life       Princeton           NJ
                123    Bay Sports              Jacksonville        FL
                124    Putnum's Putters        Bartlesville        OK
                125    Total Fitness Sports Brighton               MA
```

```
        126   Neelie's Discount Sp Denver          CO
        127   Big Blue Bike Shop    Blue Island     NY
        128   Phoenix College       Phoenix         AZ
```

指定一定范围的行

下列查询显示在 WHERE 子句中指定一定范围内行的两种方法。

图: 查询

```
SELECT catalog_num, stock_num, manu_code, cat_advert
        FROM catalog
        WHERE catalog_num BETWEEN 10005 AND 10008;

        SELECT catalog_num, stock_num, manu_code, cat_advert
        FROM catalog
        WHERE catalog_num >= 10005 AND catalog_num <= 10008;
```

查询中的每个子句都指定 catalog_num 的范围，从 10005 至 10008（包括 10005 和 10008），第一个语句使用关键字，第二个语句使用关系运算符检索行。如下所示。

图: 查询结果

```
catalog_num   10005
        stock_num     3
        manu_code     HSK
        cat_advert    High-Technology Design Expands the Sweet Spot

        catalog_num   10006
        stock_num     3
        manu_code     SHM
        cat_advert    Durable Aluminum for High School and Collegiate Athletes

        catalog_num   10007
        stock_num     4
        manu_code     HSK
        cat_advert    Quality Pigskin with Joe Namath Signature

        catalog_num   10008
        stock_num     4
        manu_code     HRO
```

```
        cat_advert      Highest Quality Football for High School
                        and Collegiate Competitions
```

尽管 catalog 表标记具有 BYTE 数据类型的列，但该列不包括在此 SELECT 语句中，原因是输出将按列名只显示词 <BYTE value>。可以编写 SQL API 应用程序来显示 TEXT 和 BYTE 值。

排除一定范围的行

以下查询使用关键字 NOT BETWEEN 排除 zipcode 列中字符范围在 94000 到 94999 的行，如下所示。

图: 查询

```
SELECT fname, lname, city, state
        FROM customer
        WHERE zipcode NOT BETWEEN '94000' AND '94999'
        ORDER BY state;
```

图: 查询结果

```
fname           lname           city            state

        Frank           Lessor          Phoenix         AZ
        Fred            Jewell          Phoenix         AZ
        Eileen          Neelie          Denver          CO
        Jason           Wallack         Wilmington      DE
        Marvin          Hanlon          Jacksonville    FL
        James           Henry           Brighton        MA
        Bob             Shorter         Cherry Hill     NJ
        Cathy           O'Brian         Princeton       NJ
        Kim             Satifer         Blue Island     NY
        Chris           Putnum          Bartlesville    OK
```

使用 WHERE 子句查找值的子集

就像排除行，以下查询假定使用符合 ANSI 的数据库。所有者限定符在引号中，以保护文字字符串的区分大小写。

图: 查询

```
SELECT lname, city, state, phone
        FROM 'Aleta'.customer
```

```
        WHERE state = 'AZ' OR state = 'NJ'
        ORDER BY lname;

        SELECT lname, city, state, phone
        FROM 'Aleta'.customer
        WHERE state IN ('AZ', 'NJ')
        ORDER BY lname;
```

查询中的每个语句在 Aleta.customer 表的 state 列中检索包括 AZ 或 NJ 子集的行。

图: 查询结果

```
lname       city            state phone

        Jewell      Phoenix         AZ      602-265-8754
        Lessor      Phoenix         AZ      602-533-1817
        O'Brian     Princeton       NJ      609-342-0054
        Shorter     Cherry Hill     NJ      609-663-6079
```

不能使用 IN 关键字来测试 TEXT 或 BYTE 列。

另外，当使用 GBase 8s 时，不能使用 IN 关键字来测试 BLOB 或 CLOB 列。

在查询（对符合 ANSI 的数据库进行查询的示例）中，表所有者名称两边没有引号。鉴于图 1中两个语句搜索 Aleta.customer 表，以下查询搜索表 ALETA.customer，这是一个不同的表，原因在于符合 ANSI 的数据库查看所有者名称的方式。

图: 查询

```
SELECT lname, city, state, phone
        FROM Aleta.customer
        WHERE state NOT IN ('AZ', 'NJ')
        ORDER BY state;
```

上一个查询添加了关键字 NOT IN，以便子集更改为排除 state 列中的子集 AZ 和 NJ，下图以 state 列的顺序显示结果。

图: 查询结果

```
lname       city            state phone

        Pauli       Sunnyvale       CA      408-789-8075
        Sadler      San Francisco   CA      415-822-1289
        Currie      Palo Alto       CA      415-328-4543
        Higgins     Redwood City    CA      415-368-1100
        Vector      Los Altos       CA      415-776-3249
```

```
        Watson          Mountain View     CA      415-389-8789
        Ream             Palo Alto        CA      415-356-9876
        Quinn           Redwood City      CA      415-544-8729
        Miller         Sunnyvale          CA      408-723-8789
        Jaeger          Redwood City      CA      415-743-3611
        Keyes           Sunnyvale         CA      408-277-7245
        Lawson          Los Altos         CA      415-887-7235
        Beatty          Menlo Park        CA      415-356-9982
        Albertson       Redwood City      CA      415-886-6677
        Grant           Menlo Park        CA      415-356-1123
        Parmelee        Mountain View     CA      415-534-8822
        Sipes           Redwood City      CA      415-245-4578
        Baxter          Oakland           CA      415-655-0011
        Neelie          Denver            CO      303-936-7731
        Wallack         Wilmington        DE      302-366-7511
        Hanlon          Jacksonville      FL      904-823-4239
        Henry           Brighton          MA      617-232-4159
        Satifer         Blue Island       NY      312-944-5691
        Putnum          Bartlesville      OK      918-355-2074
```

标识 NULL 值

使用 IS NULL 或 IS NOT NULL 选项检查 NULL 值。NULL 值表示没有数据或未知值。NULL 值不等同于零或空白。

以下查询返回具有空 paid_date 的所有行，如下所示。

图: 查询

```
SELECT order_num, customer_num, po_num, ship_date
        FROM orders
        WHERE paid_date IS NULL
        ORDER BY customer_num;
```

图: 查询结果

```
order_num   customer_num   po_num     ship_date

        1004            106   8006       05/30/1998
        1006            112   Q13557
        1007            117   278693     06/05/1998
```

```
1012            117   278701     06/29/1998
1016            119   PC6782     07/12/1998
1017            120   DM354331 07/13/1998
```

构成复合条件

要连接两个或多个比较条件或 Boolean 表达式，使用逻辑运算符 AND 、OR 和 NOT。Boolean 表达式的值求出为 true 或 false ，如果涉及到 NULL 值，那么为 unknown。

在以下查询中，运算符 AND 组合 WHERE 子句中的两个比较表达式。

图: 查询

```
SELECT order_num, customer_num, po_num, ship_date
        FROM orders
        WHERE paid_date IS NULL
        AND ship_date IS NOT NULL
        ORDER BY customer_num;
```

该查询返回具有 NULL paid_date 或 NOT NULL ship_date 的所有值。

图: 查询结果

```
order_num customer_num po_num       ship_date

        1004            106 8006        05/30/1998
        1007            117 278693      06/05/1998
        1012            117 278701      06/29/1998
        1017            120 DM354331    07/13/1998
```

使用精确文本比较

以下示例包含一个 WHERE 子句，它通过使用关键字 LIKE 或 MATCHES 或者等号（=）关系运算符来搜索精确文本比较。与较早的示例不同，这些示例说明如何查询不在当前数据库中的表。仅当包含该表的数据库与当前数据库的 ANSI 兼容状态相同时，才能访问不在当前数据库中的表。如果当前数据库是符合 ANSI 的数据库，那么要访问的表必须也驻留在符合 ANSI 的数据库中。如果当前数据库不是符合 ANSI 的数据库，那么要访问的表必须也驻留在不符合 ANSI 的数据库中。

虽然本章前面使用的数据库是演示数据库，但是下列示例中的 FROM 子句指定了由所有者 bubba 创建的 manatee 表，该表驻留在名为 syzygy的符合 ANSI 的数据库中。有关如何访问不在不在当前数据库中的表的更多信息，请参阅《GBase 8s SQL 指南：语法》。

下列查询中的每个语句检索 description 列中具有单词 helmet 的所有行，如下所示。

图: 查询

```
SELECT stock_no, mfg_code, description, unit_price
        FROM syzygy:bubba.manatee
        WHERE description = 'helmet'
        ORDER BY mfg_code;

        SELECT stock_no, mfg_code, description, unit_price
        FROM syzygy:bubba.manatee
        WHERE description LIKE 'helmet'
        ORDER BY mfg_code;

        SELECT stock_no, mfg_code, description, unit_price
        FROM syzygy:bubba.manatee
        WHERE description MATCHES 'helmet'
        ORDER BY mfg_code;
```

该结果可能如下图所示。

图: 查询结果

```
stock_no mfg_code   description       unit_price

        991 ABC         helmet              $222.00
        991 BKE         helmet              $269.00
        991 HSK         helmet              $311.00
        991 PRC         helmet              $234.00
        991 SPR         helmet              $245.00
```

使用变量文本搜索

可对基于字段的子串搜索的变量文本查询使用关键字 LIKE 和 MATCHES。包含关键字 NOT 以指示相反的条件。

关键字 LIKE 是 SQL 的 ISO/ANSI 标准，而 MATCHES 是 GBase 8s 扩展。

变量文本搜索字符串可将列出的通配符与下表中的 LIKE 或 MATCHES 包括在一起。

下表显示了您可以与关键字 LIKE 和 MATCHES 一起使用的通配符。说明了这些符号及其含义。

关键字	符号	含义

关键字	符号	含义
LIKE	%	求值为零或多个字符
LIKE	_	求值为单个字符
LIKE	\	对下一字符的特殊有效位数进行转义
MATCHES	*	求值为零个或多个字符
MATCHES	?	求值为单个字符（空值除外）
MATCHES	[ ]	求值为单个字符或一定范围内的值
MATCHES	\	对下一字符的特殊有效位数进行转义

不能使用 LIKE 或 MATCHES 运算符测试 BLOB 、CLOB 、TEXT 或 BYTE 列。

使用单字符通配符

下列查询中的语句说明如何在 WHERE 子句中使用单字符通配符。而且，它们还演示了如何查询非当前数据库中的表。stock 表位于数据库 sloth中。除了在当前演示数据库外部之外，sloth 还在称为 meerkat 的独立数据库服务器上。

有关更多信息，请参阅在外部数据库中访问和修改数据和《 GBase 8s SQL 指南：语法》。

图: 查询

```
SELECT stock_num, manu_code, description, unit_price
        FROM sloth@meerkat:stock
        WHERE manu_code LIKE '_R_'
        AND unit_price >= 100
        ORDER BY description, unit_price;

        SELECT stock_num, manu_code, description, unit_price
        FROM sloth@meerkat:stock
        WHERE manu_code MATCHES '?R?'
        AND unit_price >= 100
        ORDER BY description, unit_price;
```

查询中的每个语句只检索 manu_code 的中间字母是 R 的那些行。如下所示。比较 '_R_'（对于 LIKE）或 '?R?' （对于 MATCHES）从左到右指定下列项：

- 任何单个字符
- 字母 R

- 任何单个字符

图: 查询结果

```
stock_num manu_code description        unit_price

    205 HRO          3 golf balls          $312.00
    2 HRO          baseball                $126.00
    1 HRO          baseball gloves       $250.00
    7 HRO          basketball              $600.00
    102 PRC          bicycle brakes        $480.00
    114 PRC          bicycle gloves        $120.00
    4 HRO          football                $480.00
    110 PRC          helmet                  $236.00
    110 HRO          helmet                  $260.00
    307 PRC          infant jogger         $250.00
    306 PRC          tandem adapter        $160.00
    308 PRC          twin jogger           $280.00
    304 HRO          watch                   $280.00
```

指定一定范围内的词首字符的 WHERE 子句

下列查询只选择 manu_code 以 A 到 H 开头的那些列，并返回结果显示的行。测试 '[A-H]' 指定从 A 到 H 之间（包括 A 和 H）的任何一个字母。对于 LIKE 关键字，不存在等价的通配符。

图: 查询

```
SELECT stock_num, manu_code, description, unit_price
        FROM stock
        WHERE manu_code MATCHES '[A-H]*'
        ORDER BY description, manu_code;
```

图: 查询结果

```
stock_num manu_code description        unit_price

    205 ANZ          3 golf balls          $312.00
    205 HRO          3 golf balls          $312.00
    2 HRO          baseball                $126.00
    3 HSK          baseball bat          $240.00
    1 HRO          baseball gloves       $250.00
    1 HSK          baseball gloves       $800.00
```

```
7 HRO          basketball          $600.00
⋮
313 ANZ          swim cap              $60.00
6 ANZ          tennis ball         $48.00
5 ANZ          tennis racquet      $19.80
8 ANZ          volleyball          $840.00
9 ANZ          volleyball net      $20.00
304 ANZ          watch               $170.00
```

具有可变长通配符的 WHERE 子句

下列查询中的语句在字符串的末尾使用通配符来检索 description 以字符 bicycle 开头的所有行。

图: 查询

```
SELECT stock_num, manu_code, description, unit_price
        FROM stock
        WHERE description LIKE 'bicycle%'
        ORDER BY description, manu_code;

        SELECT stock_num, manu_code, description, unit_price
        FROM stock
        WHERE description MATCHES 'bicycle*'
        ORDER BY description, manu_code;
```

任一语句都返回以下行。

图: 查询结果

```
stock_num manu_code description       unit_price

    102 PRC          bicycle brakes      $480.00
    102 SHM          bicycle brakes      $220.00
    114 PRC          bicycle gloves      $120.00
    107 PRC          bicycle saddle       $70.00
    106 PRC          bicycle stem         $23.00
    101 PRC          bicycle tires       $88.00
    101 SHM          bicycle tires       $68.00
    105 PRC          bicycle wheels       $53.00
    105 SHM          bicycle wheels       $80.00
```

比较 'bicycle%' 或 'bicycle*' 指定字符 bicycle 后跟零个字符或任何字符序列。它与 bicycle stem 匹配，而 stem 与通配符匹配。如果具有该描述的行存在，那么它只与字符 bicycle 匹配。

以下查询通过添加排除 PRC 的 manu_code 的另一个比较条件来缩小搜索范围。

图: 查询

```
SELECT stock_num, manu_code, description, unit_price
        FROM stock
        WHERE description LIKE 'bicycle%'
        AND manu_code NOT LIKE 'PRC'
        ORDER BY description, manu_code;
```

该语句只检索到下列行。

图: 查询结果

```
stock_num manu_code description       unit_price

      102 SHM         bicycle brakes      $220.00
      101 SHM         bicycle tires       $68.00
      105 SHM         bicycle wheels       $80.00
```

当从大型表中进行选择并在比较字符串中使用词首通配符时（如 '%cycle'），查询通常需要较长时间来执行。由于不能使用索引，所以搜索每一行。

MATCHES 子句和非缺省语言环境

缺省情况下，对数据库数据，GBase 8s 数据库服务器使用美国英语语言环境，称为语言环境。缺省的语言环境使用 ISO 8859-1 代码集。该美国英语语言环境指定 MATCHES 将使用代码集顺序。

如果数据库使用非缺省语言环境,那么指定范围的 MATCHES 子句将该语言环境的整理顺序用于字符数据类型（包括 CHAR 、NCHAR 、VARCHAR 、NVARCHAR 和 LVARCHAR）。MATCHES 范围的此功能是一般规则（只有 NCHAR 和 NVARCHAR 列可使用特定于语言环境的整理）的例外情况。然而，如果语言环境不能指定任何特殊整理顺序，那么 MATCHES 使用代码集顺序。

在 GBase 8s 中，可以使用 SET COLLATION 语句为会话指定不同于 DB_LOCALE 设置的数据库语言环境。有关 SET COLLATION 的描述，请参阅《GBase 8s SQL 指南：语法》。

保护特殊字符

下列查询使用 ESCAPE 与 LIKE 或 MATCHES 配合使用，以便您可以保护特殊字符，使它们不会被误认为是通配符。

图: 查询

```
SELECT * FROM cust_calls
        WHERE res_descr LIKE '%!%%' ESCAPE '!';
```

ESCAPE 关键字指定包含下一个字符的转义字符（在本示例中为 !）以便将它解释为数据而不是通配符。在该示例中，转义字符导致将中间的百分号（%）当作数据。通过使用 ESCAPE 关键字，您可以使用 LIKE 通配符百分号（%）在 res_descr 列中搜索百分号（%）的出现次数。查询检索下列显示的行。

图: 查询结果

```
customer_num    116
        call_dtime      1997-12-21 11:24
        user_id          mannyn
        call_code        I
        call_descr       Second complaint from this customer!
        Received two cases righthanded outfielder
        glove (1 HRO) instead of one case lefties.
        res_dtime        1997-12-27 08:19
        res_descr        Memo to shipping (Ava Brown) to send case
        of lefthanded gloves, pick up wrong case;
        memo to billing requesting 5% discount to
        placate customer due to second offense
        and lateness of resolution because of
        holiday.
```

在 WHERE 子句中使用下标

您可以在 SELECT 语句的 WHERE 子句中使用下标，以指定选择某列中一定范围内额字符或数字，如下所示。

```
SELECT catalog_num, stock_num, manu_code, cat_advert,
        cat_descr
        FROM catalog
        WHERE cat_advert[1,4] = 'High';
```

下标 [1,4] 导致该查询检索 cat_advert 列的前四个字母为 High 的所有行。如下所示。

图: 查询结果

```
catalog_num   10004
          stock_num      2
          manu_code      HRO
```

```
cat_advert     Highest Quality Ball Available, from Hand-Sti
tching to the Robinson Signature
cat_descr
Jackie Robinson signature ball. Highest professional quality,
used by National League.

catalog_num    10005
stock_num      3
manu_code      HSK
cat_advert     High-Technology Design Expands the Sweet Spot
cat_descr
Pro-style wood. Available in sizes: 31, 32, 33, 34, 35.
⋮
catalog_num    10045
stock_num      204
manu_code      KAR
cat_advert     High-Quality Beginning Set of Irons. Appropriate
for High School Competitions
cat_descr
Ideally balanced for optimum control. Nylon covered shaft.

catalog_num    10068
stock_num      310
manu_code      ANZ
cat_advert     High-Quality Kickboard
cat_descr
White. Standard size.
```

3.2.6 使用 FIRST 子句选择特定行

可以在 SELECT 语句的 Projection 子句中包含 FIRST max 规范（其中 max 具有整数值）来构建查询，使其仅返回匹配 SELECT 语句条件的最初max 行。在（且仅在）此上下文中，也可以使用关键字 LIMIT 作为 FIRST 的同义词。执行具有FIRST 子句的 SELECT 语句时返回的行可能会不同，这取决于该语句是否还包含 ORDER BY 子句。

在 Projection 子句中，后面跟无符号整数的关键字 SKIP 可用在 FIRST 或 LIMIT 关键字前面。SKIP offset 子句指示数据库服务器在返回 FIRST 子句指定的行数之前，从查询结果集中排除最初 offset 行满足条件的行。在 SPL 例程中，SKIP 、FIRST 或 LIMIT 的参数

可以是字面值整数或局部 SPL 变量。如果 Projection 子句包含 SKIP offset 但不包含 FIRST 或 LIMIT 规范，那么查询返回除最初 offset 行以外所有满足条件的行。

Projection 子句在下列上下文中不能包含 SKIP 、FIRST 或 LIMIT 关键字：

- 当 SELECT 语句是视图定义的一部分
- 在子查询中，除了外部查询的 FROM 子句
- 在跨服务器分发的查询中，其中参与的数据库服务器不支持 SKIP 、FIRST 或 LIMIT 关键字。

有关使用 FIRST 子句的限制的信息，请参阅《GBase 8s SQL 指南：语法》中 SELECT 语句的 Projection 子句的描述。

不具有 ORDER BY 子句的 FIRST 子句

如果具有 FIRST 子句的 SELECT 语句中没有 ORDER BY 子句，那么可能返回符合 SELECT 语句条件的任何行。换言之，数据库服务器确定返回哪些限定行，并且查询结果可能会不同，这取决于优化器选择的查询计划。

以下查询使用 FIRST 子句来返回 state 表中的前五行。

图: 查询

```
SELECT FIRST 5 * FROM state;
```

图: 查询结果

```
code sname

     AK    Alaska
     HI    Hawaii
     CA    California
     OR    Oregon
     WA    Washington
```

当只想知道表包含的所有列的名称和数据的类型，或者测试可能会返回许多行的查询时，可以使用 FIRST 子句。以下查询显示了如何使用 FIRST 子句来返回表的第一行的列值。

图: 查询

```
SELECT FIRST 1 * FROM orders;
```

图: 查询结果

```
order_num       1001
     order_date      05/20/1998
     customer_num    104
     ship_instruct   express
```

```
backlog          n
po_num           B77836
ship_date        06/01/1998
ship_weight      20.40
ship_charge      $10.00
paid_date        07/22/1998
```

具有 ORDER BY 子句的 FIRST 子句

可以在具有 FIRST 子句的 SELECT 语句中包括 ORDER BY 子句，以返回包含指定列的最高值或最低值的行。以下查询显示了包含 ORDER BY 子句以（按字母顺序）返回包含在 state 表中的前五个州的查询。该查询，（除 ORDER BY 子句以外，它与图 1相同）返回不同于图 1的一组行。

图: 查询

```
SELECT FIRST 5 * FROM state ORDER BY sname;
```

图: 查询结果

```
code sname

AL    Alabama
AK    Alaska
AZ    Arizona
AR    Arkansas
CA    California
```

以下查询显示如何在具有 ORDER BY 子句的查询中使用 FIRST 子句来查找 stock 表中列出的 10 中最贵的商品。

图: 查询

```
SELECT FIRST 10 description, unit_price
        FROM stock ORDER BY unit_price DESC;
```

图: 查询结果

```
description       unit_price

football              $960.00
volleyball            $840.00
baseball gloves       $800.00
18-spd, assmbld       $685.90
irons/wedge           $670.00
```

```
basketball          $600.00
12-spd, assmbld     $549.00
10-spd, assmbld     $499.99
football            $480.00
bicycle brakes      $480.00
```

应用程序可以将 Projection 子句的 SKIP 和 FIRST 关键字与 ORDER BY 子句相结合使用，以执行连续查询，对某些固定大小（例如，最大行数可在一屏显示，无需滚动）的子集中所有满足条件的行进行增量检索。通过在每次查询后使用 FIRST 子句的 max 参数增大 SKIP 子句的 offset 参数值可实现上述操作。通过对满足条件的行施加唯一的命令，ORDER BY 子句确保每次查询返回满足条件行的不同子集。

以下查询显示了包含 SKIP 、FIRST 和 ORDER BY 规范以（按字母顺序）返回 state 表的 10 个州中的第六个州，而不是前五个州的查询。该查询类似于图 1，但 SKIP 5 规范指示数据库服务器返回不同于图 1的行集。

图: 查询

```
SELECT SKIP 5 FIRST 5 * FROM state ORDER BY sname;
```

图: 查询结果

```
code sname

CO   Colorado
CT   Connecticut
DE   Delaware
FL   Florida
GA   Georgia
```

如果使用 SKIP 、FIRST 和 ORDER BY 关键字，必须指定对应于应用程序设计目标的参数，如果 SKIP 的 offset 参数大于满足条件的行数，那么任何 FIRST 或 LIMIT 规范都无效，并且查询不会返回任何结果。

3.2.7 表达式和派生的值

不限制您按名称选择列。可以在 SELECT 语句的 Projection 子句中列出表达式来执行对列数据的计算，并显示派生自一列或多列的内容的信息。

表达式由列名、常量、带引号字符串、关键字或用运算符连接的这些项的任何组合组成。当在程序中嵌入 SELECT 语句时，它还可以包括主变量（程序数据）。

算术表达式

算术运算符至少包含下表中列出的算术运算符之一并产生一个数字

运算符

	操作
+	加
-	减
*	乘
/	除

不能在算术表达式中使用 TEXT 或 BYTE 列。

在 GBase 8s，不能在算术表达式中指定 BLOB 或 CLOB 。

算术表达式使您能够查看建议的计算的结果而不必实际改变数据库中的数据。可以添加 INTO TEMP 子句来将已改变的数据保存在临时表中以供将来参考、计算或即时报告。

当 unit_price 为 $400 或更多时，以下查询对 unit_price 列计算 7% 的销售税（但不在数据库中更新它）。

图: 查询

```
SELECT stock_num, description, unit_price, unit_price * 1.07
        FROM stock
        WHERE unit_price >= 400;
```

此结果在 expression 列中显示。

图: 查询结果

```
stock_num description       unit_price      (expression)

        1 baseball gloves     $800.00           $856.00
        1 baseball gloves     $450.00           $481.50
        4 football            $960.00          $1027.20
        4 football            $480.00           $513.60
        7 basketball          $600.00           $642.00
        8 volleyball          $840.00           $898.80
        102 bicycle brakes    $480.00           $513.60
        111 10-spd, assmbld   $499.99           $534.99
        112 12-spd, assmbld   $549.00           $587.43
        113 18-spd, assmbld   $685.90           $733.91
```

```
203 irons/wedge          $670.00          $716.90
```

当订货数量小于 5 时，下列查询对订单计算 $6.50 的附加费用。

图: 查询

```
SELECT item_num, order_num, quantity,
        total_price, total_price + 6.50
        FROM items
        WHERE quantity < 5;
```

结果显示在 expression 列中。

图: 查询结果

```
item_num    order_num quantity total_price   (expression)

      1         1001        1     $250.00        $256.50
      1         1002        1     $960.00        $966.50
      2         1002        1     $240.00        $246.50
      1         1003        1      $20.00         $26.50
      2         1003        1     $840.00        $846.50
      1         1004        1     $250.00        $256.50
      2         1004        1     $126.00        $132.50
      3         1004        1     $240.00        $246.50
      4         1004        1     $800.00        $806.50
      :
      1         1023        2      $40.00         $46.50
      2         1023        2     $116.00        $122.50
      3         1023        1      $80.00         $86.50
      4         1023        1     $228.00        $234.50
      5         1023        1     $170.00        $176.50
      6         1023        1     $190.00        $196.50
```

下列查询计算并在 expression 列中显示按接收到客户电话（call_dtime）与处理电话（res_dtime）之间的时间间隔（以天、小时和分钟计）。

图: 查询

```
SELECT customer_num, call_code, call_dtime,
        res_dtime - call_dtime
        FROM cust_calls
        ORDER BY customer_num;
```

图: 查询结果

```
customer_num call_code call_dtime                (expression)

         106 D          1998-06-12 08:20          0 00:05
         110 L          1998-07-07 10:24          0 00:06
         116 I          1997-11-28 13:34          0 03:13
         116 I          1997-12-21 11:24          5 20:55
         119 B          1998-07-01 15:00          0 17:21
         121 O          1998-07-10 14:05          0 00:01
         127 I          1998-07-31 14:30
```

使用显示标签

可以将显示标签指定给计算或派生的数据列，以替换缺省列头 expression。在图 1 、图 3 和 图 1中，派生数据显示在 expression 列中。下列查询还显示派生值，但显示派生值的列具有描述性头 taxed。

图: 查询

```
SELECT stock_num, description, unit_price,
        unit_price * 1.07 taxed
        FROM stock
        WHERE unit_price >= 400;
```

结果显示将标记 taxed 指定给用于显示操作 unit_price * 1.07 的结果的投影列表中的表达式。

图: 查询结果

```
stock_num description       unit_price          taxed

         1 baseball gloves     $800.00           $856.00
         1 baseball gloves     $450.00           $481.50
         4 football            $960.00          $1027.20
         4 football            $480.00           $513.60
         7 basketball          $600.00           $642.00
         8 volleyball          $840.00           $898.80
         102 bicycle brakes    $480.00           $513.60
         111 10-spd, assmbld   $499.99           $534.99
         112 12-spd, assmbld   $549.00           $587.43
         113 18-spd, assmbld   $685.90           $733.91
         203 irons/wedge       $670.00           $716.90
```

在下列查询中，为显示操作 total_price + 6.50 的结果的列定义标签 surcharge。

图: 查询

```
SELECT item_num, order_num, quantity,
        total_price, total_price + 6.50 surcharge
        FROM items
        WHERE quantity < 5;
```

在输出中对 surcharge 列添加标签。

图: 查询结果

```
item_num    order_num quantity total_price      surcharge

      1         1001        1      $250.00         $256.50
      1         1002        1      $960.00         $966.50
      2         1002        1      $240.00         $246.50
      1         1003        1       $20.00          $26.50
      2         1003        1      $840.00         $846.50
      :
      1         1023        2       $40.00          $46.50
      2         1023        2      $116.00         $122.50
      3         1023        1       $80.00          $86.50
      4         1023        1      $228.00         $234.50
      5         1023        1      $170.00         $176.50
      6         1023        1      $190.00         $196.50
```

下列查询将标签 span 指定给显示从 DATETIME 列 res_dtime 减去 DATETIME 列 call_dtime 的结果的列。

图: 查询

```
SELECT customer_num, call_code, call_dtime,
        res_dtime - call_dtime span
        FROM cust_calls
        ORDER BY customer_num;
```

在输出中标记了 span 列。

图: 查询结果

```
customer_num call_code call_dtime              span

         106 D          1998-06-12 08:20        0 00:05
         110 L          1998-07-07 10:24        0 00:06
         116 I          1997-11-28 13:34        0 03:13
```

```
116 I          1997-12-21 11:24          5 20:55
119 B           1998-07-01 15:00          0 17:21
121 O           1998-07-10 14:05          0 00:01
127 I          1998-07-31 14:30
```

CASE 表达式

CASE 表达式条件表达式，类似于编程语言中的 CASE 语句。当要更改表示数据的方式时，可以使用 CASE 表达式。CASE 表达式允许语句返回若干可能结果之一，这取决于若干条件测试中哪一个求值为 TRUE。

在 CASE 表达式中不允许 TEXT 或 BYTE 值。

考虑用数字表示婚姻状态的列，1 、2 、3 和 4 是相应表示单身、已婚、离异和丧偶的值。在某些情况下，考虑到数据库的效率，您可能更想存储短的值（1 、2 、3 和 4），但人力资源部的职员可能更具有描述性的值（单身、已婚、离异和丧偶）。CASE 表达式简化了这种不同值集之间的转换。

下列示例显示具有多个 WHEN 子句的 CASE 表达式，它返回 stock 表的 manu_code 列更具有描述性的值。如果没有任何 WHEN 条件为 true，那么 NULL 是缺省的结果。（可以省略 ELSE NULL 子句。）

```
SELECT
        CASE
        WHEN manu_code = "HRO" THEN "Hero"
        WHEN manu_code = "SHM" THEN "Shimara"
        WHEN manu_code = "PRC" THEN "ProCycle"
        WHEN manu_code = "ANZ" THEN "Anza"
        ELSE NULL
        END
        FROM stock;
```

在 CASE 表达式中必须至少包含一个 WHEN 子句；后续的 WHEN 子句和 ELSE 子句是可选的。如果没有 WHEN 条件求值为 true，那么结果值为 NULL。可以使用 IS NULL 表达式来处理为 NULL 值的结果。有关处理值（NULL）的信息，请参阅《GBase 8s SQL 指南：语法》。

下列查询显示了简单的 CASE 表达式，它返回一个字符串值来标记 orders 表中尚未交付给客户的任何订单。

图: 查询

```
SELECT order_num, order_date,
        CASE
```

```
        WHEN ship_date IS NULL
        THEN "order not shipped"
        END
        FROM orders;
```

图: 查询结果

```
order_num order_date (expression)

    1001 05/20/1998
    1002 05/21/1998
    1003 05/22/1998
    1004 05/22/1998
    1005 05/24/1998
    1006 05/30/1998 order not shipped
    1007 05/31/1998
    ⋮
    1019 07/11/1998
    1020 07/11/1998
    1021 07/23/1998
    1022 07/24/1998
    1023 07/24/1998
```

有关如何使用 CASE 表达式来更新列的信息，请参阅更新列的 CASE 表达式。

对派生列进行排序

当想要在表达式中使用 ORDER BY 时，可以使用指定给表达式的显示标注或整数。如图 1和图 3所示。

图: 查询

```
SELECT customer_num, call_code, call_dtime,
        res_dtime - call_dtime span
        FROM cust_calls
        ORDER BY span;
```

下列查询从 cust_calls 表中检索图 5所检索的相同数据。在此查询中， ORDER BY 子句导致以 span 列中派生值的升序显示数据，如下所示。

图: 查询结果

```
customer_num call_code call_dtime                span
```

```
127 I          1998-07-31 14:30
121 O           1998-07-10 14:05          0 00:01
106 D           1998-06-12 08:20          0 00:05
110 L           1998-07-07 10:24          0 00:06
116 I          1997-11-28 13:34          0 03:13
119 B           1998-07-01 15:00          0 17:21
116 I          1997-12-21 11:24          5 20:55
```

下列查询使用整数表示运算 res_dtime - call_dtime 的结果，并检索出现在上一结果中的相同行。

图: 查询

```
SELECT customer_num, call_code, call_dtime,
        res_dtime - call_dtime span
        FROM cust_calls
        ORDER BY 4;
```

3.2.8 在 SELECT 语句中使用 Rowid 值

数据库服务器将唯一的 rowid 指定给未分片的表中的行。实际上，rowid 是每个表中的隐藏列。rowid 的顺序值没有任何特殊意义，可能根据 chunk 中的物理数据的位置的不同而变化。可以使用 rowid 来找到与表中的某行相关联的内部记录号。分片表中的行不自动包含 rowid 列。

建议您在应用程序中使用主键而不是 rowid 作为访问的方法。因为主键是用 SQL 的 ANSI 规范定义的，所以使用它们来访问数据提高了应用程序的可移植性。另外，当数据库服务器使用主键时，它访问分片表中的数据所需的时间比使用rowid 时访问相同数据所需的时间要要少。

有关 rowid 的更多信息，请参阅《GBase 8s 管理员指南》。

下列查询在 Projection 子句中使用 rowid 和星号（*）来检索 manufact 表中的每一行及其相应的 rowid。

图: 查询

```
SELECT rowid, * FROM manufact;
```

图: 查询结果

```
rowid manu_code manu_name          lead_time

      257 SMT         Smith                3
      258 ANZ         Anza                 5
      259 NRG         Norge                7
```

```
260 HSK         Husky          5
261 HRO         Hero           4
262 SHM         Shimara        30
263 KAR         Karsten        21
264 NKL         Nikolus        8
265 PRC         ProCycle       9
```

不要在 permanent 表中存储 rowid 或尝试将它用作外键。如果删除了一个表然后从外部数据重新装入它，那么所有 rowid 都将不同。

3.3 多表 SELECT 语句

要从两个或多个表中选择数据，在 FROM 子句中指定表名。添加 WHERE 子句一章每个表中的至少一个相关列间创建连接条件。WHERE 子句创建临时组合表。在其中，满足连接添加的每一对行都被链接以组成单个行。

简单连接根据每个表中某列的关系组合来自两个或多个表的信息。组合连接根据每个表中两个或多个列之间的关系连接两个或多个表。

要创建连接，必须在每个表中至少一列之间指定称为连接条件的关系。因为要对列进行比较，所以它们必须具有兼容的数据类型。当连接大型表时，对连接条件中的列进行索引会提高性能。

数据类型在 GBase 8s SQL 参考指南中描述。索引在 GBase 8s 管理员指南中详细所讨论。

3.3.1 创建笛卡尔积

当执行未显式声明表之间的连接条件的多表查询时，就创建了笛卡尔积。笛卡尔积由表的行的每种可能的组合构成。此结果通常很大且不实用。

以下查询从两个表中进行选择并生成笛卡尔积。

图: 查询

```
SELECT * FROM customer, state;
```

state 表只有 52 行，customer 表只有 28 行，然而查询的影响是将一个表的行数乘以另一个表的行数并检索不实用的 1,456 行，如下所示。

图: 查询结果

```
customer_num    101
            fname           Ludwig
            lname           Pauli
            company         All Sports Supplies
```

```
address1        213 Erstwild Court
address2
city            Sunnyvale
state           CA
zipcode         94086
phone           408-789-8075
code            AK
sname           Alaska

customer_num    101
fname           Ludwig
lname           Pauli
company         All Sports Supplies
address1        213 Erstwild Court
address2
city            Sunnyvale
state           CA
zipcode         94086
phone           408-789-8075
code            HI
sname           Hawaii

customer_num    101
fname           Ludwig
lname           Pauli
company         All Sports Supplies
address1        213 Erstwild Court
address2
city            Sunnyvale
state           CA
zipcode         94086
phone           408-789-8075
code            CA
sname           California
:
```

另外，显示在连续行中的某些数据是矛盾的。例如：虽然 customer 表中的 city 和 state 指示在 California 的地址，但是 state 表的 code 和sname 可能是另一个州的。

3.3.2 创建连接

在概念上，任何连接的第一阶段是创建笛卡尔积，要改进或限制此笛卡尔积并除去数据行的无意义组合，在 SELECT 语句的 WHERE 子句中包括有效的连接条件。

本节说明了跨连接、等值连接、自然连接和多表连接。其他复杂构成（如自连接和外链接）在编写高级 SELECT 语句中讨论。

跨连接

跨连接组合所有选择的表中的所有行并创建笛卡尔积。跨连接的结果可能会非常大并且难于管理。

下列查询使用 ANSI 连接语法创建跨连接。

图: 查询

```
SELECT * FROM customer CROSS JOIN    state;
```

该查询的结果与图 1的结果完全相同。另外，可能通过指定 WHERE 子句来过滤跨连接。

有关笛卡尔积的更多信息，请参阅创建笛卡尔积。有关 ANSI 语法的更多信息，请参阅ANSI 连接语法。

等值连接

等值连接是基于相等或匹配列值的连接。在 WHERE 子句中，使用作为比较运算符的等号（=）来表示这一相等关系。如下所示。

图: 查询

```
SELECT * FROM manufact, stock
            WHERE manufact.manu_code = stock.manu_code;
```

该查询在 manu_code 列上连接 manufact 和 stock 表。它只检索两个列的值相等的那些行。以下结果显示了一些这样的行。

图: 查询结果

```
manu_code      SMT
               manu_name      Smith
               lead_time       3
               stock_num       1
               manu_code      SMT
               description     baseball gloves
               unit_price      $450.00
```

```
unit            case
unit_descr      10 gloves/case

manu_code     SMT
manu_name     Smith
lead_time      3
stock_num      5
manu_code     SMT
description     tennis racquet
unit_price      $25.00
unit            each
unit_descr      each

manu_code     SMT
manu_name     Smith
lead_time      3
stock_num      6
manu_code     SMT
description     tennis ball
unit_price      $36.00
unit            case
unit_descr      24 cans/case

manu_code      ANZ
manu_name     Anza
lead_time      5
stock_num      5
manu_code     ANZ
description     tennis racquet
unit_price      $19.80
unit            each
unit_descr      each
⋮
```

在等值连接中，该结果同时包括 manufact 和 stock 表中的 manu_code 列，原因是选择列表请求每个列。

还可以使用附加约束创建等值连接，此时比较条件基于连接列中值的不相等性。这些连接在 WHERE 子句中指定的比较条件中除等号（=）之外还使用其他关系运算符。

要连接包含相同名称的列的表，用列的表名和句点（.）限定每个列名，如下列查询所示。

图: 查询

```
SELECT order_num, order_date, ship_date, cust_calls.*
        FROM orders, cust_calls
        WHERE call_dtime >= ship_date
        AND cust_calls.customer_num = orders.customer_num
        ORDER BY orders.customer_num;
```

该查询连接 customer_num 列，然后只选择 cust_calls 表中 call_dtime 大于或等于 orders 表中的 ship_date 那些行。该结果显示它返回的组合行。

图: 查询结果

```
order_num       1004
        order_date      05/22/1998
        ship_date       05/30/1998
        customer_num    106
        call_dtime      1998-06-12 08:20
        user_id         maryj
        call_code       D
        call_descr      Order received okay, but two of the cans of
        ANZ tennis balls within the case were empty
        res_dtime       1998-06-12 08:25
        res_descr       Authorized credit for two cans to customer,
        issued apology. Called ANZ buyer to report
        the qa problem.

        order_num       1008
        order_date      06/07/1998
        ship_date       07/06/1998
        customer_num    110
        call_dtime      1998-07-07 10:24
        user_id         richc
        call_code       L
        call_descr      Order placed one month ago (6/7) not received.
        res_dtime       1998-07-07 10:30
        res_descr       Checked with shipping (Ed Smith). Order out
```

```
        yesterday-was waiting for goods from ANZ.
        Next time will call with delay if necessary.

        order_num          1023
        order_date         07/24/1998
        ship_date          07/30/1998
        customer_num       127
        call_dtime         1998-07-31 14:30
        user_id            maryj
        call_code          I
        call_descr         Received Hero watches (item # 304) instead
        of ANZ watches
        res_dtime
        res_descr          Sent memo to shipping to send ANZ item 304
        to customer and pickup HRO watches. Should
        be done tomorrow, 8/1
```

自然连接

自然连接是等值连接的一种，构建它来使连接列不会多余地显示数据，如以下查询所示。

图: 查询

```
SELECT manu_name, lead_time, stock.*
        FROM manufact, stock
        WHERE manufact.manu_code = stock.manu_code;
```

类似等值连接的示例，该查询在 manu_code 列上连接 manufact 和 stock 表。因为更接近地定义了投影列表，所以只对检索到的每一行列出一次 manu_code ，如下所示。

图: 查询结果

```
manu_name       Smith
        lead_time          3
        stock_num       1
        manu_code       SMT
        description    baseball gloves
        unit_price     $450.00
        unit            case
        unit_descr     10 gloves/case
```

```
manu_name       Smith
lead_time        3
stock_num       5
manu_code       SMT
description     tennis racquet
unit_price      $25.00
unit             each
unit_descr      each

manu_name       Smith
lead_time        3
stock_num       6
manu_code       SMT
description     tennis ball
unit_price      $36.00
unit             case
unit_descr      24 cans/case

manu_name       Anza
lead_time        5
stock_num       5
manu_code       ANZ
description     tennis racquet
unit_price      $19.80
unit             each
unit_descr      each
:
```

所有的连接都是相关联的。即，WHERE 子句中的连接术语不影响连接的意义。

下列查询中的两个语句都创建相同的自然连接。

图: 查询

```
SELECT catalog.*, description, unit_price, unit, unit_descr
        FROM catalog, stock
        WHERE catalog.stock_num = stock.stock_num
        AND catalog.manu_code = stock.manu_code
        AND catalog_num = 10017;
```

```
SELECT catalog.*, description, unit_price, unit, unit_descr
FROM catalog, stock
WHERE catalog_num = 10017
AND catalog.manu_code = stock.manu_code
AND catalog.stock_num = stock.stock_num;
```

每个语句检索到下列行。

图: 查询结果

```
catalog_num   10017
        stock_num      101
        manu_code      PRC
        cat_descr
        Reinforced, hand-finished tubular. Polyurethane belted.
        Effective against punctures. Mixed tread for super wear
        and road grip.
        cat_picture   <BYTE value>

        cat_advert     Ultimate in Puncture Protection, Tires
        Designed for In-City Riding
        description   bicycle tires
        unit_price    $88.00
        unit           box
        unit_descr    4/box
```

图 3包括 TEXT 列 cat_descr 、BYTE 列 cat_picture 和 VARCHAR 列 cat_advert。

多表连接

多表连接在一个或多个相关联列上连接两个以上的表。它可以是等值连接或自然连接。

下列查询在 catalog 、stock 和 manufact 表上创建等值连接。

图: 查询

```
SELECT * FROM catalog, stock, manufact
        WHERE catalog.stock_num = stock.stock_num
        AND stock.manu_code = manufact.manu_code
        AND catalog_num = 10025;
```

该查询检索到下列行。

图: 查询结果

```
catalog_num   10025
            stock_num       106
            manu_code       PRC
            cat_descr
            Hard anodized alloy with pearl finish; 6mm hex bolt hard ware.
            Available in lengths of 90-140mm in 10mm increments.
            cat_picture    <BYTE value>

            cat_advert     ProCycle Stem with Pearl Finish
            stock_num       106
            manu_code       PRC
            description     bicycle stem
            unit_price      $23.00
            unit            each
            unit_descr      each
            manu_code       PRC
            manu_name       ProCycle
            lead_time       9
```

manu_code 重复三次，每个表一次，stock_num 重复两次。

为避免多表查询的大量重复（如图 1），在投影列表中包括特定的列以更确切地定义 SELECT 语句，如下所示。

图: 查询

```
SELECT catalog.*, description, unit_price, unit,
            unit_descr, manu_name, lead_time
            FROM catalog, stock, manufact
            WHERE catalog.stock_num = stock.stock_num
            AND stock.manu_code = manufact.manu_code
            AND catalog_num = 10025;
```

该查询使用通配符来从具有大多数列的表中选择所有列，然后从其他两个表中指定列。下表显示了此查询生成的自然连接。它与前一示例显示相同的信息，但不重复。

图: 查询结果

```
catalog_num   10025
            stock_num       106
            manu_code       PRC
            cat_descr
            Hard anodized alloy with pearl finish. 6mm hex bolt
```

```
        hardware. Available in lengths of 90-140mm in 10mm increments.
        cat_picture    <BYTE value>

        cat_advert     ProCycle Stem with Pearl Finish
        description    bicycle stem
        unit_price     $23.00
        unit           each
        unit_descr     each
        manu_name      ProCycle
        lead_time      9
```

3.3.3 某些查询快捷方式

可以使用别名，INTO TEMP 子句和显示标注来加快连接和多表查询，并生成输出以用于其它用途。

别名

可以在 SELECT 语句的 FROM 子句中将别名指定给表，以使多表查询更节省时间，可读性更高。每当要使用表名时，就可以使用别名，例如：在其他子句中作为列名的前缀。

图: 查询

```
SELECT s.stock_num, s.manu_code, s.description,
        s.unit_price, c.catalog_num,
        c.cat_advert, m.lead_time
        FROM stock s, catalog c, manufact m
        WHERE s.stock_num = c.stock_num
        AND s.manu_code = c.manu_code
        AND s.manu_code = m.manu_code
        AND s.manu_code IN ('HRO', 'HSK')
        AND s.stock_num BETWEEN 100 AND 301
        ORDER BY catalog_num;
```

SELECT 语句的相关特性允许您在定义别名之前使用别名。在此查询中，stock 表的别名是 s，catalog 表的别名是 c，manufact 表的别名是 m，它们分别在 FROM 子句中指定。并在整个 SELECT 和 WHERE 子句中用作列前缀。

将图 1 的长度与下列查询比较，后者不使用别名。

图: 查询

```
SELECT stock.stock_num, stock.manu_code, stock.description,
```

```
        stock.unit_price, catalog.catalog_num,
        catalog.cat_advert,
        manufact.lead_time
        FROM stock, catalog, manufact
        WHERE stock.stock_num = catalog.stock_num
        AND stock.manu_code = catalog.manu_code
        AND stock.manu_code = manufact.manu_code
        AND stock.manu_code IN ('HRO', 'HSK')
        AND stock.stock_num BETWEEN 100 AND 301
        ORDER BY catalog_num;
```

图 1 和 图 2 是等价的且都检索到以下查询显示的数据。

图: 查询结果

```
stock_num       110
        manu_code       HRO
        description     helmet
        unit_price      $260.00
        catalog_num     10033
        cat_advert      Lightweight Plastic with Vents Assures Cool
        Comfort Without Sacrificing Protection
        lead_time           4

        stock_num       110
        manu_code       HSK
        description     helmet
        unit_price      $308.00
        catalog_num     10034
        cat_advert      Teardrop Design Used by Yellow Jerseys; You
        Can Time the Difference
        lead_time           5
        ⋮
```

不能将 ORDER BY 子句用于 TEXT 列 cat_descr 或 BYTE 列 cat_picture。

可以使用别名来缩短对不在当前数据库中的表的查询时间。

下列查询连接驻留在不同数据库和系统（均不是当前数据库或系统）中的 2 个表中的列。

图: 查询

```
SELECT order_num, lname, fname, phone

        FROM masterdb@central:customer c, sales@western:orders o

        WHERE c.customer_num = o.customer_num

        AND order_num <= 1010;
```

通过分别将 c 和 o 指定给长 database@system:table 名称 masterdb@central:customer 和 sales@western:orders，您可以使用别名来缩短 WHERE 子句中的表达式并检索数据，如下所示。

图: 查询结果

```
order_num lname          fname          phone

        1001 Higgins       Anthony        415-368-1100
        1002 Pauli         Ludwig         408-789-8075
        1003 Higgins       Anthony        415-368-1100
        1004 Watson         George         415-389-8789
        1005 Parmelee      Jean           415-534-8822
        1006 Lawson         Margaret       415-887-7235
        1007 Sipes         Arnold         415-245-4578
        1008 Jaeger        Roy             415-743-3611
        1009 Keyes          Frances        408-277-7245
        1010 Grant         Alfred         415-356-1123
```

有关如何访问不在当前数据库中的表的更多信息，请参阅访问其他数据库服务器和《 GBase 8s SQL 指南：语法》。

还可以使用同义词作为不在当前数据库中的表以及当前表和视图的长名称的简写引用。

INTO TEMP 子句

通过将 INTO TEMP 子句添加到您的 SELECT 语句，可以在独立的表中临时保存多表查询的结果，您可以查询或处理该表，而无需修改数据库。当结束 SQL 会话或者程序或报告终止时，就会删除临时表。

下列查询创建名为 stockman 的临时表，并将在其中保存查询的结果。由于临时表中的所有列都必须具有名称，所以别名 adj_price 是必需的。

图: 查询

```
SELECT DISTINCT stock_num, manu_name, description,
        unit_price, unit_price * 1.05    adj_price
        FROM stock, manufact
        WHERE manufact.manu_code = stock.manu_code
        INTO TEMP stockman;
        SELECT * from stockman;
```

图: 查询结果

stock_num	manu_name	description	unit_price	adj_price
1	Hero	baseball gloves	$250.00	$262.5000
1	Husky	baseball gloves	$800.00	$840.0000
1	Smith	baseball gloves	$450.00	$472.5000
2	Hero	baseball	$126.00	$132.3000
3	Husky	baseball bat	$240.00	$252.0000
4	Hero	football	$480.00	$504.0000
4	Husky	football	$960.00	$1008.0000
⋮				
306	Shimara	tandem adapter	$190.00	$199.5000
307	ProCycle	infant jogger	$250.00	$262.5000
308	ProCycle	twin jogger	$280.00	$294.0000
309	Hero	ear drops	$40.00	$42.0000
309	Shimara	ear drops	$40.00	$42.0000
310	Anza	kick board	$84.00	$88.2000
310	Shimara	kick board	$80.00	$84.0000
311	Shimara	water gloves	$48.00	$50.4000
312	Hero	racer goggles	$72.00	$75.6000
312	Shimara	racer goggles	$96.00	$100.8000
313	Anza	swim cap	$60.00	$63.0000
313	Shimara	swim cap	$72.00	$75.6000

可以查询此表并将该表与其它表连接，这可以避免多次排序，并使您能够更快地在数据库中移动。有关临时表的更多信息，请参阅《GBase 8s SQL 指南：语法》和《GBase 8s 管理员指南》。

3.4 总结

本章介绍了用于查询关系数据库的基本 SELECT 语句类型的语法示例和结果。单个表的 SELECT 语句一节显示了如何执行以下操作：

- 使用 Projection 和 FROM 子句从表中选择列和行
- 使用 Projection 、FROM 和 WHERE 子句从表中选择行
- 在 Projection 子句中使用 DISTINCT 或 UNIQUE 关键字来消除查询结果中重复的行
- 使用 ORDER BY 子句和 DESC 关键字来排序检索的数据
- 选择包含非英语字符的数据值并对其排序
- 在 WHERE 子句中使用 BETWEEN 、IN 、MATCHES 和 LIKE 关键字以及各种关系运算符来创建比较条件
- 创建包括值、排除值、查找一定范围内的值（使用关键字、关系运算符和下标）查找值的子集的比较条件
- 使用精确文本比较、变长通配符和受限及非受限通配符来执行变量文本搜索
- 使用逻辑运算符 AND 、OR 和 NOT 来在 WHERE 子句中连接搜索条件或 Boolean 表达式
- 使用 ESCAPE 关键字来保护查询中的特殊字符
- 在 WHERE 子句中使用 IS NULL 和 IS NOT NULL 关键字来搜索 NULL 值
- 使用 FIRST 子句指定查询只返回符合 SELECT 语句的条件的指定书目的行
- 在 Projection 子句中使用算术运算符对数字字段执行计算并显示派生数据
- 将显示标签指定个计算列作为用于报告的格式化工具

本章还介绍了简单连接条件，使您能够从两个或多个表中选择和显示数据。多表 SELECT 语句一节描述了如何执行下列操作：

- 创建笛卡尔积
- 创建 CROSS JOIN，它创建笛卡尔积
- 在查询中将 WHERE 子句与有效连接条件包括在一起以抑制笛卡尔积
- 定义和创建自然连接和等值连接
- 在一列或多列上连接两个或多个表
- 在多表查询中使用别名作为快捷方式
- 使用 INTO TEMP 子句将选择的数据检索到独立的临时表中，以便在数据库外部执行计算

4 从复杂类型选择数据

本章描述如何查询复杂数据类型。复杂数据类型是使用 SQL 类型构造函数从其他数据类型的组合构建的。SQL 语句可以访问复杂数据类型中的个别组件。复杂数据类型是行类型或集合类型。

ROW 类型具有组合一个或多个相关数据字段的实例。这两种 ROW 类型是已命名和未命名。

集合类型具有这样的实例：在其中，每种集合值包含具有相同数据类型的一组元素，这些数据类型可以是任何基本或复杂数据类型。集合可以由 LIST 、SET 或 MULTISET 数据类型组成。

重要： 对于复杂数据类型没有跨数据库的支持。只能在本地数据库中对它们进行操作。

有关数据库服务器支持的数据类型的更完整描述，请参阅《GBase 8s SQL 参考指南》中的数据类型一章。

有关如何创建使用复杂类型的信息，请参阅《GBase 8s SQL 参考指南》和《GBase 8s SQL 指南：语法》 。

4.1 选择行类型数据

本节描述如何查询定义为行类型的数据。ROW 类型是一个复杂类型，包含一个或多个相关数据字段。

两种 ROW 类型如下：

已命名 ROW 类型

已命名的 ROW 类型可以定义表、列、其它行类型列的字段、程序变量、语句变量以及例程返回值。

未命名 ROW 类型

未命名 ROW 类型可以定义列、其它行类型列的字段、程序变量、语句局部变量、例程返回值和常量。

本节中使用的示例使用已命名 ROW 类型 zip_t 、address_t 和 employee_t，这些 ROW 类型定义 employee 表。下图显示创建 ROW 类型和表的 SQL 语法。

图: 创建 ROW 类型和表的 SQL 语法

```
CREATE ROW TYPE zip_t
                (
                z_code      CHAR(5),
                z_suffix    CHAR(4)
```

```
)

CREATE ROW TYPE address_t
(
street      VARCHAR(20),
city        VARCHAR(20),
state       CHAR(2),
zip         zip_t
)

CREATE ROW TYPE employee_t
(
name        VARCHAR(30),
address     address_t,
salary      INTEGER
)

CREATE TABLE employee OF TYPE employee_t
```

已命名的 ROW 类型 zip_t 、address_t 和 employee_t 充当类型表 employee 的字段和列的模板。类型表是在已命名 ROW 类型上定义的表。充当 employee 表的模板的 employee_t 类型将 address_t 类型用作 address 字段的数据类型。address_t 类型使用 zip_t 类型作为 zip 字段的数据类型。

下图显示了创建 student 表的 SQL 语法。student 表的 s_address 列定义为未命名的 ROW 类型。（s_address 列被定义为已命名的 ROW 类型。）

图: 创建 student 表的 SQL 语法

```
CREATE TABLE student
(
s_name        VARCHAR(30),
s_address     ROW(street VARCHAR (20), city VARCHAR(20),
state CHAR(2), zip VARCHAR(9)),
grade_point_avg DECIMAL(3,2)
)
```

4.1.1 选择类型表的列

对类型表的查询与对任何其他表的查询没有区别。例如：下列查询使用星号（*）来指定返回 employee 表所有列的 SELECT 语句。

图: 查询

```
SELECT * FROM employee
```

employee 表上的 SELECT 语句返回所有列的所有行。

图: 查询结果

```
name          Paul, J.
              address      ROW(102 Ruby, Belmont, CA, 49932, 1000)
              salary       78000

              name         Davis, J.
              address      ROW(133 First, San Jose, CA, 85744, 4900)
              salary       75000
              :
```

下列查询显示如何构造返回 employee 表的 name 和 address 列的行的查询。

图: 查询

```
SELECT name, address FROM employee
```

图: 查询结果

```
name          Paul, J.
              address      ROW(102 Ruby, Belmont, CA, 49932, 1000)

              name         Davis, J.
              address      ROW(133 First, San Jose, CA, 85744, 4900)
              :
```

4.1.2 选择包含行类型数据的列

行类型列是在已命名 ROW 类型或未命名 ROW 类型上定义的列。使用相同的 SQL 语法来查询已命名 ROW 类型和未命名行类型列。

对行类型列的查询返回 ROW 类型的所有字段的数据。字段是 ROW 类型中的组件数据类型。例如：employee 表的 address 列包含 street、city 、state 和 zip 字段。下列查询显示如何构造返回 address 列的所有字段的查询。

图: 查询

```
SELECT address FROM employee
```

图: 查询结果

```
address      ROW(102 Ruby, Belmont, CA, 49932, 1000)
             address      ROW(133 First, San Jose, CA, 85744, 4900)
             address      ROW(152 Topaz, Willits, CA, 69445, 1000))
             :
```

要访问列包含的个别字段，使用单个点符号表示法来投影列的个别字段。例如：假设您要访问 employee 表的 address 列中的特定字段。以下 SELECT 语句投影 address 列的 city 和 state 字段。

图: 查询

```
SELECT address.city, address.state FROM employee
```

图: 查询结果

```
city                 state

             Belmont              CA
             San Jose             CA
             Willits              CA
             :
```

用对已命名行类型列构造查询所用的方法来对未命名行类型列构造查询。例如：假设您想要访问图 2中 student 表的 s_address 列的数据。可以使用点符号表示法来查询对未命名行类型定义的列的个别字段。以下查询显示如何对 student 表构造返回 s_address 列的 city 和 state 字段的行的 SELECT 语句。

图: 查询

```
SELECT s_address.city, s_address.state FROM student
```

图: 查询结果

```
city                 state

             Belmont              CA
             Mount Prospect       IL
             Greeley              CO
             :
```

字段投影

不要混淆列和字段。列只与表相关联，并且列投影将格式为 name_1.name2 的常规点符号表示法分别用于表和列。字段是 ROW 类型中的组件数据类型。使用 ROW 类型（和将 ROW 类型指定给单个列的能力），您可以使用形式为 name_a.name_b.name_c.name_d 的单点符

号表示法来投影列的各个字段。GBase 8s 数据库服务器使用以下优先顺序规则来解释点符号表示法：

1. table_name_a . column_name_b . field_name_c . field_name_d
2. column_name_a . field_name_b . field_name_c . field_name_d

当特定标识的意义有歧义时。数据库服务器使用优先顺序规则来确定标识指定哪个数据库对象。考虑以下两个语句：

```
CREATE TABLE b (c ROW(d INTEGER, e CHAR(2)))
        CREATE TABLE c (d INTEGER)
```

在下列 SELECT 语句中，表达式 c.d 引用表 c 的列 d（而不是 b 中列 c 的字段 d），原因是表标识具有比列标识更高的优先顺序：

```
SELECT * FROM b,c WHERE c.d = 10
```

为了避免引用错误的数据库对象，可为字段投影指定完全符号表示法。例如：假定您想要引用 b 中列 c 的字段 d（而不是表 c 的列 d）。以下语句指定想要引用的对象的表、列和字段标识：

```
SELECT * FROM b,c WHERE b.c.d = 10
```

重要： 虽然优先顺序规则减少了数据库服务器误解字段投影的几率，但是建议对所有表、列和字段标识使用唯一名称。

使用字段投影来选择嵌套字段

行类型通常是列，但可将任何行类型表达式用于字段投影。当行类型表达式本身包含其它行类型时，表达式就包含嵌套字段。要访问表达式或个别字段中的嵌套字段，使用点符号表示法。要访问行类型的所有字段，使用星号（*）。本节描述行类型访问的两种方法。

有关如何将点符号表示法和星号符号表示法与行类型表达式配合使用的讨论，请参阅《GBase 8s SQL 指南：语法》中的表达式段。

选择行类型的个别字段

考虑 employee 表的 address 列，它包含字段 street 、city 、state 和 zip。此外，zip 字段包含嵌套字段：z_code 和 z_suffix。（您可能想要复查图 1的行类型和表定义。）对 zip 字段的查询返回 z_code 和 z_suffix 字段的行。但是，可以指定查询只返回特定嵌套字段。以下查询显示如何使用点符号表示法来构造只返回 address 列中 z_code 字段的行的 SELECT 语句。

图: 查询

```
SELECT address.zip.z_code FROM employee
```

图: 查询结果

```
z_code
```

```
39444
6500
76055
19004
⋮
```

使用星号表示法来访问行类型的所有字段

星号符号表示法仅在 SELECT 语句的选择列表中受支持。为投影列表中的行类型列指定列名时，数据库服务器返回列的所有字段的值。想要投影 ROW 类型内的所有字段时，还可以使用星号符号表示法。

下列查询使用星号符号表示法来返回 employee 表中 address 列的所有字段。

图: 查询

```
SELECT address.* FROM employee;
```

图: 查询结果

```
address    ROW(102 Ruby, Belmont, CA, 49932, 1000)
           address    ROW(133 First, San Jose, CA, 85744, 4900)
           address    ROW(152 Topaz, Willits, CA, 69445, 1000))
           ⋮
```

星号符号表示法使得执行某些 SQL 任务更容易。假设您创建返回行类型值的函数 new_row() 并且想要调用此函数并将返回的行插入到表中。数据库服务器没有提供处理此类操作的简便方法。但是，以下查询显示如何使用星号表示法来返回 new_row() 的所有字段并将返回的字段插入到 tab_2表中。

图: 查询

```
INSERT INTO tab_2 SELECT new_row(exp).* FROM tab_1
```

有关如何使用 INSERT 语句的信息，请参阅修改数据。

重要： 只能对使用 .* 符号表示法的表达式求值一次。

4.2 从集合中选择

本节描述如何查询对集合类型定义的列。集合类型是一种复杂 数据类型。其中每个集合值包含具有相同数据类型的一组元素。有关如何访问集合包含的个别元素的信息，请参阅处理 SELECT 语句中的集合。

下图显示 manager 表，在本节的示例中使用了该表。manager 表同时包含简单集合类型和嵌套集合类型。简单集合是一种集合类型。它不包含本身就是集合类型的任何字段。 manager 表的 direct_reports 列就是一个简单集合。嵌套集合是包含另一集合类型的集合类型。manager 表的projects 列就是一个嵌套集合。

图: manager 表

```
CREATE TABLE manager
        (
        mgr_name            VARCHAR(30),
        department          VARCHAR(12),
        direct_reports    SET(VARCHAR(30) NOT NULL),
        projects            LIST(ROW(pro_name VARCHAR(15),
        pro_members SET(VARCHAR(20) NOT NULL)
        ) NOT NULL)
        )
```

对于表中的每一行，对作为集合类型的列的查询返回特定集合包含的所有元素。例如：以下查询显示对 manager 表的每一行返回 department 列中的数据和 direct_reports 列中的所有元素的查询。

图: 查询

```
SELECT department, direct_reports FROM manager
```

图: 查询结果

```
department          marketing
                direct_reports    SET {Smith, Waters, Adams, Davis, Kurasawa}

                department          engineering
                ddirect_reports    SET {Joshi, Davis, Smith, Waters, Fosmire, Evans,
Jones}

                department          publications
                direct_reports    SET {Walker, Fremont, Porat, Johnson}

                department          accounting
                direct_reports    SET {Baker, Freeman, Jacobs}
                ⋮
```

对集合类型查询的输出总是包括类型构造函数，它指定集合是 SET 、MULTISET 或 LIST 。例如：在此结果中，SET 构造函数位于每个集合的元素前面。花括号（{}）划分集合的元素；逗号隔开集合的个别元素。

4.2.1 选择嵌套集合

manager 表的 projects 列是嵌套集合（请参阅图 1）。对嵌套集合类型的查询返回特定集合包含的所有元素。下列查询显示返回 projects 列的特定行中所有元素的查询。WHERE 子句将查询限制为单个行，在其中，mgr_name 列中的值是 Sayles。

图: 查询

```
SELECT projects
        FROM manager
        WHERE mgr_name = 'Sayles'
```

查询结果显示 manager 表的单个行的 project 列集合。查询返回管理 Sayles 描述的那些项目的名称。对于 LIST 中的每个元素，集合包含项目名（pro_name）和指定给每个项目的成员（pro_members）的 SET 。

图: 查询结果

```
projects   LIST {ROW(voyager_project, SET{Simonian, Waters, Adams, Davis})}

        projects   LIST {ROW(horizon_project, SET{Freeman, Jacobs, Walker,
Cannan})}

        projects   LIST {ROW(sapphire_project, SET{Villers, Reeves, Doyle,
Strongin})}
        :
```

4.2.2 使用 IN 关键字来搜索集合中的元素

可以在 SQL 语句的 WHERE 子句中使用 IN 关键字来确定集合是否包含某元素。例如：下列查询显示如何构造返回 mgr_name 和 department 的值的查询，其
中 Adams 是 direct_reports 列中集合的一个元素。

图: 查询

```
SELECT mgr_name, department
        FROM manager
        WHERE 'Adams' IN direct_reports
```

图: 查询结果

```
mgr_name       Sayles
          department     marketing
```

尽管可以使用带有 IN 关键字的 WHERE 子句来搜索简单集合中的特定集合。但是查询总是返回整个集合。例如：下列查询返回集合的所有元素，其中 Adams 是 direct_reports 列中集合的一个元素。

图: 查询

```
SELECT mgr_name, direct_reports
        FROM manager
        WHERE 'Adams' IN direct_reports
```

图: 查询结果

```
mgr_name        Sayles
        direct_reports  SET {Smith, Waters, Adams, Davis, Kurasawa}
```

如上所示，对集合列的查询返回整个集合，不是集合中的特定元素。

可以在 WHERE 子句中使用 IN 关键字来只引用简单集合。不能使用 IN 关键字来引用包含本身就是集合的字段的集合。例如：不能使用 IN 关键字来引用 manager 表中的 projects 列，原因是 projects 是嵌套集合。

可以在 SELECT 语句的 WHERE 子句中组合 NOT 和 IN 关键字来搜索不包含某元素的集合。例如：下列查询显示返回 mgr_name 和 department的值的查询，其中 Adams 不是 direct_reports 列中集合的元素。

图: 查询

```
SELECT mgr_name, department
        FROM manager
        WHERE 'Adams' NOT IN direct_reports
```

图: 查询结果

```
mgr_name      Williams
        department    engineering

        mgr_name      Lyman
        department    publications

        mgr_name      Cole
        department    accounting
```

有关如何对集合列中的元素进行技术的信息，请参阅基数函数。

4.3 选择表层次结构中的行

本节描述如何从表层次结构内的表查询行。

下列查询显示创建本节中的示例使用的类型和表层次结构的语句。

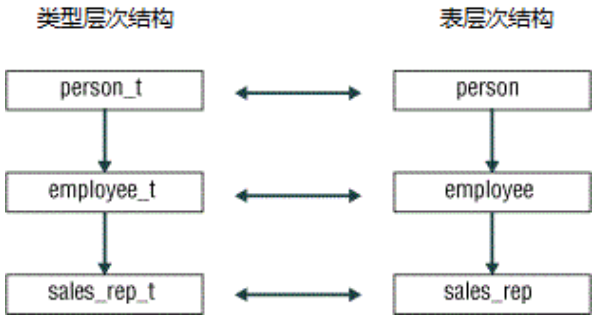
图: 创建类型和表层次结构的语句

```
CREATE ROW TYPE address_t
                (
                street     VARCHAR (20),
                city       VARCHAR(20),
                state      CHAR(2),
                zip        VARCHAR(9)
                )

                CREATE ROW TYPE person_t
                (
                name       VARCHAR(30),
                address    address_t,
                soc_sec    CHAR(9)
                )

                CREATE ROW TYPE employee_t
                (
                salary          INTEGER
                )
                UNDER person_t

                CREATE ROW TYPE sales_rep_t
                (
                rep_num        SERIAL8,
                region_num     INTEGER
                )
                UNDER employee_t

                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
```

下图显示上图中的行类型和表的层次关系。

图: 类型和表层次结构



4.3.1 不使用 ONLY 关键字选择超表的行

表层次结构允许您在单个 SQL 语句中构造其作用域是超表及其子表的查询。对超表的查询同时从超表及其子表中返回行。下列查询显示对 person表（它是表层次结构中的 root 超表）的查询。

图: 查询

```
SELECT * FROM person
```

图 8返回超表中的所有列以及子表（employee 和 sales_rep）中继承自超表的那些列。对超表的查询不返回不在超表中的子表的列。该查询显示person 、employee 和 sales_rep 表中的 name 、address 和 soc_sec 列。

图: 查询结果

```
name          Rogers, J.
address       ROW(102 Ruby Ave, Belmont, CA, 69055)
soc_sec       454849344

name          Sallie, A.
address       ROW(134 Rose St, San Carlos, CA, 69025)
soc_sec       348441214
⋮
```

4.3.2 使用 ONLY 关键字选择超表的行

尽管超表上的 SELECT 语句同时从该超表及其子表中返回行。但是不能辨别哪些行来自超表，哪些行来自子表。要将查询的结果限制为只是超表，必须在 SELECT 语句中包含 ONLY 关键字。例如，下列查询只返回 person 表中的行。

图: 查询

```
SELECT * FROM ONLY(person);
```

图: 查询结果

```
name            Rogers, J.
                address         ROW(102 Ruby Ave, Belmont, CA, 69055)
                soc_sec         454849344
                :
```

4.3.3 对超表使用别名

别名是 FROM 子句中紧跟在表名后面的词，可以在 SELECT 或 UPDATE 语句中为类型表指定别名，然后将别名本身用作表达式。如果为超表创建别名，那么该别名可表示来自该超表或继承自该超表的子表中的值。在 DB-Access 中，下列查询返回 person 、employee 和 sales_rep 表的所有实例的行值。

图: 查询

```
SELECT p FROM person p;
```

GBase 8s ESQL/C 不能识别此构造。在 GBase 8s ESQL/C 程序中，该查询返回一个错误。

4.4 总结

本章介绍了通过使用 SELECT 语句来查询关系数据库以从复杂类型中选择数据的样本语法和结果。选择行类型数据一节显示如何执行下列操作：

- 从类型表和列中选择行类型数据
- 将行类型表达式用于字段投影

从集合中选择一节显示如何执行下列操作：

- 集合类型上定义的查询列
- 查询定义为嵌套集合类型的列
- 在嵌套集合类型上定义的查询列

选择表层次结构中的行一节显示如何执行下列操作：

- 使用或不使用 ONLY 关键字来查询超表
- 为超表指定别名

5 在 SELECT 语句中使用函数

除了列名和运算符之外，表达式还可包括一个或多个函数。本章说明如何在 SELECT 语句中使用函数来执行更复杂的数据库查询和数据处理。

有关下列 SQL 函数以及其它 SQL 函数的语法的信息，请参阅《GBase 8s SQL 指南：语法》中表达式段。

提示： 还可以使用您自己创建的函数。有关用户定义函数的信息，请参阅创建和使用 SPL 例程和《GBase 8s 用户定义的例程和数据类型开发者指南》。

5.1 在 SELECT 语句中使用函数

可以在选择列表中使用任何基本类型的表达式（列、常量、函数、聚集函数和过程）或它们的组合。

函数表达式使用对查询中的每一行进行求值的函数。所有函数表达式都需要参数。当在列名用作参数的情况下使用这一组表达式时，这些表达式包含时间函数和长度函数。

5.1.1 聚集函数

聚集函数对一组查询返回一个值。聚集函数取用依赖于 SELECT 语句的 WHERE 子句返回的一组行的值。没有 WHERE 子句时，聚集函数取用依赖于 FROM 子句组成的所有行的值。

不能将聚集函数用于包含下列数据类型的表达式中：

- TEXT
- BYTE
- CLOB
- BLOB
- 集合数据类型（LIST 、MULTISET 和 SET）
- ROW 类型
- 不透明数据类型（支持不透明数据类型的用户定义的聚集函数除外）

聚集通常用于总结有关表中的行组的信息。此用法在编写高级 SELECT 语句中讨论。当将聚集函数应用于整个表时，结果将包含总结所有选择的行的一行。

所有的 GBase 8s 数据库服务器都支持下列聚集函数。

AVG 函数

下列查询计算 stock 表中所有行的平均值 unit_price。

图: 查询

```
SELECT AVG (unit_price) FROM stock;
```

图: 查询结果

```
(avg)

        $197.14
```

下列查询只计算 **stock** 表中 **manu_code** 为 SHM 的那些行的平均值 **unit_price。**

图: 查询

```
SELECT AVG (unit_price) FROM stock WHERE manu_code = 'SHM';
```

图: 查询结果

```
(avg)

        $204.93
```

COUNT 函数

下列查询对 stock 表中的总行数进行计数和显示。

图: 查询

```
SELECT COUNT(*) FROM stock;
```

图: 查询结果

```
(count(*))

          73
```

下列查询包含 WHERE 子句来对 stock 表中的特定行（在此示例中，
是 manu_code 为 SHM 的那些行）进行计数。

图: 查询

```
SELECT COUNT (*) FROM stock WHERE manu_code = 'SHM';
```

图: 查询结果

```
(count(*))

        17
```

通过包含 DISTINCT 关键字（或它的同义词 UNIQUE）和列名，可以计算 stock 表中不同制造商代码的数目。

图: 查询

```
SELECT COUNT (DISTINCT manu_code) FROM stock;
```

图: 查询结果

```
(count)

            9
```

MAX 和 MIN 函数

可以在同一 SELECT 语句中组合聚集函数。例如，可以同时在选择列表中包括 MAX 和 MIN 函数。如下所示。

图: 查询

```
SELECT MAX (ship_charge), MIN (ship_charge) FROM orders;
```

该查询查找并显示 orders 表中的最大和最小 ship_charge。

图: 查询结果

```
(max)       (min)

            $25.20        $5.00
```

RANGE 函数

RANGE 函数计算所选行的最大值与最小值之差。

只能将 RANGE 函数应用于数字列。下列查询查找 stock 表中商品的价格范围。

图: 查询

```
SELECT RANGE(unit_price) FROM stock;
```

图: 查询结果

```
(range)

            955.50
```

对于其它聚集函数，当查询包括 GROUP BY 子句时，RANGE 函数适用于组的行，如下所示。

图: 查询

```
SELECT RANGE(unit_price) FROM stock
```

```
        GROUP BY manu_code;
```

图: 查询结果

```
(range)

          820.20
          595.50
          720.00
          225.00
          632.50
          0.00
          460.00
          645.90
          425.00
```

STDEV 函数

STDEV 函数计算所选行的标准偏差。它是 VARIANCE 函数的平方根。.

可将 STDEV 函数应用于数字列。下列查找入口的标准偏差：

```
SELECT STDEV(age) FROM u_pop WHERE age > 21;
```

对于其它聚集，当查询包括 GROUP BY 子句时，STDEV 函数适用于组的行。如下所示：

```
SELECT STDEV(age) FROM u_pop
        GROUP BY state
        WHERE STDEV(age) > 21;
```

除非指定列中的每个值都是空值，否则会忽略空值。如果每个列值都是空值，那么 STDEV 函数对该列返回空值。有关 STDEV 函数的更多信息，请参阅《GBase 8s SQL 指南：语法》中的表达式段。

SUM 函数

下列查询计算 1998 年 7 月 13 日交付的所有订单的总 ship_weight。

图: 查询

```
SELECT SUM (ship_weight) FROM orders
        WHERE ship_date = '07/13/1998';
```

图: 查询结果

```
(sum)
```

```
                130.5
```

VARIANCE 函数

VARIANCE 函数返回值样本的方差作为所有选择行的方差的无偏估计。它计算以下值：

```
(SUM(Xi**2) - (SUM(Xi)**2)/N)/(N-1)
```

在此示例中，Xi 是列中的每个值，N 是列中值的总数。只能将 VARIANCE 函数应用于数字列。以下查询查找入口的标准偏差：

```
SELECT VARIANCE(age) FROM u_pop WHERE age > 21;
```

对于其它聚集，当查询包括 GROUP BY 子句时，VARIANCE 函数适用于组的行。如下所示：

```
SELECT VARIANCE(age) FROM u_pop
        GROUP BY birth
        WHERE VARIANCE(age) > 21;
```

除非指定列中的每个值都是空值，否则会忽略空值。如果每个列值都是空值，那么 VARIANCE 函数对该列返回空值。有关 VARIANCE 函数的更多信息，请参阅《GBase 8s SQL 指南：语法》中的表达式段。

将函数应用于表达式

下列查询显示如何将函数应用于算术表达式并为其结果提供显示标签：

图: 查询

```
SELECT MAX (res_dtime - call_dtime) maximum,
        MIN (res_dtime - call_dtime) minimum,
        AVG (res_dtime - call_dtime) average
        FROM cust_calls;
```

该查询查找和显示收到和处理客户来电之间的最长、最短和平均时间（以日、小时和分钟计）并相应地标记派生值。该查询结果显示这些时间量。

图: 查询结果

```
maximum         minimum         average

        5 20:55         0 00:01         1 02:56
```

5.1.2 时间函数

可以在查询的 Projection 子句或 WHERE 子句中使用时间函数 DAY 、MONTH 、WEEKDAY 和 YEAR 。这些函数返回与用来调用函数的表达式或参数对应的值。还可以使用 CURRENT 或 SYSDATE 函数返回具有当前日期和时间的值，或者使用 EXTEND 函数调整 DATE 或 DATETIME 值。

DAY 和 CURRENT 函数

下列查询在两个 expression 列中对 call_dtime 和 res_dtime 列返回日期（一个月中的某一天）。

图: 查询

```
SELECT customer_num, DAY (call_dtime), DAY (res_dtime)
        FROM cust_calls;
```

图: 查询结果

```
customer_num   (expression) (expression)

        106            12           12
        110             7            7
        119             1            2
        121            10           10
        127            31
        116            28           28
        116            21           27
```

下列查询使用 DAY 和 CURRENT 函数来将列值与当前日期（月中某日）进行比较。它只选择值比当前日期早的那些行。在此示例中，CURRENT 日是 15。

图: 查询

```
SELECT customer_num, DAY (call_dtime), DAY (res_dtime)
        FROM cust_calls
        WHERE DAY (call_dtime) < DAY (CURRENT);
```

图: 查询结果

```
customer_num   (expression) (expression)
        106            12           12
        110             7            7
        119             1            2
        121            10           10
```

下列查询使用 CURRENT 函数来选择除今天打的电话之外的所有来电。

图: 查询

```
SELECT customer_num, call_code, call_descr
        FROM cust_calls
        WHERE call_dtime < CURRENT YEAR TO DAY;
```

图: 查询结果

```
customer_num    106
        call_code       D
        call_descr      Order was received, but two of the cans of ANZ tennis
balls
        within the case were empty

        customer_num    110
        call_code       L
        call_descr       Order placed one month ago (6/7) not received.
        ⋮
        customer_num    116
        call_code       I
        call_descr      Second complaint from this customer! Received two cases
        right-handed outfielder gloves (1 HRO) instead of one case
        lefties.
```

SYSDATE 函数与 CURRENT 函数及其类似，但当未指定 DATETIME 限定符时，其返回值的缺省精度是 DATETIME YEAR TO FRACTION(5)，而不是 CURRENT 的缺省精度 DATETIME YEAR TO FRACTION(3)。

MONTH 函数

下列查询使用 MONTH 函数来抽取和显示在哪个月份接收和处理客户来电，并且它将对结果列使用显式标签。但是，它不区分年份。

图: 查询

```
SELECT customer_num,
        MONTH (call_dtime) call_month,
        MONTH (res_dtime) res_month
        FROM cust_calls;
```

图: 查询结果

```
customer_num    call_month      res_month

        106             6               6
        110             7               7
```

```
119             7             7
121             7             7
127             7
116            11            11
116            12            12
```

如果 DAY 比当前日期早，那么下列查询使用 MONTH 函数和 DAY 及 CURRENT 来显示在哪个月份接收和处理客户来电。

图: 查询

```
SELECT customer_num,
        MONTH (call_dtime) called,
        MONTH (res_dtime) resolved
        FROM cust_calls
        WHERE DAY (res_dtime) < DAY (CURRENT);
```

图: 查询结果

```
customer_num       called    resolved

         106            6           6
         119            7           7
         121            7           7
```

WEEKDAY 函数

下列查询使用 WEEKDAY 函数来指示在星期几接收并处理来电（0 表示星期日，1 表示星期一，以此类推），并标记表达式列。

图: 查询

```
SELECT customer_num,
        WEEKDAY (call_dtime) called,
        WEEKDAY (res_dtime) resolved
        FROM cust_calls
        ORDER BY resolved;
```

图: 查询结果

```
customer_num       called    resolved

         127            3
         110            0           0
         119            1           2
```

```
121         3         3
116         3         3
106         3         3
116         5         4
```

下列查询使用 COUNT 和 WEEKDAY 函数来对在周末收到的来电进行计数。此类语句能够使您了解客户来电模式或指示是否需要加班费。

图: 查询

```
SELECT COUNT(*)
        FROM cust_calls
        WHERE WEEKDAY (call_dtime) IN (0,6);
```

图: 查询结果

```
(count(*))

          4
```

YEAR 函数

下列查询检索 call_dtime 比当前年份的开始早的行。

图: 程序

```
SELECT customer_num, call_code,
        YEAR (call_dtime) call_year,
        YEAR (res_dtime) res_year
        FROM cust_calls
        WHERE YEAR (call_dtime) < YEAR (TODAY);
```

图: 查询结果

```
customer_num call_code call_year res_year

         116 I              1997      1997
         116 I              1997      1997
```

格式化 DATETIME 值

在下列查询中，EXTEND 函数仅显示指定的子字段以限制两个 DATETIME 值。

图: 程序

```
SELECT customer_num,
        EXTEND (call_dtime, month to minute) call_time,
```

```
        EXTEND (res_dtime, month to minute) res_time
        FROM cust_calls
        ORDER BY res_time;
```

该查询为标签为 call_time 和 res_time 的列返回月份至分钟范围，提供工作量的指示。

图: 查询结果

```
customer_num call_time      res_time

        127 07-31 14:30
        106 06-12 08:20 06-12 08:25
        119 07-01 15:00 07-02 08:21
        110 07-07 10:24 07-07 10:30
        121 07-10 14:05 07-10 14:06
        116 11-28 13:34 11-28 16:47
        116 12-21 11:24 12-27 08:19
```

TO_CHAR 函数也可以格式化 DATETIME 值。有关内置函数的信息，请参阅 TO_CHAR 函数，该函数也可接受作为参数的 DATE 值或数字值并返回格式化的字符串。

除了这些示例说明的内置时间函数之外，GBase 8s 还支持 ADD_MONTHS 、LAST_DAY 、MDY 、MONTHS_BETWEEN 、NEXT_DAY 和QUARTER 函数。除了这些函数，TRUNC 和 ROUND 函数也可返回更改 DATE 或 DATETIME 参数精度的值。这些附加时间函数在 GBase 8s SQL 指南：语法中进行了描述。

5.1.3 数据转换函数

可以在使用表达式的任何地方使用数据转换函数。

下列转换函数在日期与字符串之间转换：

DATE 函数

DATE 函数将字符串转换为 DATE 值。在以下查询中，DATE 函数将字符串转换为 DATE 值，以允许与 DATETIME 值进行比较。仅当 call_dtime值比指定的 DATE 晚时查询才会检索 DATETIME 值。

图: 查询

```
SELECT customer_num, call_dtime, res_dtime
        FROM cust_calls
        WHERE call_dtime > DATE ('12/31/97');
```

图: 查询结果

```
customer_num call_dtime          res_dtime

        106 1998-06-12 08:20 1998-06-12 08:25
        110 1998-07-07 10:24 1998-07-07 10:30
        119 1998-07-01 15:00 1998-07-02 08:21
        121 1998-07-10 14:05 1998-07-10 14:06
        127 1998-07-31 14:30
```

仅当 call_dtime 大于或等于指定日期时，下列查询才会将 DATETIME 值转换为 DATE 格式并带标签显示这些值。

图: 查询

```
SELECT customer_num,
        DATE (call_dtime) called,
        DATE (res_dtime) resolved
        FROM cust_calls
        WHERE call_dtime >= DATE ('1/1/98');
```

图: 查询结果

```
customer_num called          resolved

        106 06/12/1998    06/12/1998
        110 07/07/1998    07/07/1998
        119 07/01/1998    07/02/1998
        121 07/10/1998    07/10/1998
        127 07/31/1998
```

TO_CHAR 函数

TO_CHAR 函数将 DATETIME 或 DATE 值转换为字符串值。TO_CHAR 函数根据您指定的日期格式化伪指令对 DATETIME 值进行求值并返回 NVARCHAR 值。有关受支持的日期格式化伪指令的列表，请参阅《GBase 8s GLS 用户指南》 GL_DATETIME 环境变量的描述。

还可以使用 TO_CHAR 函数将 DATETIME 或 DATE 值转换为 LVARCHAR 值。

下列查询使用 TO_CHAR 函数将 DATETIME 值转换为可读性更强的字符串。

图: 查询

```
SELECT customer_num,
        TO_CHAR(call_dtime, "%A %B %d %Y") call_date
        FROM cust_calls
```

```
        WHERE call_code = "B";
```

图: 查询结果

```
customer_num    119
        call_date       Friday July 01 1998
```

下列查询使用 TO_CHAR 函数将 DATE 值转换为可读性更强的字符串。

图: 查询

```
SELECT order_num,
        TO_CHAR(ship_date,"%A %B %d %Y") date_shipped
        FROM orders
        WHERE paid_date IS NULL;
```

图: 查询结果

```
order_num               1004
        date_shipped            Monday May 30 1998

        order_num               1006
        date_shipped

        order_num               1007
        date_shipped            Sunday June 05 1998

        order_num               1012
        date_shipped            Wednesday June 29 1998

        order_num               1016
        date_shipped            Tuesday July 12 1998

        order_num               1017
        date_shipped            Wednesday July 13 1998
```

TO_CHAR 函数还可以格式化数字值。有关内置 TO_CHAR 函数的更多信息，请参阅《GBase 8s SQL 指南：语法》。

TO_DATE 函数

TO_DATE 函数接受字符数据类型的参数并将此值转换为 DATETIME 值。TO_DATE 函数根据您指定的日期格式化伪指令对字符串求值并返回 DATETIME 值。有关受支持的日期

格式化伪指令的列表，请参阅《GBase 8s GLS 用户指南》中 GL_DATETIME 环境变量的描述。

还可以使用 TO_DATE 函数将 LVARCHAR 值转换为 DATETIME 值。

下列查询使用 TO_DATE 函数将字符串转换为指定格式的 DATETIME 值。

图: 查询

```
SELECT customer_num, call_descr
        FROM cust_calls
        WHERE call_dtime = TO_DATE("2008-07-07 10:24",
        "%Y-%m-%d %H:%M");
```

图: 查询结果

```
customer_num    110
call_descr      Order placed one month ago (6/7) not received.
```

可以使用 DATE 或 TO_DATE 函数来将字符串转换为 DATE 值。TO_DATE 函数的一个优点是它允许您为返回的值指定格式。（可以使用 TO_DATE函数（它总是返回 DATETIME 值）来将字符串转换为 DATE 值，原因是数据库服务器隐式处理 DATE 和 DATETIME 值之间的转换。）

5.1.4 基数函数

CARDINALITY 函数对集合包含的元素数目计数。可以将 CARDINALITY 函数与简单或嵌套集合配合使用。将集合中的任何重复作为个别元素计数。下列查询显示一个查询，它对 manager 表中的每一列返回 department 值和每个 direct_reports 集合中的元素数。

图: 查询

```
SELECT department, CARDINALITY(direct_reports) FROM manager;
```

图: 查询结果

```
department    marketing 5
department    engineering 7
department    publications 4
department    accounting 3
```

还可以从谓词表达式中对集合的元素数进行求值，如下所示。

```
SELECT department, CARDINALITY(direct_reports) FROM manager
        WHERE CARDINALITY(direct_reports) < 6
        GROUP BY department;
```

图: 查询结果

```
department    accounting 3

              department    marketing 5

              department    publications 4
```

5.1.5 智能大对象函数

数据库服务器提供了四个 SQL 函数,您可以从 SQL 语句中调用这些函数来导入和导出智能大对象。下表显示智能大对象函数。

表 1. 智能大对象的 SQL 函数

函数名称	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 数据复制的另一个 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《GBase 8s SQL 指南：语法》中的表达式段。

可以在 SELECT UPDATE 和 INSERT 语句中使用该表显示的任何函数。有关如何在 INSERT 和 UPDATE 语句中使用上述函数的示例，请参阅修改数据。

假设您创建 inmate 和 fbi_list 表，如下图所示。

图: 创建 inmate 和 fbi_list 表

```
CREATE TABLE inmate
        (
        id_num     INT,
        picture    BLOB,
        felony     CLOB
        );
```

```
CREATE TABLE fbi_list
(
id          INTEGER,
mugshot    BLOB
) PUT mugshot IN (sbspace1);
```

以下 SELECT 语句使用 LOTOFILE() 函数将数据从 felony 列复制到位于客户机上的 felon_322.txt 文件中：

```
SELECT id_num, LOTOFILE(felony, 'felon_322.txt', 'client')
        FROM inmate
        WHERE id = 322;
```

LOTOFILE() 的第一个参数指定将从中导出数据的列的名称。第二个参数指定要将数据复制到其中的文件的名称。第三个参数指定目标文件是位于客户端计算机（'client'）或服务器计算机（'server'）上。

根据源文件是驻留在客户机还是服务器计算机上，下列规则可用来指定函数参数中文件名的路径：

如果源文件驻留在服务器计算机上，那么必须指定文件的全路径名（不是与当前工作目录相对的路径名）。

如果源文件驻留在客户端计算机上，那么可以指定文件的全路径名或相对路径名。

5.1.6 字符串处理函数

字符串处理函数接受类型为 CHAR 、NCHAR 、VARCHAR 、NVARCHAR 或 LVARCHAR 的参数。可以在使用表达式的任何地方使用字符串处理函数。

下列函数在字符串中进行大写字母和小写字母之间的转换：

- LOWER
- UPPER
- INITCAP

下列函数以各种方法处理字符串：

- REPLACE
- SUBSTR
- SUBSTRING
- LPAD
- RPAD

限制： 不能过载任何字符串处理函数来处理扩展数据类型。

LOWER 函数

使用 LOWER 函数来将字符串中的每个大写字母替换为小写字母。LOWER 函数接受字符数据类型的参数并返回具有与指定的参数相同数据类型的值。

下列函数使用 LOWER 函数来将字符串的任何大写字母转换为小写字母。

图: 查询

```
SELECT manu_code, LOWER(manu_code)
        FROM items
        WHERE order_num = 1018
```

图: 查询结果

```
manu_code    (expression)

             PRC            prc
             KAR            kar
             PRC            prc
             SMT            smt
             HRO            hro
```

UPPER 函数

使用 UPPER 函数来将字符串中的每个小写字母替换为大写字母。UPPER 函数接受字符数据类型的参数并返回具有与指定的参数相同数据类型的值。

下列查询中 UPPER 函数将字符串中的任何小写字母转换为大写字母。

图: 查询

```
SELECT call_code, UPPER(code_descr) FROM call_type
```

图: 查询结果

```
call_code     (expression)

          B              BILLING ERROR
          D              DAMAGED GOODS
          I              INCORRECT MERCHANDISE SENT
          L              LATE SHIPMENT
          O              OTHER
```

INITCAP 函数

使用 INITCAP 函数来将字符串中每个词的首字母替换为大写字母。每当函数遇到字母之前是非字母字符时，INITCAP 函数就会假设是一个新词。INITCAP 函数接受字符数据类型的参数并返回指定参数相同数据类型的值。

下列查询使用 INITCAP 函数将字符串中每个词的首字母替换为大写字母。

图: 查询

```
SELECT INITCAP(description) FROM stock
        WHERE manu_code = "ANZ";
```

图: 查询结果

```
(expression)

        Tennis Racquet
        Tennis Ball
        Volleyball
        Volleyball Net
        Helmet
        Golf Shoes
        3 Golf Balls
        Running Shoes
        Watch
        Kick Board
        Swim Cap
```

REPLACE 函数

使用 REPLACE 函数来将字符串中的某一组字符替换为其它字符。

在以下查询中，REPLACE 函数将单元列值 each 替换为查询返回的每一行的 item。

REPLACE 函数的第一个参数是要进行求值的表达式。第二个参数指定想要替换的字符。第三个参数指定要替换除去的字符的新字符串。

图: 查询

```
SELECT stock_num, REPLACE(unit,"each", "item") cost_per, unit_price
        FROM stock
        WHERE manu_code = "HRO";
```

图: 查询结果

```
stock_num       cost_per    unit_price
```

1	case	$250.00
2	case	$126.00
4	case	$480.00
7	case	$600.00
110	case	$260.00
205	case	$312.00
301	item	$42.50
302	item	$4.50
304	box	$280.00
305	case	$48.00
309	case	$40.00
312	box	$72.00

SUBSTRING 和 SUBSTR 函数

可以使用 SUBSTRING 和 SUBSTR 函数返回部分字符串。指定开始位置和长度（可选）来确定函数返回字符串的哪部分。

限制： 这两个函数在参数中的使用测量单位是字节，而非逻辑字符。这在缺省语言环境和另一个单字节语言环境中都不重要，但是您不能在代码集与其存储长度不同的逻辑字符的语言环境中调用 SUBSTRING 或 SUBSTR。

SUBSTRING 函数

可以使用 SUBSTRING 函数来返回字符串的某部分。指定开始位置和长度（可选）来确定返回字符串的哪部分。可以指定正数或负数作为开始位置。开始位置 1 指定 SUBSTRING 函数从字符串的第一个位置开始。当开始位置为（0）或负数时，SUBSTRING 函数从字符串的开头开始向后计数。

下列查询显示 SUBSTRING 函数的一个示例，其返还查询返回的任何 sname 列值的前四个字符。在本示例中，SUBSTRING 函数从字符串的开头开始，返回开始位置开始的四个字符。

图: 查询

```
SELECT sname, SUBSTRING(sname FROM 1 FOR 4) FROM state
        WHERE code = "AZ";
```

图: 查询结果

```
sname           (expression)

Arizona         Ariz
```

在下列查询中，SUBSTRING 函数指定开始位置 6，但未指定长度。函数返回从字符串的第六个位置开始到字符串结尾的字符串。

图: 查询

```
SELECT sname, SUBSTRING(sname FROM 6) FROM state
        WHERE code = "WV";
```

图: 查询结果

```
sname                (expression)

          West Virginia      Virginia
```

在下列查询中，SUBSTRING 函数只返回查询返回的任何 sname 列值的第一个字符。对于 SUBSTRING 函数，开始位置 -2 表示从字符的开始位置向后数三个位置（0 、-1 、-2）（对于开始位置 0，函数从字符串的开始位置向后数一个位置）。

图: 查询

```
SELECT sname, SUBSTRING(sname FROM -2 FOR 4) FROM state
        WHERE code = "AZ";
```

图: 查询结果

```
sname                (expression)

          Arizona            A
```

SUBSTR 函数

SUBSTR 函数的作用于 SUBSTRING 函数相同，但两个函数的语法有区别。

要返回字符串的一部分，指定开始位置和长度（可选）来确定 SUBSTR 函数返回子串的哪个部分。为 SUBSTR 函数指定的开始位置可以是正数，也可以是负数。然而，SUBSTR 函数用与 SUBSTRING 函数不同的方式处理开始位置中的负数。当开始位置是负数时，SUBSTR 函数从字符串的末尾开始向后计数，这取决于字符串的长度，而不是字符串包含词或可视字符的长度。SUBSTR 函数将开始位置中的零（0）或 1 识别为字符串中的第一个位置。

下列查询显示包括负数作为开始位置的 SUBSTR 函数的一个示例。假定开始位置为 -15，那么 SUBSTR 函数从字符串末尾开始向后数 15 个位置来找到开始位置，然后返回下五个字符。

图: 查询

```
SELECT sname, SUBSTR(sname, -15, 5) FROM state
        WHERE code = "CA";
```

图: 查询结果

```
sname                (expression)


                California          Calif
```

要使用负数作为开始位置，需要指定求出的长度值。sname 列被定义为 CHAR(15)，因此接受类型为 sname 的参数的 SUBSTR 函数可以将开始位置 0 、1 或 -15 用于函数来返回从字符串的第一个位置开始的字符串。

下列查询返回与图 1相同的结果。

图: 查询

```
SELECT sname, SUBSTR(sname, 1, 5) FROM state
        WHERE code = "CA";
```

LPAD 函数

使用 LPAD 函数返回已用重复次数达到必要次数的字符序列在左边填充或截断的字符串的副本，这取决于字符串中填充部分的指定长度。指定源字符串、要返回的字符串的长度和要用来填充的字符串。

源字符串和用来填充的字符串的数据类型可以是能转换为 VARCHAR 或 NVARCHAR 的任何数据类型。

下列查询显示具有指定长度 21 个字节的 LPAD 函数的一个示例。由于源字符串长度为 15 个字节（sname 被定义为 CHAR(15)），所以 LPAD 函数填充字符串左边的前六个位置。

图: 查询

```
SELECT sname, LPAD(sname, 21, "-")
            FROM state
            WHERE code = "CA" OR code = "AZ";
```

图: 查询结果

```
sname                (expression)


                California          ------California
                Arizona              ------Arizona
```

RPAD 函数

所以 RPAD 函数返回已用重复次数达到必要次数的字符序列在右边填充或截断的字符串的副本，这取决于字符串中填充部分的指定长度。指定源字符串、要返回的字符串的长度和要用来填充的字符串。

源字符串和用来填充的字符串的数据类型可以是能转换为 VARCHAR 或 NVARCHAR 的任何数据类型。

下列查询显示具有指定长度 21 个字节的 RPAD 函数的一个示例。由于源字符串长度为 15 个字节（sname 被定义为 CHAR(15)），所以 LPAD 函数填充字符串右边的前六个位置。

图: 查询

```
SELECT sname, RPAD(sname, 21, "-")
    FROM state
    WHERE code = "WV" OR code = "AZ";
```

图: 查询结果

```
sname                (expression)
West Virginia     West Virginia    ------
Arizona              Arizona             ------
```

除了这些函数之外，LTRIM 和 RTRIM 函数可以返回删除其字符串参数中指定前导或尾随填充字符的值，并且 ASCII 函数可以返回在其字符串参数中第一个字符的 ASCII 字符集中代码点的数字值。这些内置函数对字符串值的操作在 GBase 8s SQL 指南：语法中进行了描述。

5.1.7 其它函数

还可以在使用常量的 SQL 表达式中的任意位置使用 LENGTH 、USER 、CURRENT 、SYSDATE 和 TODAY 函数。另外，可以在 SELECT 语句中包括 DBSERVERNAME 函数来显示当前数据库所驻留的数据库服务器的名称。

还可以使用这些函数来选择全部由常量值组成的表达式或包括列数据的表达式。在一个实例中，对于所有输出行，结果相同。

另外，可以使用 HEX 函数返回表达式的十六进制编码，使用 ROUND 函数来返回表达式的四舍五入值，使用 TRUNC 函数来返回表达式的截断值。有关上述函数的更多信息，请参阅《GBase 8s SQL 指南：语法》。

LENGTH 函数

在下列查询中，LENGTH 函数针对 company 的长度大于 15 的每个行计算组合 fname 和 lname 列的字节数。

图: 查询

```
SELECT customer_num,
            LENGTH (fname) + LENGTH (lname) namelength
            FROM customer
```

```
        WHERE LENGTH (company) > 15;
```

图: 查询结果

customer_num	namelength
101	11
105	13
107	11
112	14
115	11
118	10
119	10
120	10
122	12
124	11
125	10
126	12
127	10
128	11

尽管 LENGTH 函数在使用 DB-Access 时可能不是非常有用，但用于确定程序和报告的长度时它就非常重要。LENGTH 函数返回 CHARACTER 或 VARCHAR 字符串的剪切长度以及 TEXT 或 BYTE 字符串中的全部字节数。

GBase 8s 还支持 CHAR_LENGTH 函数，该函数在其字符串参数中返回逻辑字符数而不是返回字节数。该函数在单个逻辑字符可能需要多个单字节存储的语言环境中非常有用。有关 CHAR_LENGTH 函数的更多信息，请参阅《GBase 8s SQL 指南：语法》和《GBase 8s GLS 用户指南》。

USER 函数

当想要定义仅包含包括您的用户标识行的表的受限视图时，使用 USER 函数。有关如何创建视图的信息，请参阅《 GBase 8s SQL 指南：语法》中的 GRANT 和 CREATE VIEW 语句。

下列查询返回执行查询的用户的用户名（登录用户名），对表中的每行重复一次。

图: 查询

```
SELECT * FROM cust_calls
        WHERE user_id = USER;
```

如果当前用户的用户名是 richc，该查询仅检索 cust_calls 表中 user_id = richc 的行。

图: 查询结果

```
customer_num    110
            call_dtime      1998-07-07 10:24
            user_id          richc
            call_code        L
            call_descr      Order placed one month ago (6/7) not received.
            res_dtime        1998-07-07 10:30
            res_descr        Checked with shipping (Ed Smith). Order sent yesterday-we
            were waiting for goods from ANZ. Next time will call with
            delay if necessary

            customer_num    119
            call_dtime      1998-07-01 15:00
            user_id          richc
            call_code        B
            call_descr      Bill does not reflect credit from previous order
            res_dtime        1998-07-02 08:21
            res_descr        Spoke with Jane Akant in Finance. She found the error and
is
            sending new bill to customer
```

TODAY 函数

TODAY 函数返回当前系统日期。如果下列查询是在当前系统日期为 1998 年 7 月 10 日时发出的，它返回这一行。

图: 查询

```
SELECT * FROM orders WHERE order_date = TODAY;
```

图: 查询结果

```
order_num         1018
            order_date        07/10/1998
            customer_num      121
            ship_instruct     SW corner of Biltmore Mall
            backlog            n
            po_num              S22942
            ship_date         07/13/1998
            ship_weight       70.50
```

```
        ship_charge     $20.00
        paid_date       08/06/1998
```

DBSERVERNAME 和 SITENAME 函数

可以在 SELECT 语句中包含 DBSERVERNAME（或它的同义词 SITENAME）函数来查询数据库服务器的名称。可以查询 DBSERVERNAME 以找到具有行的任何表，包括系统目录表。

在下列查询中，将标签 server 指定给 DBSERVERNAME 表达式并且也从 systables 系统目录表中选择 tabid 列。此表描述数据库表，tabid 就是表标识。

图: 查询

```
SELECT DBSERVERNAME server, tabid
        FROM systables
        WHERE tabid <= 4;
```

图: 查询结果

```
server          tabid

        montague        1
        montague        2
        montague        3
        montague        4
```

WHERE 子句限制显示的行数。否则，可能会对 systables 表的每一行显示数据库服务器名一次。

HEX 函数

在下列查询中，HEX 函数返回 customer 中两列的十六进制格式，如下所示。

图: 查询

```
SELECT HEX (customer_num) hexnum, HEX (zipcode) hexzip
        FROM customer;
```

图: 查询结果

```
hexnum      hexzip

        0x00000065 0x00016F86
        0x00000066 0x00016FA5
        0x00000067 0x0001705F
```

```
0x00000068 0x00016F4A
0x00000069 0x00016F46
0x0000006A 0x00016F6F
⋮
```

DBINFO 函数

可以在 SELECT 与中调用 DBINFO 函数来查询下列任何信息：

- 与 tblspace 号或表达式对应的 dbspace 的名称
- 表中插入的最后一个 SERIAL 、SERIAL8 或 BIGSERIAL 值
- SELECT 、INSERT 、DELETE 、UPDATE 、MERGE 、EXECUTE FUNCTION 、EXECUTE PROCEDURE 或 EXECUTE ROUTINE 语句处理的行数
- 当前会话的会话 ID
- 会话连接的当前的数据库的名称
- INSERT 、UPDATE 或 DELETE 语句是否作为应答事务一部分正在执行
- 数据库服务器在其上运行的主计算机的名称
- 操作系统的类型和主计算机的名称
- 全球标准时间（UTC）格式的本地时区和当前日期和时间
- 对应于指定的整型列或指定的 UTC 时间值的 DATETIME 值（作为自 1970-01-01 00:00:00+00:00 的秒数）
- 客户机应用程序连接至的数据库服务器的精确版本或指定完整版本字符串的组件

可以在 SQL 语句中和 SPL 例程中的任何地方使用 DBINFO 函数。

下列查询显示可以和如何使用 DBINFO 函数来找出数据库服务器在其上运行的主计算机的名称。

图: 查询

```
SELECT FIRST 1 DBINFO('dbhostname') FROM systables;
```

图: 查询结果

```
(constant)

            lyceum
```

没有 FIRST 1 子句来限制 tabid 中的值，将对 systables 表的每一行重复数据库服务器在其上运行的计算机的主机名。下列查询显示可以如何使用 DBINFO 函数来找出当前数据库服务器的完整版号和类型。

图: 查询

```
SELECT FIRST 1 DBINFO('version','full') FROM systables;
```

有关如何使用 DBINFO 函数查找您当前数据库服务器、数据库会话或数据库的信息的更多信息，请参阅《GBase 8s SQL 指南：语法》。

DECODE 函数

可以使用 DECODE 函数来将具有一个值的表达式转换为另一个值。DECODE 函数具有以下格式：

```
DECODE(test, a, a_value, b, b_value, ..., n, n_value, exp_m )
```

在通常情况下，当 a 等于 test 时 DECODE 函数返回 a_value，当 b 等于 test 时，返回 b_value，当 n 等于 test 时返回 n_value 。

如果有若干表达式与 test 匹配，那么 DECODE 返回找到的第一个表达式的 n_value。如果没有表达式与 test 匹配，那么 DECODE 返回 exp_m;；如果没有表达式与 test 匹配并且不存在 exp_m，那么 DECODE 返回 NULL。

限制： DECODE 函数不支持类型为 TEXT 或 BYTE 的参数。

假设包括 emp_id 和 evaluation 列的 employee 表存在。此外还假设对 employee 表执行下列查询则返回以下所示的行。

图: 查询

```
SELECT emp_id, evaluation FROM employee;
```

图: 查询结果

```
emp_id          evaluation

        012233          great
        012344          poor
        012677          NULL
        012288          good
        012555          very good
```

在某些情况下，您可能想要转换一组值。例如：假设您想要将前一示例中 evaluation 列的描述值转换为相应的数字值。下列查询显示如何使用DECODE 函数来针对 employee 表中的每一行将 evaluation 列中的值转换为数字值。

图: 查询

```
SELECT emp_id, DECODE(evaluation, "poor", 0, "fair", 25, "good",
            50, "very good", 75, "great", 100, -1) AS evaluation
            FROM employee;
```

图: 查询结果

```
emp_id          evaluation

        012233          100
        012344          0
        012677          -1
        012288          50
        012555          75
        :
```

可为 DECODE 函数的参数指定任何数据类型，只要这些参数满足以下需求：

参数 test 、a 、b 、... 、n 都具有相同的数据类型或求值为公共兼容的数据类型。

参数 a_value 、b_value 、... 、n_value 都具有相同的数据类型或求值为公共兼容的数据类型。

NVL 函数

可以使用 NVL 函数将求值为 NULL 的表达式转换为您指定的值。NVL 函数接受两个参数：第一个参数获取要求值的表达式的名称；第二个参数指定当第一个参数求值为 NULL 时函数返回的值。如果第一个参数求值不为 NULL，那么函数将返回第一个参数的值。假设包括 name 和 address列的 student 表存在。同时假设对 student 表执行以下查询。

图: 查询

```
SELECT name, address FROM student;
```

图: 查询结果

```
name                address

        John Smith          333 Vista Drive
        Lauren Collier      1129 Greenridge Street
        Fred Frith          NULL
        Susan Jordan        NULL
```

以下是包括 NVL 函数的一个示例，该函数为表 address 列包含 NULL 值的每一行返回一个新值。

图: 查询

```
SELECT name, NVL(address, "address is unknown") AS address
        FROM student;
```

图: 查询结果

```
name                address
```

```
John Smith          333 Vista Drive
Lauren Collier      1129 Greenridge Street
Fred Frith          address is unknown
Susan Jordan        address is unknown
```

可以为 NVL 函数指定任何数据类型，只要这两个参数求值为公共兼容的数据类型。

如果 NVL 函数的两个参数都求值为 NULL，那么函数返回 NULL。

GBase 8s 还支持 NULLIF 函数。该函数类似于 NVL 函数。但语义不同。如果其两个参数相等，NULLIF 返回 NULL，或者两个参数不相等，将返回第一个参数。有关 NULLIF 函数的更多信息，请参阅《GBase 8s SQL 指南：语法》。

5.2 SELECT 语句中的 SPL 例程

本章前面的示例显示由列名、运算符和 SQL 函数组成的 SELECT 语句表达式。本章提供了包含 SPL 例程调用的表达式。

SPL 例程包含特定的存储过程语言（SPL）语句和 SQL 语句。有关 SPL 例程的更多信息，请参阅创建和使用 SPL 例程。

GBase 8s 允许用 C 和 Java™ 编写外部例程。有关更多信息，请参阅《GBase 8s 用户定义的例程和数据类型开发者指南》。

当您在投影列表中包含 SPL 例程表达式时，该 SPL 例程必须是返回单个值（一行一列）的例程。例如：仅当 test_func() 返回单个值时，以下语句才有效：

```
SELECT col_a, test_func(col_b) FROM tab1
        WHERE col_c = "Davis";
```

当您在 SELECT 语句的 Projection 子句中包含 SPL 例程表达式时，该 SPL 例程必须是返回单个值（一行一列）的例程。例如：仅当 test_func() 返回单个值时，数据库服务器返回一个错误消息。returns more than one value, the database server returns an error message.

SPL 例程通过允许您对选择的每行执行子查询来扩展可用函数的范围。

例如，假设您现有客户号、客户的姓和客户已下订单数的列表。下列查询查询了检索此信息的一种方法。customer 表具有 customer_num 和lname 列，但没有每个客户已下订单数的记录。可以编写 get_orders 例程，该例程查询每个 customer_num 的 orders 表并返回相应订单的数目（标记为 n_orders）。

图: 查询

```
SELECT customer_num, lname, get_orders(customer_num) n_orders
        FROM customer;
```

该结果显示了此 SPL 例程的输出。

图: 查询结果

```
customer_num      lname       n_orders

         101      Pauli              1
         102      Sadler             9
         103      Currie             9
         104      Higgins            4
         ⋮
         123      Hanlon             1
         124      Putnum             1
         125      Henry              0
         126      Neelie             1
         127      Satifer            1
         128      Lessor             0
```

使用 SPL 例程来封装查询中经常执行的操作。例如：以下查询中的条件包括例程 conv_price，该例程将库存商品的单击转换为不同的货币并添加任何进口关税。

图: 查询

```
SELECT stock_num, manu_code, description FROM stock
        WHERE conv_price(unit_price, ex_rate = 1.50,
        tariff = 50.00) < 1000;
```

5.3 数据加密函数

您可以将 SET ENCRYPTION PASSWORD 语句与内置 SQL 加密函数（使用 Advanced Encryption Standard （AES）和 Triple DES （3DES）加密）一起使用来保护您的敏感数据。如果使用加密，只有拥有正确密码的用户才能读取、复制或修改数据。

将SET ENCRYPTION PASSWORD 语句与下列内置加密和解密函数一起使用 ：

- ENCRYPT_AES

```
ENCRYPT_AES(data-string-expression
        [, password-string-expression [, hint-string-expression ]])
```

- ENCRYPT_TDES

```
ENCRYPT_TDES (data-string-expression
        [, password-string-expression [, hint-string-expression ]])
```

- DECRYPT_CHAR

```
DECRYPT_CHAR(EncryptedData [, PasswordOrPhrase])
```

- DECRYPT_BINARY

```
DECRYPT_BINARY(EncryptedData [, PasswordOrPhrase])
```

- GETHINT

```
GETHINT(EncryptedData)
```

如果您使用了 SET ENCRYPTION PASSWORD 语句来指定缺省密码，那么数据库服务器将该密码应用于同一会话中调用的对加密和解密函数的后续调用中。

使用 ENCRYPT_AES 和 ENCRYPT_TDES 定义加密的数据，使用 DECRYPT_CHAR 和 DECRYPT_BINARY 查询加密的数据。使用 GETHINT 显示密码提示符（如果在服务器上设置了该字符串）。

可以使用这些 SQL 内置函数来实现列级别或单元级别加密。

使用列级别加密，用相同的密码为给定列上的所有值加密。

使用单元级别加密，用不同的密码为列内的数据加密。

提示： 如果想从大型表中选择加密数据，请指定未加密的列。在其中选择行，可对包含加密数据的列创建索引或外键约束，但是这样做对资源使用的效率较低，原因是查询优化器不使用此类索引和外键约束。

5.3.1 使用列级别数据加密来保护信用卡数据

下列示例使用列级别加密来保护信用卡数据。

使用列级别加密来保护信用卡数据：

1. 创建表：create table customer (id char(30), creditcard lvarchar(67));
2. 插入加密数据：
   a. 设置会话密码：SET ENCRYPTION PASSWORD "credit card number is encrypted";
   b. 加密数据。

```
INSERT INTO customer VALUES
        ("Alice",  encrypt_aes("1234567890123456"));
        INSERT INTO customer VALUES
        ("Bob", encrypt_aes("2345678901234567"));
```

3. 使用解密函数查询加密数据。

```
SET ENCRYPTION PASSWORD "credit card number is encrypted";
    SELECT id FROM customer
    WHERE DECRYPT_CHAR(creditcard) = "2345678901234567";
```

重要： 加密数据值比相应的未加密数据占用更多的存储空间。列宽足够存储明文的列可能需要增大宽度才能支持列级别加密或单元级别加密。如果要将加密值插入声明宽度小于加密字符串的列，那么列存储截断后的值，该值无法被解密。

有关加密安全性的更多信息，请参阅《GBase 8s 管理员指南》。

有关内置加密和解密函数语法和存储要求的更多信息，请参阅《GBase 8s SQL 指南：语法》。

5.4 总结

本章介绍了在基本 SELECT 语句中用来查询关系数据库和处理返回数据的函数的样本语法和结果。在 SELECT 语句中使用函数 显示如何执行以下操作：

- 在 Projection 子句中使用聚集函数来计算并检索特定数据。
- 在 SELECT 语句中包括时间函数 DATE 、DAY 、MDY 、MONTH 、WEEKDAY 、YEAR 、CURRENT 和 EXTEND 以及 TODAY 、LENGTH 和 USER 函数。
- 在 SELECT 子句中使用转换函数来在日期与字符串之间转换。
- 在 SELECT 子句中使用字符串处理函数来转换大写和小写字母或以各种方法处理字符串。

6 编写高级 SELECT 语句

本章中增大了使用 SELECT 语句可执行的操作的范围。并使您能够执行更复杂的数据库查询和数据处理。编写 SELECT 语句着重于 SELECT 语句语法中的五个子句。本章添加了 GROUP BY 子句和 HAVING 子句。可以将 GROUP BY 子句与聚集函数配合使用来组织 FROM 子句返回的行。可以包括 HAVING 子句来对 GROUP BY 子句返回的值设置条件。

本章还扩展了连接的早期讨论。它说明了自连接（它使您能够将表连接至它本身）和四种类型的外连接（在其中应用关键字 OUTER 来以不同的方式处理两个或多个连接的表）。本章还介绍了相关和非相关子查询及其操作关键字，显示了如何使用 UNION 运算符来组合查询。并定义了称为联合、相交和差异的集合运算。

本章中的示例显示如何在查询中使用 SELECT 语句子句的一部分或全部。子句必须按以下顺序显示

1. Projection
2. FROM
3. WHERE

4. GROUP BY
5. HAVING
6. ORDER BY
7. INTO TEMP

有关以正确顺序使用所有这些子句的 SELECT 语句的示例，请参阅图 5。

附加 SELECT 语句子句 INTO （可用于在 SQL API 中指定程序和主变量）在 SQL 编程和随产品提供的出版物中进行描述。

本章还描述嵌套的 SELECT 语句，其中子查询在主查询的 Projection 、FROM 或 WHERE 子句中指定。其它几节说明 SELECT 语句如何定义和操作集合，以及如何对查询结果进行集合运算。

6.1 GROUP BY 和 HAVING 子句

可选 GROUP BY 和 HAVING 子句向 SELECT 语句添加功能。可以在基本 SELECT 语句中包括一个或全部两个子句来增大处理聚集的能力。

GROUP BY 子句组合类似的行，针对 Projection 子句中列出的每个列，为具有相同值的每组行生成单一结果行。HAVING 子句在构成组之后对那些组设置条件。可以不带 HAVING 子句使用 GROUP BY 子句或不带 GROUP BY 子句使用 HAVING 子句。

6.1.1 GROUP BY 子句

GROUP BY 子句将表分为几组。此子句通常与为每个这样的组生成总结值的聚集函数组合。编写 SELECT 语句中的某些示例显示了应用于整个表的聚集函数的用法。本章说明应用于行组的聚集函数。

使用不带聚集的 GROUP BY 子句与在 SELECT 子句中使用 DISTINCT（或 UNIQUE）关键字很相似。下列查询在选择特定列中描述。

图: 查询

```
SELECT DISTINCT customer_num FROM orders;
```

还可以按以下查询编写此语句。

图: 查询

```
SELECT customer_num FROM orders
        GROUP BY customer_num;
```

图 1和图 2返回下列行。

图: 查询结果

```
customer_num
```

```
101
104
106
110
⋮
124
126
127
```

GROUP BY 子句将行收集到组中，因此每一组中的每一行具有相同的客户号。在没有选择任何其它列的情况下，结束是唯一 customer_num 值的列表。

GROUP BY 子句的功能在将它与聚集函数配合使用时更明显。

下列查询检索每个订单的商品数和所有商品的总价。

图: 查询

```
SELECT order_num, COUNT (*) number, SUM (total_price) price
        FROM items
        GROUP BY order_num;
```

GROUP BY 子句导致 items 表的行数被收集为组，每个组由具有相同 order_num 值的行组成（即，将每个订单的商品收集在一起）。在数据库服务器构成组之后，就在每个组中应用聚集行数 COUNT 和 SUM 。

图 4对每一组返回每一行。它还使用标号来为 COUNT 和 SUM 表达式的结果提供名称，如下所示。

图: 查询结果

```
order_num       number          price

    1001            1          $250.00
    1002            2         $1200.00
    1003            3          $959.00
    1004            4         $1416.00
    ⋮
    1021            4         $1614.00
    1022            3          $232.00
    1023            6          $824.00
```

该结果将 items 表的行收集到具有相同订单号的组中，并计算每个组中行的 COUNT 和价格的 SUM。

不能在 GROUP BY 子句中包含 TEXT 、BYTE 、CLOB 或 BLOB 列。要进行分组，必须能够进行排序，并且这些数据类型不存在自然排序顺序。

与 ORDER BY 子句不同，GROUP BY 子句不对数据进行排序。如果想要按特定顺序对数据进行排序，或在投影列表中的聚集上排序，那么在 GROUP BY 子句之后包含 ORDER BY 子句。

下列查询与图 4相同，但包括 ORDER BY 子句以按 price 的升序对检索到的行进行排序，如下所示。

图: 查询

```
SELECT order_num, COUNT(*) number, SUM (total_price) price
        FROM items
        GROUP BY order_num
        ORDER BY price;
```

图: 查询结果

order_num	number	price
1010	2	$84.00
1011	1	$99.00
1013	4	$143.80
1022	3	$232.00
1001	1	$250.00
1020	2	$438.00
1006	5	$448.00
⋮		
1002	2	$1200.00
1004	4	$1416.00
1014	2	$1440.00
1019	1	$1499.97
1021	4	$1614.00
1007	5	$1696.00

选择特定列一节描述如何在 ORDER BY 子句中使用整数来指示投影列表中列的位置。还可以在 GROUP BY 子句中使用整数来指示列名的位置或在 GROUP BY 列表中显示标号。

以下查询返回与图 6所示相同的行。

图: 查询

```
SELECT order_num, COUNT(*) number, SUM (total_price) price
```

```
        FROM items
        GROUP BY 1
        ORDER BY 3;
```

构建查询时，Projection 子句的投影列表中的所有非聚集列还必须包含在 GROUP BY 子句中。具有 GROUP BY 子句的 SELECT 语句必须针对每一组返回一行。列出在 GROUP BY 后面的列能够在一组中只反映一个特异值。并且可以返回该值。但是，未列出在 GROUP BY 后面的列可在包含在组中的行中包含不同的值。

下列查询显示如何在连接表的 SELECT 语句中使用 GROUP BY 子句。

图: 查询

```
SELECT o.order_num, SUM (i.total_price)
        FROM orders o, items i
        WHERE o.order_date > '01/01/98'
        AND o.customer_num = 110
        AND o.order_num = i.order_num
        GROUP BY o.order_num;
```

该查询连接 orders 和 items 表，将表别名指定给它们，并返回以下所示的行。

图: 查询结果

```
order_num          (sum)

    1008           $940.00
    1015           $450.00
```

6.1.2 HAVING 子句

要完成 GROUP BY 子句，使用 HAVING 子句来在构成组之后将一个或多个限制条件应用于这些组。HAVING 子句对组的影响类似于 WHERE 子句限定个别行的方式，使用 HAVING 子句的一个优点是可以在搜索条件中包括聚集，而在 WHERE 子句的搜索条件中去不能包含聚集。

每个 HAVING 条件将组的一列或一个聚集表达式与组的另一个聚集表达式或与常量作比较。可以使用 HAVING 来对列值或组列表中的聚集值设置条件。

下列查询返回具有两个商品以上的订单上每个商品的平均总价格。HAVING 子句在每个组构成时测试每个组，并选择由两行以上构成的那些组。

图: 查询

```
SELECT order_num, COUNT(*) number, AVG (total_price) average
        FROM items
```

```
        GROUP BY order_num
        HAVING COUNT(*) > 2;
```

图: 查询结果

order_num	number	average
1003	3	$319.67
1004	4	$354.00
1005	4	$140.50
1006	5	$89.60
1007	5	$339.20
1013	4	$35.95
1016	4	$163.50
1017	3	$194.67
1018	5	$226.20
1021	4	$403.50
1022	3	$77.33
1023	6	$137.33

如果使用不带 GROUP BY 子句的 HAVING 子句，那么 HAVING 条件应用于满足搜索条件的所有行。也就是说，满足搜索条件的所有行组成了一个组。

下列查询（图 1的修改版本）只返回一行，即表中所有 total_price 值的平均数，如下所示。

图: 查询

```
SELECT AVG (total_price) average
        FROM items
        HAVING count(*) > 2;
```

图: 查询结果

average
$270.97

如果图 3与图 1一样，在 Projection 子句中包含了非聚集列 order_num ，那么必须将 GROUP BY 子句与组列表中的列包含子啊一起。此外，如果不满足 HAVING 子句中的条件，那么输出将显示列标题以及一条消息指示没有找到任何行。

下列查询包含可以在 GBase 8s 版本的交互 SQL 中使用的所有 SELECT 语句子句（命名主变量的 INTO 子句只在 SQL API 中可用）。

图: 查询

```
SELECT o.order_num, SUM (i.total_price) price,
```

```
        paid_date - order_date span
        FROM orders o, items i
        WHERE o.order_date > '01/01/98'
        AND o.customer_num > 110
        AND o.order_num = i.order_num
        GROUP BY 1, 3
        HAVING COUNT (*) < 5
        ORDER BY 3
        INTO TEMP temptab1;
```

该查询连接 orders 和 items 表；使用显示标号、表列名和用作列指示符的整数；对数据进行分组和排序；并将结果放置在临时表中，如下所示。

图: 查询结果

```
order_num       price       span

     1017       $584.00
     1016       $654.00
     1012      $1040.00
     1019      $1499.97        26
     1005       $562.00        28
     1021      $1614.00        30
     1022       $232.00        40
     1010        $84.00        66
     1009       $450.00        68
     1020       $438.00        71
```

6.2 创建高级连接

创建连接一节显示如何在 SELECT 语句中包括 WHERE 子句以在一个列或多个列上连接两个或多个表。它说明了自然连接和等值连接。

本章讨论如何使用两种更复杂的连接：自连接和外连接。如对简单连接描述的那样，可以为表定义别名并将显示标号指定给表达式以缩短多表查询时间。还可以带 ORDER BY 子句发出 SELECT 语句，这将把数据排序到临时表中。

6.2.1 自连接

连接不一定总是涉及两个不同的表，可以将表连接至它本身，创建自连接。当想要将列中的值与同一列中的其他值进行比较时，将表连接至它本身非常有用。

要创建自连接，在 FROM 子句中列出表两次，并且每次为它指定不同的别名。使用别名在 Projection 和 WHERE 子句中引用表。如同是两个独立的表一样。（SELECT 语句中的别名在别名和 GBase 8s SQL 指南：语法中讨论。）

与表之间的连接一样，可以在自连接中使用算术表达式，可以测试空值。可以使用 ORDER BY 子句来以升序或降序对指定列中的值进行排序。

下列查询查询 ship_weight 相差五倍或更多并且 ship_date 不为空的订单。接着，查询按照 ship_date 对数据进行排序。

图: 查询

```
SELECT x.order_num, x.ship_weight, x.ship_date,
        y.order_num, y.ship_weight, y.ship_date
        FROM orders x, orders y
        WHERE x.ship_weight >= 5 * y.ship_weight
        AND x.ship_date IS NOT NULL
        AND y.ship_date IS NOT NULL
        ORDER BY x.ship_date;
```

表 1. 查询结果

order_num	ship_weight	ship_date	order_num	ship_weight	ship_date
1004	95.80	05/30/1998	1011	10.40	07/03/1998
1004	95.80	05/30/1998	1020	14.00	07/16/1998
1004	95.80	05/30/1998	1022	15.00	07/30/1998
1007	125.90	06/05/1998	1015	20.60	07/16/1998
1007	125.90	06/05/1998	1020	14.00	07/16/1998

如果想要将自连接的结果存储到临时表中，那么将 INTO TEMP 子句追加到 SELECT 语句中，并至少对一组列指定显示标号，以重命名这些列。否则，重复列名将导致错误，并且不会创建临时表。

下列查询，类似于图 1，标记从 orders 表选择的所有列，并将这些列放置在称为 shipping 的临时表中。

图: 查询

```
SELECT x.order_num orders1, x.po_num purch1,
        x.ship_date ship1, y.order_num orders2,
```

```
        y.po_num purch2, y.ship_date ship2
        FROM orders x, orders y
        WHERE x.ship_weight >= 5 * y.ship_weight
        AND x.ship_date IS NOT NULL
        AND y.ship_date IS NOT NULL
        ORDER BY orders1, orders2
        INTO TEMP shipping;
```

如果您从表 shipping 中查询 SELECT *，可以看到下列行。

图: 查询结果

```
orders1 purch1      ship1          orders2 purch2      ship2

        1004 8006       05/30/1998          1011 B77897     07/03/1998
        1004 8006       05/30/1998          1020 W2286      07/16/1998
        1004 8006       05/30/1998          1022 W9925      07/30/1998
        1005 2865       06/09/1998          1011 B77897     07/03/1998
        :
        1019 Z55709     07/16/1998          1020 W2286      07/16/1998
        1019 Z55709     07/16/1998          1022 W9925      07/30/1998
        1023 KF2961     07/30/1998          1011 B77897     07/03/1998
```

可以多次将表连接至它本身。自连接的最大次数取决于您可用的资源。

下列查询中的自连接在 stock 表中创建由三个制造商供货的那些商品的列表。自连接在 WHERE 子句中包括最后两个条件，来除去行中检索到的重复的制造商代码。

图: 查询

```
SELECT s1.manu_code, s2.manu_code, s3.manu_code,
        s1.stock_num, s1.description
        FROM stock s1, stock s2, stock s3
        WHERE s1.stock_num = s2.stock_num
        AND s2.stock_num = s3.stock_num
        AND s1.manu_code < s2.manu_code
        AND s2.manu_code < s3.manu_code
        ORDER BY stock_num;
```

图: 查询结果

```
manu_code manu_code manu_code stock_num description

          HRO       HSK       SMT             1 baseball gloves
```

```
ANZ     NRG     SMT      5 tennis racquet
ANZ     HRO     HSK      110 helmet
ANZ     HRO     PRC      110 helmet
ANZ     HRO     SHM      110 helmet
ANZ     HSK     PRC      110 helmet
ANZ     HSK     SHM      110 helmet
ANZ     PRC     SHM      110 helmet
HRO     HSK     PRC      110 helmet
HRO     HSK     SHM      110 helmet
HRO     PRC     SHM      110 helmet
⋮
KAR     NKL     PRC      301 running shoes
KAR     NKL     SHM      301 running shoes
KAR     PRC     SHM      301 running shoes
NKL     PRC     SHM      301 running shoes
```

如果想要从 payroll 表选择行来确定哪些职员的薪水高于他们的经理，可以按以下 SELECT 语句所示构造自连接：

```
SELECT emp.employee_num, emp.gross_pay, emp.level,
        emp.dept_num, mgr.employee_num, mgr.gross_pay,
        mgr.dept_num, mgr.level
        FROM payroll emp, payroll mgr
        WHERE emp.gross_pay > mgr.gross_pay
        AND emp.level < mgr.level
        AND emp.dept_num = mgr.dept_num
        ORDER BY 4;
```

下列查询使用相关子查询来检索并列出预订的 10 种价格最高的商品。

图: 查询

```
SELECT order_num, total_price
        FROM items a
        WHERE 10 >
        (SELECT COUNT (*)
        FROM items b
        WHERE b.total_price < a.total_price)
        ORDER BY total_price;
```

该查询返回 10 行。

图: 查询结果

order_num	total_price
1018	$15.00
1013	$19.80
1003	$20.00
1005	$36.00
1006	$36.00
1013	$36.00
1010	$36.00
1013	$40.00
1022	$40.00
1023	$40.00

可以创建类似的查询来查找并列出公司中资格最老的 10 个职员。

有关相关子查询的更多信息，请参阅SELECT 语句中的子查询。

6.2.2 外连接

本章显示如何在 SELECT 语句中创建和使用外部连接。创建连接讨论内部连接。尽管内连接同等看待两个或多个连接的表，但外连接不同等看待两个或多个连接的表。外连接使其中一个表成为控制表（也称为外部表），控制其他从属表（也称为内部表）。

在内连接或简单连接中，结果只包含满足连接条件的行组合。废弃不满足连接条件的行。

在外连接中，结果包含满足连接条件的行与控制表中的行（如果在从属表中找不到匹配的行那么将废弃这些行）的组合。在从属表中无匹配行控制表的行在选自从属表的列中包括 NULL 值。

外连接允许您在应用连接条件之前将连接过滤器应用于内部表。

数据库服务器的较早版本只支持对用于外连接的 ANSI-SQL 标准语法的 GBase 8s 扩展。此语法仍受支持。然而，ANSI-SQL 标准语法在创建查询方面灵活性更高。建议使用 ANSI-SQL 标准语法来创建新查询。不管您使用何种形式的语法，必须将它用于单个查询块中的所有外连接。

在依赖于外连接之前，确定一个或多个内连接是否可工作。当不需要来自其它表的补充信息时，通常可以使用内连接。

限制： 不能在同一查询块中组合 GBase 8s 和 ANSI 外连接语法。

有关外连接的语法的信息，请参阅《GBase 8s SQL 指南：语法》。

对外连接语法的 GBase 8s 扩展

对外连接语法的 GBase 8s 扩展在外连接的开始处使用 OUTER 关键字。当使用 GBase 8s 语法时，必须在 WHERE 子句中包含连接条件。在将 GBase 8s 语法用于外连接时，数据库服务器支持以下三种基本类型的外连接：

- 对两个表的外连接
- 与第三个表进行简单连接的外连接
- 将两个表与第三个表进行外连接

外连接必须具有 Projection 子句、FROM 子句和 WHERE 子句。连接条件在 WHERE 子句中表述。要将简单连接转换为外连接，在 FROM 子句中从属表的名称前面之间插入关键字 OUTER 。如在本节中后面所示，可以在查询中多次包括 OUTER 关键字。

没有对外连接语法的 GBase 8s 扩展等价于 ANSI 右外连接。

ANSI 连接语法

以下 ANSI 连接受支持：

- 左外连接
- 右外连接

ANSI 外连接语法用 LEFT JOIN 、LEFT OUTER JOIN 、RIGHT JOIN 或 RIGHT OUTER JOIN 关键字开始外连接。OUTER 关键字是可选的。查询可在 ON 子句中指定连接条件和可选连接过滤器。WHERE 子句指定后连接（post-join）过滤器。另外，可以使用 LEFT 或 right 子句显式指定连接的类型。当用左括号开始连接时，ANSI 连接语法还允许外连接的控制部分或从属部分作为另一个连接的结果集。

如果将 ANSI 语法用于外连接，那么必须将 ANSI 语法用于单个查询块中的所有外连接。

提示： 为了简介起见，本节中的示例使用表别名。别名讨论表别名。

左外连接

在左外连接的语法中，外连接的控制表显示在开始外连接的关键字左边。左外连接返回连接条件为 true 的所有行，除此之外，还返回控制表中的所有其它行并将从属表中的相应值显示为 NULL。

下列查询使用 ANSI 语法 LEFT OUTER JOIN 来获取与图 1（它使用 GBase 8s 外连接语法）相同的结果：

图: 查询

```
SELECT c.customer_num, c.lname, c.company, c.phone,
        u.call_dtime, u.call_descr
        FROM customer c LEFT OUTER JOIN cust_calls u
```

```
        ON c.customer_num = u.customer_num;
```

在此示例中，可以使用 ON 子句来指定连接条件。可以在 WHERE 子句中添加其它过滤器来限制结果集；此类过滤器是后连接（post-join）过滤器。

以下查询只返回客户没有致电客户服务中心的行。在此查询中，数据库服务器对 customer 和 cust_calls 表的 customer_num 列执行外连接之后在 WHERE 子句中应用过滤器。

图: 查询

```
SELECT c.customer_num, c.lname, c.company, c.phone,
        u.call_dtime, u.call_descr
        FROM customer c LEFT OUTER JOIN cust_calls u
        ON c.customer_num = u.customer_num
        WHERE u.customer_num IS NULL;
```

除了前面的示例之外，下列示例显示了可与 ANSI 连接语法配合使用的各种查询构造类型：

```
SELECT *
        FROM (t1 LEFT OUTER JOIN (t2 LEFT OUTER JOIN t3 ON t2.c1=t3.c1)
        ON t1.c1=t3.c1) JOIN (t4 LEFT OUTER JOIN t5 ON t4.c1=t5.c1)
        ON t1.c1=t4.c1;

        SELECT *
        FROM (t1 LEFT OUTER JOIN (t2 LEFT OUTER JOIN t3 ON t2.c1=t3.c1)
        ON t1.c1=t3.c1),
        (t4 LEFT OUTER JOIN t5 ON t4.c1=t5.c1)
        WHERE t1.c1 = t4.c1;

        SELECT *
        FROM (t1 LEFT OUTER JOIN (t2 LEFT OUTER JOIN t3 ON t2.c1=t3.c1)
        ON t1.c1=t3.c1) LEFT OUTER JOIN (t4 JOIN t5 ON t4.c1=t5.c1)
        ON t1.c1=t4.c1;

        SELECT *
        FROM t1 LEFT OUTER JOIN (t2 LEFT OUTER JOIN t3 ON t2.c1=t3.c1)
        ON t1.c1=t2.c1;

        SELECT *
        FROM t1 LEFT OUTER JOIN (t2 LEFT OUTER JOIN t3 ON t2.c1=t3.c1)
```

```
        ON t1.c1=t3.c1;

        SELECT *
        FROM (t1 LEFT OUTER JOIN t2 ON t1.c1=t2.c1)
        LEFT OUTER JOIN t3 ON t2.c1=t3.c1;

        SELECT *
        FROM (t1 LEFT OUTER JOIN t2 ON t1.c1=t2.c1)
        LEFT OUTER JOIN t3 ON t1.c1=t3.c1;

        SELECT *
        FROM t9, (t1 LEFT JOIN t2 ON t1.c1=t2.c1),
        (t3 LEFT JOIN t4 ON t3.c1=10), t10, t11,
        (t12 LEFT JOIN t14 ON t12.c1=100);

        SELECT * FROM
        ((SELECT c1,c2 FROM t3) AS vt3(v31,v32)
        LEFT OUTER JOIN
        ( (SELECT c1,c2 FROM t1) AS vt1(vc1,vc2)
        LEFT OUTER JOIN
        (SELECT c1,c2 FROM t2) AS vt2(vc3,vc4)
        ON vt1.vc1 = vt2.vc3)
        ON vt3.v31 = vt2.vc3);
```

上面最后一个示例说明了关于派生表的连接。它指定将外查询的 FROM 子句中子查询的结果和另一个其它两个子查询结果的左外连接的结果进行左外连接。请参阅FROM 子句中的子查询获得较为简单的符合 ANSI 语法的子查询示例。

右外连接

在右外连接的语法中，外连接的控制表显示在开始外连接的关键字右边。右外连接返回连接条件为 true 的所有行，除此之外，还返回控制表中的所有其它行并将从属表中的相应值显示为 NULL。

下列查询是对 customer 和 orders 表上的右外连接的一个示例。

图: 查询

```
SELECT c.customer_num, c.fname, c.lname, o.order_num,
        o.order_date, o.customer_num
```

```
        FROM customer c RIGHT OUTER JOIN orders o
        ON (c.customer_num = o.customer_num);
```

该查询返回控制表 orders 中的所有行，并且在必要时，将从属表 customer 中的相应值显示为 NULL。

图: 查询结果

```
customer_num     fname       lname order_num     order_date    customer_num
        104    Anthony    Wiggins         1001     05/30/1998             104
        101    Ludwig       Pauli         1002     05/30/1998             101
        104 Anthony     Wiggins         1003     05/30/1998             104
        <NULL>   <NULL>       <NULL>         1004     06/05/1998             106
```

简单连接

以下查询是 customer 和 cust_calls 表上简单连接的示例。

图: 查询

```
SELECT c.customer_num, c.lname, c.company,
        c.phone, u.call_dtime, u.call_descr
        FROM customer c, cust_calls u
        WHERE c.customer_num = u.customer_num;
```

该查询只返回客户已致电客户服务中心的那些行，如下所示。

图: 查询结果

```
customer_num    106
        lname           Watson
        company          Watson & Son
        phone           415-389-8789
        call_dtime      1998-06-12 08:20
        call_descr      Order was received, but two of the cans of
        ANZ tennis balls within the case were empty
        ⋮
        customer_num    116
        lname           Parmelee
        company          Olympic City
        phone           415-534-8822
        call_dtime      1997-12-21 11:24
        call_descr      Second complaint from this customer! Received
        two cases right-handed outfielder gloves (1 HRO)
```

```
        instead of one case lefties.
```

对两个表的简单外连接

下列查询使用与前面示例相同的 Projection 子句、表和比较条件，但这一次它用 GBase 8s 扩展语法创建简单外连接。

图: 查询

```
SELECT c.customer_num, c.lname, c.company,
        c.phone, u.call_dtime, u.call_descr
        FROM customer c, OUTER cust_calls u
        WHERE c.customer_num = u.customer_num;
```

cust_calls 表前面的附加关键字 OUTER 使该表成为从属表。外连接导致查询返回有关所有客户的信，而不管它们是否已致电客户服务中心。检索控制表 customer 的所有行，并且将 NULL 值指定给从属表 cust_calls 的列，如下所示。

图: 查询结果

```
customer_num    101
        lname           Pauli
        company          All Sports Supplies
        phone           408-789-8075
        call_dtime
        call_descr

        customer_num    102
        lname           Sadler
        company          Sports Spot
        phone           415-822-1289
        call_dtime
        call_descr
        .
        customer_num    107
        lname           Ream
        company          Athletic Supplies
        phone           415-356-9876
        call_dtime
        call_descr
```

```
customer_num  108
lname         Quinn
company       Quinn's Sports
phone         415-544-8729
call_dtime
call_descr
```

与第三个表进行简单连接的外连接

使用 GBase 8s 语法，下列查询显示作为第三个表的简单连接结果的外连接。这第二种类型的外连接也称为嵌套简单连接。

图: 查询

```
SELECT c.customer_num, c.lname, o.order_num,
        i.stock_num, i.manu_code, i.quantity
        FROM customer c, OUTER (orders o, items i)
        WHERE c.customer_num = o.customer_num
        AND o.order_num = i.order_num
        AND manu_code IN ('KAR', 'SHM')
        ORDER BY lname;
```

该查询首先对 orders 和 items 表执行简单连接，并检索 manu_code 为 KAR 或 SHM 的商品的所有订单的信息。然后，它执行外连接以将此消息与控制 customer 表的数据结合。可选的 ORDER BY 子句将数据重组为以下格式。

图: 查询结果

```
customer_num lname          order_num stock_num manu_code quantity

         114 Albertson
         118 Baxter
         113 Beatty
         ⋮
         105 Vector
         121 Wallack           1018      302 KAR          3
         106 Watson
```

将两个表与第三个表相连接

使用 GBase 8s 扩展语法，下列查询显示作为两个表分别与第三个表的外连接结果的外连接。在此第三种类型的外连接中，连接关系可能仅仅是控制表与从属表之间的关系。

图: 查询

```
SELECT c.customer_num, c.lname, o.order_num,
        order_date, call_dtime
        FROM customer c, OUTER orders o, OUTER cust_calls x
        WHERE c.customer_num = o.customer_num
        AND c.customer_num = x.customer_num
        ORDER BY lname
        INTO TEMP service;
```

该查询分别将从属表 orders 和 cust_calls 连接至控制表 customer；它不连接两个从属表。INTO TEMP 子句将结果选择至临时表以供进一步处理或查询，如下所示。

图: 查询结果

```
customer_num lname              order_num order_date call_dtime

        114 Albertson
        118 Baxter
        113 Beatty
        103 Currie
        115 Grant                   1010 06/17/1998
        :
        117 Sipes                   1012 06/18/1998
        105 Vector
        121 Wallack                 1018 07/10/1998 1998-07-10 14:05
        106 Watson                   1004 05/22/1998 1998-06-12 08:20
        106 Watson                   1014 06/25/1998 1998-06-12 08:20
```

如果图 1尝试在两个从属表 o 和 x 之间创建连接条件（如下所示），一条错误消息将指示创建两侧外连接。

图: 查询

```
WHERE o.customer_num = x.customer_num
```

组合外连接的连接

要实现多级嵌套，可以创建使用三种外连接类型的任何组合的连接。使用 ANSI 语法，以下查询创建了作为对两个表与另一个外连接的简单外连接组合结果的连接。

图: 查询

```
SELECT c.customer_num, c.lname, o.order_num,
        stock_num, manu_code, quantity
```

```
    FROM customer c, OUTER (orders o, OUTER items i)
    WHERE c.customer_num = o.customer_num
    AND o.order_num = i.order_num
    AND manu_code IN ('KAR', 'SHM')
    ORDER BY lname;
```

该查询首先执行 orders 和 items 表上的外连接，并检索有关 manu_code 为 KAR 或 SHM 的商品的所有订单的信息。然后，它执行组合此信息与控制表 customer 的数据的另一个外连接。

图: 查询结果

```
customer_num lname          order_num stock_num manu_code quantity

         114 Albertson
         118 Baxter
         113 Beatty
         103 Currie
         115 Grant              1010
         .
         .
         .
         117 Sipes              1012
         117 Sipes              1007
         105 Vector
         121 Wallack            1018      302 KAR         3
         106 Watson              1014
         106 Watson              1004
```

当将外连接应用于某外连接与第三个表的结果时可用两种方法指定连接查询。两个从属表已连接，但如果控制表和从属表共享公共列，那么可以将控制表连接至任一从属表而不影响结果。

6.3 SELECT 语句中的子查询

子查询（内部 SELECT 语句，其中一个 SELECT 语句嵌套在另一个 SELECT 语句中）可以返回多行或多个表达式，也可以不返回任何结果。每个子查询必须用括号分隔，并且都必须包含一个 Projection 子句和一个 FROM 子句，子查询本身可以包含其它子查询。

数据库服务器支持下列上下文中的子查询：

- 嵌套在另一个 SELECT 语句的 Projection 子句中的 SELECT 语句
- 嵌套在另一个 SELECT 语句中的 WHERE 子句中的 SELECT 语句

- 嵌套在另一个 SELECT 语句的 FROM 子句中的 SELECT 语句

还可以在 INSERT 、DELETE 、MERGE 或 UPDATE 语句（子查询有效）的各种子句中指定子查询。

Projection 子句或 WHERE 子句中的子查询可以是相关的或是不相关的。当子查询产生的值取决于包含它的外部 SELECT 语句产生的值时，该子查询是相关的。有关更多信息，请参阅相关子查询。

任何其它类型的子查询都被认为是不相关的。在 SELECT 语句的 FROM 子句中，只有不相关的子查询才是有效的。

6.3.1 相关子查询

相关子查询是引用不列在其 FROM 子句中的表的列的子查询。该列可以在 Projection 子句或在 WHERE 子句中。要查找相关子查询引用的表，搜索列直到找到相关为止。

通常，相关子查询会降低性能。在子查询中使用表名或别名，这样就不会对所在的表产生疑问。

数据库服务器将使用外查询来获取值。例如：如果表 taba 具有列 col1，表 tabb 具有列 col2，并且它们包含以下内容：

```
taba.col1          aa,bb,null
            tabb.col2          bb, null
```

那么查询为：

```
select * from taba where col1 in (select col1 from tabb);
```

那么结果可能会毫无意义。数据库服务器将提供 taba.col1 中所有的值，并接着它们与 taba.col1 进行比较（外查询 WHERE 子句）。这将返回所有的行。通常使用子查询从内表返回列值。如果查询写成：

```
select * from taba where col1 in (select tabb.col1 from tabb);
```

那么将导致错误 error -217 column not found。

相关子查询的重要功能是，由于它取决于来自外部 SELECT 的值。所以必须重复执行它，对外部 SELECT 产生的每个值执行一次。非相关子查询只能执行一次。

6.3.2 SELECT 语句中的子查询

可以构造具有子查询的 SELECT 语句来替换两个独立的 SELECT 语句。

SELECT 语句中的子查询允许您执行各种任务，包括下列操作：

- 将表达式与另一 SELECT 语句的结果进行比较
- 确定另一 SELECT 语句的结果是否包含特定的表达式

- 确定另一 SELECT 语句是否选择任何行

子查询中的可选 WHERE 子句通常用于缩小搜索条件。

子查询选择值并将值返回到第一个或外部 SELECT 语句。子查询可以不返回任何值。返回单个值或返回一组值，如下所示：

- 如果子查询不返回任何值，那么查询不返回任何行。该子查询等价于 NULL 值。
- 如果子查询返回一个值，那么该值的格式为一个聚集表达式或就是一行和一列。此类子查询等价于一个数字或字符值。
- 如果子查询返回一列或一组值，那么这些值可表示一行或一列。
- 在外部查询的 FROM 子句中，子查询可表示一组行（有时候称为派生表或表表达式）。

6.3.3 Projection 子句中的子查询

子查询可发生在另一个 SELECT 语句的 Projection 子句中。下列查询显示如何在 Projection 子句中使用子查询来返回 customer 表中每个顾客的总装运费用（来自 orders 表）。还可以将此查询编写为两个表之间的连接。

图: 查询

```
SELECT customer.customer_num,
        (SELECT SUM(ship_charge)
        FROM orders
        WHERE customer.customer_num = orders.customer_num)
        AS total_ship_chg
        FROM customer;
```

图: 查询结果

```
customer_num total_ship_chg

         101        $15.30
         102
         103
         104        $38.00
         105
         ⋮
         123         $8.50
         124        $12.00
         125
```

```
        126     $13.00
        127     $18.00
        128
```

6.3.4 FROM 子句中的子查询

本节描述作为嵌套在外部 SELECT 语句的 FROM 子句中发生的子查询。由于外部查询使用子查询的结果作为数据源，因而此类子查询有时候称为派生表或表表达式。

下列查询在外部查询中使用星号表示法来返回检索 employee 表中 address 列所有字段的子查询的结果。

图: 查询

```
SELECT * FROM (SELECT address.* FROM employee);
```

图: 查询结果

```
address     ROW(102 Ruby, Belmont, CA, 49932, 1000)
            address     ROW(133 First, San Jose, CA, 85744, 4900)
            address     ROW(152 Topaz, Willits, CA, 69445, 1000))
            :
```

这说明了如何指定派生表，但是它只是该语法的一个价值不高的示例，因为外部查询不操作 FROM 子句中的子查询返回的表表达式中的任何值。（请参阅图 1获取返回相同结果的简单查询。）

下列查询是一个更复杂的示例，其中外部查询仅选择派生表中满足条件的第一行，FROM 子句中的子查询将此派生表中满足条件的第一行，FROM 子句中的子查询将此派生表指定为 customer 和 cust_calls 表的简单连接。

图: 查询

```
SELECT LIMIT 1 * FROM
            (SELECT c.customer_num, c.lname, c.company,
            c.phone, u.call_dtime, u.call_descr
            FROM customer c, cust_calls u
            WHERE c.customer_num = u.customer_num
            ORDER BY u.call_dtime DESC);
```

该查询只返回客户已致电客户服务中心的那些行，如下所示。

图: 查询结果

```
customer_num    106
            lname           Watson
            company         Watson & Son
```

```
phone           415-389-8789
call_dtime      1998-06-12 08:20
call_descr      Order was received, but two of the cans of
ANZ tennis balls within the case were empty
```

在前面的示例中，子查询包括 ORDER BY 子句，它指定出现在子查询的 Projection 列表中的一列，但如果 Projection 列表省略了 u.call_dtime列，查询还是有效的。子查询仅可在 FROM 子句这个上下文中指定 ORDER BY 子句。

6.3.5 WHERE 子句中的子查询

本节描述嵌套在另一 SELECT 语句的 WHERE 子句中的 SELECT 语句发生的子查询。

可以将任何关系运算符与 ALL 和 ANY 配合使用来将一些内容与子查询生成的值的每一个（ALL）或任一个（ANY）进行比较。可以使用关键字 SOME 代替 ANY。运算符 IN 等价于 = ANY。要创建相反的搜索条件，使用关键字 NOT 或另一个关系运算符。

EXISTS 运算符对子查询进行测试以了解子查询是否找到了任何值。即，该运算符询问子查询的结果是否非空。不能在包含具有 TEXT 或 BYTE 数据类型的列的子查询中使用 EXISTS 关键字。

有关用于创建带子查询的条件的语法，请参阅《GBase 8s SQL 指南：语法》。

下列关键字介绍了 SELECT 语句的 WHERE 子句中的子查询。

ALL 关键字

在子查询前面使用 ALL 关键字来确定对返回的每个值的比较是否为 true。如果子查询不返回任何值，那么搜索条件为 true。（如果子查询不返回任何值，那么对于所有零值条件为 true 。）

下列查询列出了包含总价小于订单号 1023 中每个商品的总价的商品的所有订单的以下信息。

图: 查询

```
SELECT order_num, stock_num, manu_code, total_price
        FROM items
        WHERE total_price < ALL
        (SELECT total_price FROM items
        WHERE order_num = 1023);
```

图: 查询结果

```
order_num stock_num manu_code total_price
```

1003	9 ANZ	$20.00
1005	6 SMT	$36.00
1006	6 SMT	$36.00
1010	6 SMT	$36.00
1013	5 ANZ	$19.80
1013	6 SMT	$36.00
1018	302 KAR	$15.00

ANY 关键字

在子查询前面使用关键字 ANY （或它的同义词 SOME）来确定是否对至少一个返回值的比较为 true 。如果子查询不返回任何值，那么搜索条件为false。（因为没有值存在，所以对于其中一个值条件不能为 true 。）

以下查询查找包含总价大于订单号 1005 中任何一个商品总价的商品的所有订单的订单号。

图: 查询

```
SELECT DISTINCT order_num
        FROM items
        WHERE total_price > ANY
        (SELECT total_price
        FROM items
        WHERE order_num = 1005);
```

图: 查询结果

```
order_num

        1001
        1002
        1003
        1004
        ⋮
        1020
        1021
        1022
        1023
```

单值子查询

如果您知道子查询可能对外部级别查询返回刚好一个值，那么不需要 ALL 或 ANY 。可如同对待函数一样对待只返回一个值的子查询。这种子查询通常使用聚集函数，原因是聚集函数总是返回单个的。

下列查询在子查询中使用聚集函数 MAX 查找包括最大排球网数目的订单的 order_num。

图: 查询

```
SELECT order_num FROM items
        WHERE stock_num = 9
        AND quantity =
        (SELECT MAX (quantity)
        FROM items
        WHERE stock_num = 9);
```

图: 查询结果

```
  order_num


      1012
```

下列查询在子查询中使用聚集函数 MIN 选择总价高于最小价格 10 倍的商品。

图: 查询

```
SELECT order_num, stock_num, manu_code, total_price
        FROM items x
        WHERE total_price >
        (SELECT 10 * MIN (total_price)
        FROM items
        WHERE order_num = x.order_num);
```

图: 查询结果

```
order_num stock_num manu_code   total_price

     1003          8 ANZ            $840.00
     1018        307 PRC            $500.00
     1018        110 PRC            $236.00
     1018        304 HRO            $280.00
```

相关子查询

相关子查询是引用不在其 FROM 子句中的列或表的子查询。该列可以在 Projection 子句或 WHERE 子句中。

通常，相关子查询会降低性能。建议使用表名或表别名限制子查询中的列名。从而除去与列所驻留的表相关的任何疑问。

下列查询是相关子查询的一个示例，它返回 orders 表中 10 个最近的装运日期的列表。它在子查询之后加上 ORDER BY 子句以对结果进行排序，原因是（除在 FROM 子句以外）您不能在子查询中包括 ORDER BY 。

图: 查询

```
SELECT po_num, ship_date FROM orders main
        WHERE 10 >
        (SELECT COUNT (DISTINCT ship_date)
        FROM orders sub
        WHERE sub.ship_date < main.ship_date)
        AND ship_date IS NOT NULL
        ORDER BY ship_date, po_num;
```

因为子查询产生的数取决于 main.ship_date（外部 SELECT 产生的一个值），所以该子查询是相关的。因此，必须对外部查询考虑的每一行重新执行子查询。

该查询使用 COUNT 函数来将值返回到主查询。然后，ORDER BY 子句对数据进行排序。查询找到并返回具有 10 个最新装运日期的 16 行，如下所示。

图: 查询结果

```
po_num      ship_date

        4745        06/21/1998
        278701      06/29/1998
        429Q        06/29/1998
        8052        07/03/1998
        B77897      07/03/1998
        LZ230       07/06/1998
        B77930      07/10/1998
        PC6782      07/12/1998
        DM354331    07/13/1998
        S22942      07/13/1998
        MA003       07/16/1998
        W2286       07/16/1998
        Z55709      07/16/1998
        C3288       07/25/1998
        KF2961      07/30/1998
        W9925       07/30/1998
```

如果对大型表使用相关子查询（如图 1），那么应对 ship_date 列建立索引以提高性能。否则，此 SELECT 语句效率降低，原因是它对表的每一行执行一次子查询。有关建立索引和性能问题的信息，请参阅《GBase 8s 管理员指南》 和 GBase 8s 性能指南 。

然而，不能在 FROM 子句中使用相关子查询，如下列无效示例所示：

```
SELECT item_num, stock_num FROM items,
        (SELECT stock_num FROM catalog
        WHERE stock_num = items.item_num) AS vtab;
```

该示例中的子查询具有错误 -24138：

```
ALL COLUMN REFERENCES IN A TABLE EXPRESSION MUST REFER
        TO TABLES IN THE FROM CLAUSE OF THE TABLE EXPRESSION.
```

数据库服务器发出该错误的原因是子查询中的 items.item_num 列还出现在外部查询的 Projection 子句中，但是内部查询的 FROM 子句仅指定catalog 表。错误消息文本中的术语表表达式指的是 FROM 子句中的子查询返回的列值或表达式集合。而在 FROM 子句中，只有不相关子查询才是有效的。

EXISTS 关键字

关键字 EXISTS 也被称为存在限定符，因为仅当外部 SELECT（如下所示）找到至少一行时，子查询才为 true 。

图: 查询

```
SELECT UNIQUE manu_name, lead_time
        FROM manufact
        WHERE EXISTS
        (SELECT * FROM stock
        WHERE description MATCHES '*shoe*'
        AND manufact.manu_code = stock.manu_code);
```

通常可使用 EXISTS 来构造等价于使用 IN 的查询的查询。下列查询使用 IN 谓词来构造与上述返回相同结果的查询。

图: 查询

```
SELECT UNIQUE manu_name, lead_time
        FROM stock, manufact
        WHERE manufact.manu_code IN
        (SELECT manu_code FROM stock
        WHERE description MATCHES '*shoe*')
        AND stock.manu_code = manufact.manu_code;
```

图 1和图 2返回生产某种鞋的制造商以及预订产品的交付周期的行。该结果显示了返回值。

图: 查询结果

```
manu_name       lead_time

        Anza             5
        Hero             4
        Karsten         21
        Nikolus          8
        ProCycle         9
        Shimara         30
```

将关键字 NOT 添加至 IN 或 EXISTS 以创建与前面查询相反的搜索条件。也可以用 !=ALL 代替 NOT IN。

下列查询显示了执行同一操作的两种方法。一种方法可能允许数据库服务器执行相对另一种方法较少的工作，则会取决于数据库的设计和表的大小。要了解哪一种查询更好，使用 SET EXPLAIN 命令来获取查询计划的清单。在 GBase 8s 性能指南 和 GBase 8s SQL 指南：语法 中讨论了 SET EXPLAIN。

图: 查询

```
SELECT customer_num, company FROM customer
        WHERE customer_num NOT IN
        (SELECT customer_num FROM orders
        WHERE customer.customer_num = orders.customer_num);

        SELECT customer_num, company FROM customer
        WHERE NOT EXISTS
        (SELECT * FROM orders
        WHERE customer.customer_num = orders.customer_num);
```

查询中的每个语句返回下列行，这些行标识尚未下订单的客户。

图: 查询结果

```
customer_num company

        102 Sports Spot
        103 Phil's Sports
        105 Los Altos Sports
        107 Athletic Supplies
        108 Quinn's Sports
        109 Sport Stuff
        113 Sportstown
```

```
        114 Sporting Place
        118 Blue Ribbon Sports
        125 Total Fitness Sports
        128 Phoenix University
```

关键字 EXISTS 和 IN 用于称为相交的集合运算，关键 NOT EXISTS 和 NOT IN 用于称为差异的集合运算。这些概念在集合运算中讨论。

下列查询执行对 items 表的子查询来标识 stock 表中尚未预订的所有商品。

图: 查询

```
SELECT * FROM stock
        WHERE NOT EXISTS
        (SELECT * FROM items
        WHERE stock.stock_num = items.stock_num
        AND stock.manu_code = items.manu_code);
```

该查询返回以下行。

图: 查询结果

```
stock_num manu_code description unit_price unit unit_descr

    101   PRC         bicycle tires       $88.00   box    4/box
    102   SHM         bicycle brakes      $220.00  case   4 sets/case
    102   PRC         bicycle brakes      $480.00  case   4 sets/case
    105   PRC         bicycle wheels      $53.00   pair   pair
    :
    312   HRO         racer goggles       $72.00   box    12/box
    313   SHM         swim cap            $72.00   box    12/box
    313   ANZ         swim cap            $60.00   box    12/box
```

对 SELECT 语句可具有的子查询数没有逻辑限制。

您可能想要检查是否在数据库中正确输入了信息。查找数据库中的错误的一种方法是编写仅当错误存在时才会返回输出的查询。这种类型的子查询充当一种审计查询，如下所示。

图: 查询

```
SELECT * FROM items
        WHERE total_price != quantity *
        (SELECT unit_price FROM stock
        WHERE stock.stock_num = items.stock_num
        AND stock.manu_code = items.manu_code);
```

该查询只返回订单上商品的总价格不等于库存单价乘以订单数量的行。如果没有应用任何折扣，那么可能在数据库中不正确地输入了此类型的行。仅当错误发生时查询才会返回行。如果正确地将信息插入到数据库中，那么不会返回任何行。

图: 查询结果

item_num	order_num	stock_num	manu_code	quantity	total_price
1	1004	1	HRO	1	$960.00
2	1006	5	NRG	5	$190.00

6.3.6 DELETE 和 UPDATE 语句中的子查询

除了在 SELECT 的 WHERE 子句中的子查询，还可以在其它数据操纵语言（DML）语句中使用子查询，包括 DELETE 和 UPDATE 的 WHERE 子句。

适用某些限制。如果子查询的 FROM 子句返回多行，并且该子句指定与其它 DML 语句正在修改相同的表或视图，那么处于下列情况下的 DML 语句会成功。

- DML 语句不能是 INSERT 语句。
- 子查询中的 SPL 例程没有引用正在被修改的表。
- 子查询不包括相关列名。
- 该子查询使用 DELETE 和 UPDATE 的 WHERE 子句中的子查询语法的条件指定。

如果这些条件中的任何条件都不符合，那么 DML 操作发生错误 -360。

以下示例修改 stock 表，通过增加价格子集的 10 % 来增加 unit_price 值。WHERE 子句通过将 IN 运算符应用到从 stock 表查找到 unit_price 值少于 75 而返回的行来增加价格。

```
UPDATE stock SET unit_price = unit_price * 1.1
        WHERE unit_price IN
        (SELECT unit_price FROM stock WHERE unit_price < 75);
```

6.4 处理 SELECT 语句中的集合

数据库服务器提供了下列 SQL 功能来处理集合表达式：

集合子查询

集合子查询采用虚拟表（子查询的结果）并将它转换为集合。

集合子查询总是返回类型 MULTISET 的集合。可使用集合子查询将关系数据库的查询结果转换为 MULTISET 集合。

集合派生的表

集合派生的表采用集合并将它转换为虚拟表。

将集合的每个元素构造成集合派生的表中的行。可以使用集合派生的表来访问集合的个别元素。

集合子查询和集合派生的表功能表示逆操作：集合子查询将关系表的行为转换为集合，而集合派生的表将集合的元素转换为关系表的行。

6.4.1 集合子查询

集合子查询使用户能够从子查询表达式构造集合表达式。集合子查询在紧邻子查询之前使用 MULTISET 关键字以将返回的值转换为 MULTISET 集合。但是，当在子查询表达式之前使用 MULTISET 关键字时，数据库服务器不会更改基础表的各行而只会修改这些行的副本。例如，如果将集合子查询传递至修改集合的用户定义的例程，那么会修改集合的副本而不会修改基础表。

集合子查询是可采用下列任何形式的表达式：

- MULTISET(SELECT expression1, expression2... FROM tab_name...)
- MULTISET(SELECT ITEM expression FROM tab_name...)

在集合子查询中省略 ITEM 关键字

如果在集合子查询表达式中省略 ITEM 关键字，那么集合子查询就是其元素类型始终为未命名的 ROW 类型的 MULTISET。未命名 ROW 类型的字段与在子查询的 Projection 子句中指定的表达式的数据类型相匹配。

假设您创建了包含类型为 MULTISET 的列的以下表：

```
CREATE TABLE tab2
        (
        id_num INT,
        ms_col MULTISET(ROW(a INT) NOT NULL)
        );
```

下列查询显示如何在 WHERE 子句中使用集合子查询来将子查询返回的 INT 值的行转换为类型为 MULTISET 的集合。在此示例中，数据库服务器在tab2 的 ms_col 列等于集合子查询表达式的结果时返回行。

图: 查询

```
SELECT id_num FROM tab2
        WHERE ms_col = (MULTISET(SELECT int_col FROM tab1));
```

该查询在集合子查询中省略了 ITEM 关键字，因此子查询返回的 INT 值类型为 MULTISET（ROW(a INT) NOT NULL）（它与 tab2 的 ms_col 列的数据类型相匹配）。

在集合子查询中指定 ITEM 关键字

当子查询的投影列表包含单个表达式时， 可以用 ITEM 关键字作为子查询的投影列表的开始以指定 MULTISET 的元素类型与子查询结果的数据类型相匹配。换言之，当包括 ITEM 关键字时，数据库服务器不在投影列表两端放置行包装器。例如：如果子查询（紧跟在 MULTISET 关键字之后）返回 INT 值，集合子查询具有类型 MULTISET（INT NOT NULL）。

假设您创建接受类型为 MULTISET（INT NOT NULL）的参数的函数 int_func()。下列查询显示将具有 INT 值的行转换为 MULTISET 并将集合子查询用作函数 int_func() 中的参数的集合子查询。

图: 查询

```
EXECUTE FUNCTION int_func(MULTISET(SELECT ITEM int_col
        FROM tab1
        WHERE int_col BETWEEN 1 AND 10));
```

该查询在子查询中包括 ITEM 关键字，因此将查询返回的 int_col 值转换为类型为 MULTISET（INT NOT NULL）的集合。没有 ITEM 关键字，集合子查询将返回类型为 MULTISET（ROW(a INT) NOT NULL）的集合。

FROM 子句的集合子查询

集合子查询在 SELECT 的 FROM 子句中有效，外部查询可使用子查询返回的值作为数据源。

集合子查询这一节中的查询示例通过使用 TABLE 关键字后面（括号内）跟 MULTISET 关键字然后跟子查询来指定集合子查询。该语法是对 SQL 语言的 ANSI/ISO 标准的 GBase 8s 扩展。

在（且仅在）SELECT 语句的 FROM 子句中，可以通过指定子查询、省略 TABLE 和 MULTISET 关键字和嵌套的括号来代替 SQL 的 ANSI/ISO 标准的语法，以指定集合子查询。

下列查询使用 GBase 8s 扩展语法连接外部查询的 FROM 子句中的两个集合子查询：

图: 查询

```
SELECT * FROM TABLE(MULTISET(SELECT SUM(C1) FROM T1 GROUP BY C1)),
        TABLE(MULTISET(SELECT SUM(C1) FROM T2 GROUP BY C2));
```

通过使用符合 ANSI/ISO 的语法来连接外部查询的 FROM 子句中的两个派生表，下列查询在逻辑上等价于上述返回相同结果的查询：

图: 查询

```
SELECT * FROM (SELECT SUM(C1) FROM T1 GROUP BY C1),
              (SELECT SUM(C1) FROM T2 GROUP BY C2);
```

该查询优于 TABLE(MULTISET(SELECT ...)) GBase 8s 扩展版本之处在于，任何支持 FROM 子句中符合 ANSI/ISO 语法的数据库服务器也可以执行该查询。有关集合子查询的语法和限制的更多信息，请参阅《GBase 8s SQL 指南：语法》。

6.4.2 集合派生的表

集合派生的表使您能都处理集合表达式的元素（例如虚拟表中的行）。在 SELECT 语句的 FROM 子句中使用 TABLE 关键字来创建集合派生的表。数据库服务器支持 SELECT 、INSERT 、UPDATE 和 DELETE 语句中的集合派生的表。

以下查询使用名为 c_table 的集合派生表访问 superstores_demo 数据库中 sales_rep 表的 sales 列的元素。sales 列是其中两个字段 month 和amount 存储销售数据的未命名行类型的集合。当 sales.month 等于 98-03 时，下列查询返回 sales.amount 的元素。由于内部选择本身就是表达式，所以它不能对外部查询的每个迭代返回多个列值。外部查询指定对 sales_rep 表的多少行进行求值。

图: 查询

```
SELECT (SELECT c_table.amount FROM TABLE (sales_rep.sales) c_table
              WHERE c_table.month = '98-03')
              FROM sales_rep;
```

图: 查询结果

```
(expression)


              $47.22
              $53.22
```

下列查询使用集合派生的表访问 sales 集合列中 rep_num 列等于 102 的元素。使用集合派生的表，可以为表和列指定列名。如果没有为集合派生的表指定表名，那么数据库服务器会自动创建表名。此示例为集合派生的表 c_table 指定派生列列表 s_month 和 s_amount。

图: 查询

```
SELECT * FROM TABLE((SELECT sales FROM sales_rep
              WHERE sales_rep.rep_num = 102)) c_table(s_month, s_amount);
```

图: 查询结果

```
s_month          s_amount


              1998-03          $53.22
```

```
1998-04          $18.22
```

下列查询创建集合派生的表但不指定派生表或派生列名。除派生列采用 sales_rep 表中的 sales 列的缺省自动名之外，该查询返回与图 3相同的结果。

图: 查询

```
SELECT * FROM TABLE((SELECT sales FROM sales_rep
        WHERE sales_rep.rep_num = 102));
```

图: 查询结果

```
month          amount

1998-03          $53.22
1998-04          $18.22
```

限制： 集合派生的表是只读的，因此它不能是 INSERT 、UPDATE 或 DELETE 语句的目标表或可更新游标或视图的基础表。

有关集合派生的表的语法和限制的完整描述，请参阅《GBase 8s SQL 指南：语法》。

6.4.3 用于集合派生表的符合 ISO 的语法

集合派生的表这一节中的查询示例通过使用 TABLE 关键字后面（括号内）跟 SELECT 语句来指定集合派生的表。该语法是对 SQL 语言的 ANSI/ISO 标准的 GBase 8s 扩展。

但是，在（且仅在）SELECT 的 FROM 子句中，可以通过指定子查询来代替使用 SQL 符合 ANSI/ISO 标准的语法，在不使用 TABLE 关键字或嵌套括号的情况下，来定义集合派生的表。

下列查询在逻辑上等价于 图 3，并为集合派生的表 c_table 指定拍摄了列表 s_month 和 s_amount。

图: 查询

```
SELECT * FROM (SELECT sales FROM sales_rep
        WHERE sales_rep.rep_num = 102) c_table(s_month, s_amount);
```

图: 查询结果

```
s_month          s_amount

1998-03          $53.22
1998-04          $18.22
```

如 GBase 8s 扩展语法中一样，声明派生表或其列的名称是可选的，而不是必需的。下列查询对外部查询的 FROM 子句使用符合ANSI/ISO 标准的语法，并产生与图 5相同的结果：

图: 查询

```
SELECT * FROM (SELECT sales FROM sales_rep
        WHERE sales_rep.rep_num = 102);
```

图: 查询结果

```
month          amount

        1998-03        $53.22
        1998-04        $18.22
```

6.5 集合运算

标准集合运算联合、相交和差异允许您处理数据库信息。这三种运算允许您使用 SELECT 语句在执行更新、插入和删除之后检查数据库的完整性。例如：当数据库传送值历史记录表，并且想要从原始表中删除数据之前验证历史记录表中的数据是否正确时，它们就非常有用。

6.5.1 联合

联合运算使用 UNION 运算符将两个查询组合成单个复合查询。可以在两个或多个 SELECT 语句之间使用 UNION 运算符来产生一个临时表，它包含存在于任何一个原始表或所有原始表中的行。还可以在视图的定义中使用 UNION 运算符。

不能在下列上下文的子查询内使用 UNION 运算符

- 在 SELECT 语句的 Projection 子句中
- 在 SELECT 、INSERT 、DELETE 或 UPDATE 语句的 WHERE 子句中

然而，UNION 运算符在 SELECT 语句的 FROM 子句中的子查询中是有效的，如下所示：

```
SELECT * FROM (SELECT col1 FROM tab1 WHERE col1 = 100) AS vtab1(c1),
        (SELECT col1 FROM tab2 WHERE col1 = 10
        UNION ALL
        SELECT col1 FROM tab1 WHERE col1 < 50 ) AS vtab2(vc1);
```

GBase 8s 不支持对 ROW 类型进行排序。由于 UNION 操作需要排序以除去重复值，所以当联合运算中的任一查询包括 ROW 类型数据时，不能使用 UNION 操作符。但是，数据库服务器确实支持具有 ROW 类型数据的 UNION ALL，这是因为此类运算无需排序。

下图举例说明了 UNION 集合运算。

图: 联合集合运算

UNION 关键字选择两个查询中的所有行，除去重复行并返回余下的行。因为查询的结果组合为一个结果，所以每个查询中的投影列表必须具有相同的列数。同时，从每个表选择的相应列必须包含兼容的数据类型（CHARACTER 数据类型列的长度必须相同），并且这些相应的列必须全部允许或全部不允许 NULL 值。

有关 SELECT 语句和 UNION 运算符的完整语法，请参阅《GBase 8s SQL 指南：语法》。有关特定于 GBase 8s ESQL/C 产品和涉及 INTO 子句和复合查询的任何限制的信息，请参阅《GBase 8s ESQL/C 程序员手册》。

下列查询对 stock 和 items 表中的 stock_num 和 manu_code 列执行联合。

图: 查询

```
SELECT DISTINCT stock_num, manu_code FROM stock
        WHERE unit_price < 25.00
        UNION
        SELECT stock_num, manu_code FROM items
        WHERE quantity > 3;
```

该查询选择单价小于 $25.00 或预订数量大于三的那些商品并列出
其 stock_num 和 manu_code，如下所示。

图: 查询结果

```
stock_num manu_code

        5 ANZ
        5 NRG
        5 SMT
        9 ANZ
        103 PRC
        106 PRC
        201 NKL
```

```
        301 KAR
        302 HRO
        302 KAR
```

将 ORDER BY 子句与 UNION 一起使用

如下列查询所示，当包括 ORDER BY 子句时，该子句必须跟在最后的 SELECT 语句后面，并使用整数（而不是标识）来引用排序的列。排序在集合运算完成之后发生。

图: 查询

```
SELECT DISTINCT stock_num, manu_code FROM stock
        WHERE unit_price < 25.00
        UNION
        SELECT stock_num, manu_code FROM items
        WHERE quantity > 3
        ORDER BY 2;
```

之前的复合查询选择与图 2相同的行但以制造商代码的顺序显示它们，如下所示。

图: 查询结果

```
stock_num manu_code

        5 ANZ
        9 ANZ
        302 HRO
        301 KAR
        302 KAR
        201 NKL
        5 NRG
        103 PRC
        106 PRC
        5 SMT
```

UNION ALL 关键字

缺省情况下，UNION 关键字排除重复的行。要保留重复值，添加可选关键字 ALL，如下所示。

图: 查询

```
SELECT stock_num, manu_code FROM stock
```

```
        WHERE unit_price < 25.00
        UNION ALL
        SELECT stock_num, manu_code FROM items
        WHERE quantity > 3
        ORDER BY 2
        INTO TEMP stock item;
```

该查询使用 UNION ALL 关键字联合两个 SELECT 语句并在最后一个 SELECT 后面添加 INTO TEMP 子句来将结果放置到临时列表中。它返回与图 1相同的行，但还包括重复的值。

图: 查询结果

```
stock_num manu_code

        9 ANZ
        5 ANZ
        9 ANZ
        5 ANZ
        9 ANZ
        ⋮
        5 NRG
        5 NRG
        103 PRC
        106 PRC
        5 SMT
        5 SMT
```

使用不同的列名

组合查询的 Projection 子句中相应的列必须具有兼容的数据类型，但各列不需要使用相同的列名。

下列查询从 customer 表中选择 state 列及 state 表中的相应 code 列。

图: 查询

```
SELECT DISTINCT state FROM customer
        WHERE customer_num BETWEEN 120 AND 125
        UNION
        SELECT DISTINCT code FROM state
        WHERE sname MATCHES '*a';
```

该查询返回客户号 120 至 125 的州代码缩写以及其 sname 以 a 结束的州的州代码缩写。

图: 查询结果

```
state

        AK
        AL
        AZ
        CA
        DE
        ⋮
        SD
        VA
        WV
```

在复合查询中，第一个 SELECT 语句中的列名或显示标注就是出现在结果中的列名或显示标注。因此，在此查询中，使用第一个 SELECT 语句中的列名 state 而不是第一个语句中的列名 code。

将 UNION 与多个表配合使用

以下查询对三个表执行联合运算。最大联合数取决于应用程序的实用性和任何内存限制。

图: 查询

```
SELECT stock_num, manu_code FROM stock
        WHERE unit_price > 600.00
        UNION ALL
        SELECT stock_num, manu_code FROM catalog
        WHERE catalog_num = 10025
        UNION ALL
        SELECT stock_num, manu_code FROM items
        WHERE quantity = 10
        ORDER BY 2;
```

该查询选择 stock 表中的 unit_price 大于 $600、catalog 表中的 catalog_num 为 10025 或 items 表中的 quantity 为 10 的商品；查询按manu_code 对数据进行排序。该结果显示了返回值。

图: 查询结果

```
stock_num manu_code
```

```
5 ANZ
9 ANZ
8 ANZ
4 HSK
1 HSK
203 NKL
5 NRG
106 PRC
113 SHM
```

在 Projection 子句中使用文字

下列查询在投影列表中使用文字来标记联合操作的部分输出，以便今后可以区分它。为该标记提供标号 sortkey。查询使用 sortkey 来对检索到的行进行排序。

图: 查询

```
SELECT '1' sortkey, lname, fname, company,
        city, state, phone
        FROM customer x
        WHERE state = 'CA'
        UNION
        SELECT '2' sortkey, lname, fname, company,
        city, state, phone
        FROM customer y
        WHERE state <> 'CA'
        INTO TEMP calcust;
        SELECT * FROM calcust
        ORDER BY 1;
```

该查询创建一个列表，在该列表中，来自 California 的客户首先显示。

图: 查询结果

```
sortkey  1
        lname     Baxter
        fname     Dick
        company   Blue Ribbon Sports
        city      Oakland
        state     CA
        phone     415-655-0011
```

```
sortkey   1
lname     Beatty
fname     Lana
company   Sportstown
city      Menlo Park
state     CA
phone     415-356-9982
⋮
sortkey   2
lname     Wallack
fname     Jason
company   City Sports
city      Wilmington
state     DE
phone     302-366-7511
```

使用 FIRST 子句

可以使用 FIRST 子句来选择由联合查询的产生的前几行。以下查询使用 FIRST 子句返回 stock 和 items 表之间的联合的前五行。

图: 查询

```
SELECT FIRST 5 DISTINCT stock_num, manu_code
        FROM stock
        WHERE unit_price < 55.00
        UNION
        SELECT stock_num, manu_code
        FROM items
        WHERE quantity > 3;
```

图: 查询结果

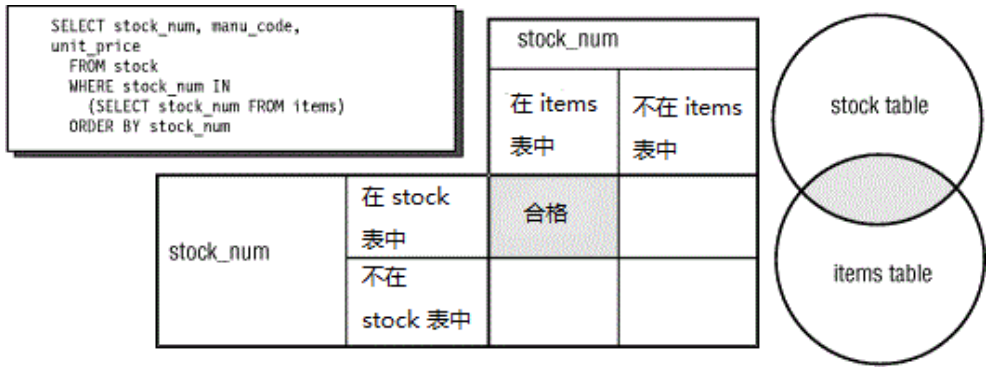
```
stock_num manu_code

        5 NRG
        5 ANZ
        6 SMT
        6 ANZ
```

```
9 ANZ
```

6.5.2 相交

两个行集的相交产生一个表。它包含同时存在两个原始表的行。使用关键字 EXISTS 或 IN 来引入显示两个集合相交的子查询。下图说明了相交集合运算。

图: 相交集合运算



以下查询是一个嵌套 SELECT 语句的示例，它显示了 stock 和 items 表的交集。该结果包含出现在这两个集合中的所有元素并返回以下行。

图: 查询

```
SELECT stock_num, manu_code, unit_price FROM stock
        WHERE stock_num IN
        (SELECT stock_num FROM items)
        ORDER BY stock_num;
```

图: 查询结果

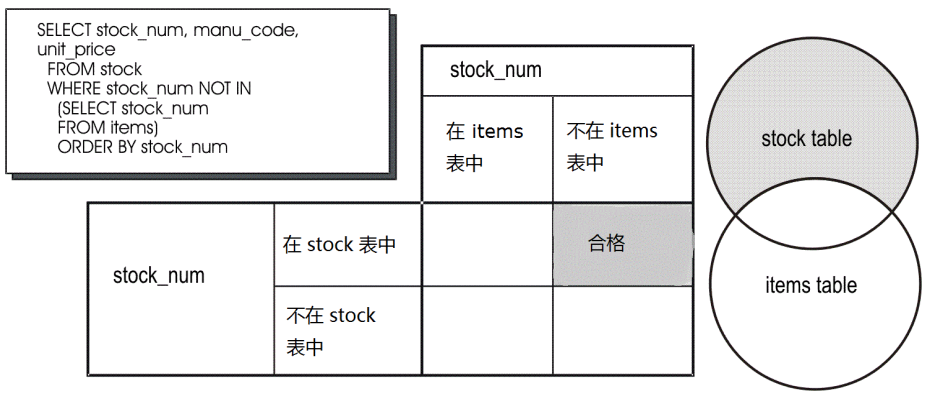
```
stock_num manu_code unit_price

        1 HRO          $250.00
        1 HSK          $800.00
        1 SMT          $450.00
        2 HRO          $126.00
        3 HSK          $240.00
        3 SHM          $280.00
        :
```

```
306 SHM          $190.00
307 PRC          $250.00
309 HRO           $40.00
309 SHM           $40.00
```

6.5.3 差异

两个行集之间的差异产生一个表，它包含在第一个行集中但不在第二个行集中的行。使用关键字 NOT EXISTS 或 NOT IN 俩引入显示两个集合之间的差异的子查询。下图说明了差异集合运算。

图: 差异集合运算



下列查询是嵌套 SELECT 语句的一个示例，它显示了 stock 和 items 表之间的差异。

图: 查询

```
SELECT stock_num, manu_code, unit_price FROM stock
        WHERE stock_num NOT IN
        (SELECT stock_num FROM items)
        ORDER BY stock_num;
```

该结果仅包含来自第一个集（它返回了 17 行）的所有元素。

图: 查询结果

```
stock_num manu_code unit_price

        102 PRC          $480.00
        102 SHM          $220.00
```

```
106 PRC          $23.00
⋮
312 HRO          $72.00
312 SHM          $96.00
313 ANZ          $60.00
313 SHM          $72.00
```

6.6 总结

本章是基于编写 SELECT 语句中介绍的概念构建的。它提供了更高级类型的 SELECT 语句（用来查询关系数据库）的样本语法和结果。本章提供了以下资料：

- 介绍了 GROUP BY 和 HAVING 子句，可将这些子句与聚集配合使用来返回行组并对那些组应用条件
- 显示如何使用自连接来将表连接至它本身以将列中的值与同一列中的其它值进行比较并标识重复
- 说明外连接如果区别看待两个或多个表，并提供了使用 GBase 8s 扩展和 ANSI 连接语法的四种外连接类型的示例
- 描述如何在另一 SELECT 语句的 WHERE 子句中嵌套 SELECT 语句，来创建相关和非相关查询并显示如何在子查询中使用聚集函数
- 演示如何将 SELECT 语句嵌套在另一个 SELECT 语句的 FROM 子句中，来指定其结果作为外部 SELECT 语句数据源的不相关子查询
- 演示如何使用关键字 ALL 、ANY 、EXISTS 、IN 和 SOME 来创建子查询以及添加关键字 NOT 或关系运算符的影响
- 描述如何使用集合子查询将关系数据转换为 MULTISET 类型的集合，以及如何使用集合派生的表反访问集合中的元素
- 讨论联合、相交和差异集合运算
- 显示任何使用 UNION 和 UNION ALL 关键字创建包含两个或多个 SELECT 语句的复合查询

7 修改数据

本部分描述如何修改数据库中的数据。修改数据与查询数据有本质区别。查询数据涉及检查表的内容。修改数据涉及更改表的内容。

7.1 修改数据库中的数据

下列语句修改数据：

- DELETE
- INSERT
- MERGE
- UPDATE

当与更高级的 SELECT 语句相比时，虽然这些 SQL 语句相对简单，但由于它们更改数据库的内容，因此请小心使用它们。

如果在查询期间系统硬件或软件出现故障，请考虑会发生什么。即使对应用程序的影响是严重的，也不会破坏数据库自身。然而，如果正在进行修改时系统发生故障，则数据库的状态就不确定了。显然，处于不确定状态的数据库具有深远的影响。在数据库中删除、插入或更新行之前，请询问自己下列问题：

- 用户对数据库及其表的访问是否安全。即，是否将有限的数据库和表级别权限授予特定用户？
- 修改了的数据是否保持数据库现有的完整性？
- 系统的状况是否使其对可能导致系统或硬件故障的外部事件具有相对较强的免疫力？

如果对这些问题不能都回答“是”，也不用担心。对所有这些问题的解决方案都内建在 GBase 8s 数据库服务器内。在对修改数据的语句进行描述之后，这部分讨论这些解决方案。

7.2 删除行

DELETE 从表中移除任何行或行的组合。在提交该事务之后，您不可恢复删除了的行。（在 中断了的修改 之下讨论事务。现在，请将事务与语句看做是一回事。）

当删除一行时，您还必须小心地删除其值依赖于该删除了的行的其他表的任何行。如果数据库强制执行引用约束，则您可使用 CREATE TABLE 或 ALTER TABLE 语句的 ON DELETE CASCADE 选项来允许从与另一表的关系中的一个表进行级联删除。要获取关于引用约束和 ON DELETE CASCADE 选项的更多信息，请参阅 引用完整性。

7.2.1 删除表的所有行

DELETE 语句指定表并通常包含 WHERE 子句，该子句指定要从表中移除的一行或多行。如果省略 WHERE 子句，则删除所有行。

重要： 请不要执行下列语句。

```
DELETE FROM customer;
```

您可编写带有或不带 FROM 关键字的 DELETE 语句。

```
DELETE customer;
```

由于这些 DELETE 语句不包含 WHERE 子句，因此从 customer 表删除所有行。如果您尝试使用 DB-Access 菜单选项来进行无条件的删除，则程序会警告您并要求确认。然而，从程序之内执行无条件的 DELETE 可在不发出警告的情况下发生。

如果想要从名为 from 的表中删除行，则您必须首先设置 DELIMIDENT 环境变量，或使用其所有者的名称来限定该表的名称：

```
DELETE legree.from;
```

要获取关于定界的标识符以及 DELIMIDENT 环境变量的更多信息，请参阅《GBase 8s SQL 指南：语法》 中对“带引号的字符串”表达式以及“标识符”段的描述。

7.2.2 使用 TRUNCATE 来删除所有行

您可使用 TRUNCATE 语句来快速地从表中移除所有行，同时还移除所有对应的索引数据。在提交该事务之后，您不可恢复删除了的行。您可对包含任何列类型（包括智能大对象）的表上使用 TRUNCATE 语句。

使用 TRUNCATE 语句来移除行的速度比使用 DELETE 语句来移除它们快。在 TRUNCATE 语句之后，不必立即运行 UPDATE STATISTICS 语句。成功地执行 TRUNCATE 之后，GBase 8s 自动地更新该表及其系统目录中的索引的统计信息和分布情况，以展示在该表中或在它的 dbspace 分区中没有任何行。

要了解日志记录的描述，请参阅 事务日志记录。

TRUNCATE 是数据定义语言语句，如果在该表上定义任何触发器，则该语句不激活 DELETE 触发器。要了解关于使用触发器的说明，请参阅 创建和使用触发器。

如果 TRUNCATE 语句指定的表是 typed 表，则成功的 TRUNCATE 操作从那个表中以及从该表层级结构内的所有子表中移除所有行和 B-tree 结构。TRUNCATE 不等同于 DELETE 语句的 ONLY 关键字，DELETE 语句将操作限制在 typed 表层级结构内的单个表。

GBase 8s 始终对 TRUNCATE 操作进行日志记录，即使对非日志记录的表也是如此。在支持事务日志记录的数据库中，在同一事务之内的 TRUNCATE 之后，仅 SQL 的 COMMIT WORK 或 ROLLBACK WORK 语句是有效的。要获取关于使用 TRUNCATE 语句对性能的影响的信息，请参阅《GBase 8s 性能指南》。要了解完整的语法，请参阅《GBase 8s SQL 指南：语法》。

7.2.3 删除指定的行

DELETE 语句中的 WHERE 子句与 SELECT 语句中的 WHERE 子句的形式相同。您可使用它来准确地指定应删除哪一行或哪些行。您可删除带有特定客户编号的客户，如下例所示：

```
DELETE FROM customer WHERE customer_num = 175;
```

在此示例中，由于 customer_num 列有唯一约束，因此您可确保只删除一行。

7.2.4 删除选择了的行

您还可选取基于非索引列的行，如下例所示：

```
DELETE FROM customer WHERE company = 'Druid Cyclery';
```

由于被测试的列没有唯一约束，因此此语句可能删除多行。（Druid Cyclery 可能有两个商店，两个商店的名称相同但客户编号不一样。）

要了解 DELETE 语句影响多少行，请从 customer 表中为 Druid Cyclery 选择符合条件的行计数。

```
SELECT COUNT(*) FROM customer WHERE company = 'Druid Cyclery';
```

您还可选择这些行并显示它们，以确保它们是您想要删除的那些行。

然而，当数据库对于多个用户同时可用时，使用 SELECT 语句作为测试只是一种近似的方法。在您执行 SELECT 语句与后续的 DELETE 语句之间的时间内，其他用户可能已修改了该表并更改了结果。在此示例中，另一用户可能执行下列操作：

- 为名为 Druid Cyclery 的另一客户插入新行
- 在插入新行之前，删除一个或多个 Druid Cyclery 行
- 更新 Druid Cyclery 行以具有新的公司名称，或更新某个其他客户以具有名称 Druid Cyclery。

在这短短的时间间隔内，虽然其他用户不太可能执行这些操作，但确实存在这种可能性。相同的问题也影响 UPDATE 语句。在 并发和锁定 之下讨论解决此问题的方法，且在 对多用户环境编程 中讨论得更详细。

您可能遇到的另一个问题是，在该语句完成之前出现硬件或软件故障。在此情况下，数据库可能还没删除行，可能已删除了一些行，或已经删除了所有指定的行。数据库的状态未知，这是我们不想看到的。要防止此情况，请使用事务日志记录，如 中断了的修改 讨论的那样。

7.2.5 删除包含 row 类型的行

当某行包含定义在 ROW 类型上的列时，您可使用点符号表示法来指定仅删除那些包含特定字段值的行。例如，下列语句仅从 employee 表中删除address 列中的 city 字段的值为 San Jose 的那些行：

```
DELETE FROM employee
        WHERE address.city = 'San Jose';
```

在前面的语句中，address 列可能是命名的 ROW 类型或未命名的 ROW 类型。您用来指定 ROW 类型的字段值的语法是相同的。

7.2.6 删除包含集合类型的行

当某行包含定义在集合类型上的列时，您可在集合中搜索特定的元素，并删除在其中找到那个元素的一行或多行。例如，下列语句删除其中的direct_reports 列包含带有元素 Baker 的集合的那些行：

```
DELETE FROM manager
        WHERE 'Baker' IN direct_reports;
```

7.2.7 从超级表中删除行

当您删除超级表的各行时，删除操作的作用域是超级表及其子表。假设您创建超级表 person，在其下定义两个子表 employee 和 sales_rep。下列对 person 表执行的 DELETE 语句可从 person、employee 和 sales_rep 全部三个表中删除行：

```
DELETE FROM person
        WHERE name ='Walker';
```

要限制为仅删除超级表的行，您必须使用 DELETE 语句中的 ONLY 关键字。例如，下列语句仅删除 person 表的行：

```
DELETE FROM ONLY(person)
        WHERE name ='Walker';
```

重要： 当您从超级表中删除行时，请小心使用，因为对超级表的删除的作用域包括该超级表及其所有子表。

7.2.8 复杂的删除条件

DELECT 语句中的 WHERE 子句可与 SELECT 语句中的一样复杂。它可包含通过 AND 和 OR 连接的多个条件，且它可能包含子查询。

假设您发现 stock 表的某些行包含不正确的制造商代码。您不想更新它们，而是想要删除它们以便重新输入它们。您知道，与正确的那些行不一样，这些行在 manufact 表中没有相匹配的行。由于这些不正确的行在 manufact 表中没有相匹配的行，因此您可以编写下例中所示的 DELETE 语句：

```
DELETE FROM stock
        WHERE 0 = (SELECT COUNT(*) FROM manufact
```

```
        WHERE manufact.manu_code = stock.manu_code);
```

该子查询对匹配的 manufact 行数进行计数；对 stock 的正确的行计数为 1，对不正确的行计数为 0。选取不正确的行来删除。

提示： 使用复杂的条件来开发 DELETE 语句的一种方法是，先开发精确地返回要删除的行的 SELECT 语句。将它编写为 SELECT *；当它返回所期望的行集时，将SELECT * 更改为读取 DELETE，并再执行它一次。

DELETE 语句的 WHERE 子句不可使用测试同一表的子查询。即，当您从 stock 进行删除时，您不可在也从 stock 中选择的 WHERE 子句中使用子查询。

此规则的关键在于 FROM 子句。如果在 DELETE 语句的 FROM 子句中命名表，则该表不可还出现在 DELECT 语句的子查询的 FROM 子句中。

7.2.9 MERGE 的 Delete 子句

不在 WHERE 子句中编写子查询，您可使用 MERGE 语句将来自源表和目标表的行连接在一起，然后从目标删除与连接条件相匹配的那些行。（Delete MERGE 中的源表还可为一个集合派生的表，它的行是查询的结果，该查询连接其他的表和视图，但是在下列的示例中，源是单个表。）

如在前面的示例中那样，假设您发现 stock 表的某些行包含不正确的制造商代码。您想要删除它们以便重新输入它们，而不是更新它们。您可使用 MERGE 语句，指定 stock 作为目标表，manufact 作为源表，ON 子句中的连接条件，并对于带有不正确的制造商代码的 stock 行使用 Delete 子句，如下例所示：

```
MERGE INTO stock USING manufact
        ON stock.manu_code != manufact.manu_code
        WHEN MATCHED THEN DELETE;
```

在此示例中，会从 stock 表中删除那些满足 ON 子句中的连接条件的所有行。在此，对于其中的 manu_code 列值不等于 manufact 中的任何manu_code 值的 stock 的那些行，连接条件中的不等谓词（stock.manu_code != manufact.manu_code）求值为真。

在 USING 子句中必须罗列正连接到目标表的源表。

MERGE 语句还可更新目标表的行，或将数据从源表插入到目标表，根据该行是否满足 ON 子句为连接目标表与源表而指定的条件。单个 MERGE 语句还可同时组合 DELETE 与 INSERT 操作，或者可同时组合 UPDATE 与 INSERT 操作而不删除任何行。MERGE 语句不更改源表。要获取关于 Delete 合并、Insert 合并和 Update 合并的语法与限制的更多信息，请参阅《GBase 8s SQL 指南：语法》 中 MERGE 语句的描述。

7.3 插入行

INSERT 将新的一行或多行添加到表。该语句有两个基本功能。它可使用您提供的列值创建单个的新行，或可使用从其他表选择的数据创建一组新的行。

7.3.1 单个行

在它的最简单形式中，INSERT 语句从一列值的列表创建一个新行，并将其放置在表中。下列语句展示如何将一行添加到 stock 表：

```
INSERT INTO stock
        VALUES (115, 'PRC', 'tire pump', 108, 'box', '6/box');
```

stock 表有下列列：

stock_num

标识商品的种类的编号。

manu_code

manufact 表的外键。

description

该商品的描述。

unit_price

该商品的单价。

unit

计量的单位

unit_descr

说明计量单位的特征。

前一示例中 VALUES 子句中罗列的值与 stock 表的列有一一对应关系。要编写 VALUES 子句，您必须知道表的列以及它们的前后次序。

可能的列值

VALUES 子句仅接受常量值，不接受通用的 SQL 表达式。您可提供下列值：

- 文字数值
- 文字 DATETIME 值
- 文字 INTERVAL 值
- 带引号的字符串
- 表示 NULL 的关键字 NULL

- 表示当前日期的关键字 TODAY
- 表示当前日期和时间的关键字 CURRENT（或 SYSDATE）
- 表示您的授权标识符的关键字 USER
- 表示正在运行数据库服务器的计算机名称的关键字 DBSERVERNAME（或 SITENAME）

注： MERGE 语句可以替代 INSERT 语句，可使用与 INSERT 语句一样的 VALUES 子句语法来将行插入到表内。MERGE 语句执行源表与目标表的外部链接，然后将连接的结果集中的任何行插入到目标表内，这些行的连接谓词求值为 FALSE。MERGE 语句不更改源表。除了插入行之外，MERGE 语句可可选地同时组合 DELETE 与 INSERT 操作，或同时组合 UPDATE 与 INSERT 操作。要获取关于 Insert 合并、Delete 合并和 Update 合并的语法与限制的更多信息，请参阅《GBase 8s SQL 指南：语法》 中 MERGE 语句的描述。

对列值的限制

表的某些列可能不允许空值。如果您尝试向这样的列插入 NULL，则会拒绝该语句。表中的其他列可能不允许重复的值。如果您指定与这样的列中已经存在的值重复的值，则会拒绝该语句。有些列甚至可能限制允许的列值。请使用数据完整性约束来限制列。要获取更多信息，请参阅 数据完整性。

限制： 请不要为包含货币值的列指定币种符号。请仅指定该金额的数值值。

数据库服务器可在数值与字符数据类型之间进行转换。您可将数值字符的字符串（例如，'-0075.6'）作为数值列的值。数据库服务器将数值字符串转换为数值。仅当该字符串不表示数值时才会发生错误。

您可指定数值或日期作为字符列的值。数据库服务器将那个值转换为字符串。例如，如果您指定 TODAY 作为字符列的值，则使用表示当前日期的字符串。（DBDATE 环境变量指定所使用的格式。）

序列数据类型

表仅可有一个 SERIAL 数据类型的列。它还可有 SERIAL8 列或 BIGSERIAL 列。

当您插入值时，请为序列列指定值零。数据库服务器按次序生成下一个实际值。序列列不允许 NULL 值。

您可为序列列指定非零值（只要它不与那一列中任何现有的值重复），数据库服务器使用该值。那个非零值可能为数据库服务器生成的值设置新的起始点。（数据库服务器为您生成的下一个值是比该列中最大值大一的值。）

罗列特定的列名称

您不必为每列都指定值。相反，您可在表名称之后罗列列名称，然后仅为您命名了的那些列提供值。下列示例展示将新行插入到 stock 表内的语句：

```
INSERT INTO stock (stock_num,  description,  unit_price,  manu_code)
            VALUES (115,  'tyre pump ',  114,  'SHM');
```

仅提供库存编号、描述、单价和制造商代码的数据。数据库服务器为其余列提供下列值：

- 它为未列出的序列列生成一个序列数值。
- 它为与它相关联的有特定缺省值的列生成一个缺省值。
- 它为任何允许空值的任何列生成 NULL 值，但它不为指定 NULL 作为缺省值的任何列指定缺省值。

您必须为未指定缺省值或不允许 NULL 值的所有列罗列并提供值。

您可以任何顺序罗列列，只要这些列的值也以相同的顺序罗列。

在执行前一示例中的 INSERT 语句之后，将下列新行插入到 stock 表内：

```
stock_num manu_code   description   unit_price unit unit_descr


          115         SHM   tyre pump           114
```

unit 和 unit_descr 都为空，表示在那两列中存在 NULL 值。由于 unit 列允许 NULL 值，因此不知道 114 美元可购买的轮胎充气泵（tire pump）的数目。当然，如果为这一列指定了 box 缺省值，则计量单位将为 box。在任何情况下，当您将值插入到表的特定的列内时，请注意那一行需要什么数据。

7.3.2 将行插入到类型的表中

您可使用与将行插入到不基于 ROW 类型的表内的相同方法，将行插入到类型的表内。

当类型的表包含一 row 类型列（定义该类型的表的命名了的 ROW 类型包含嵌套的 ROW 类型）时，您插入到 row 类型列的方法，与为不基于 ROW 类型的表插入 row 类型列的方法相同。下列部分，在列上插入的语法规则，描述如何执行插入到 row 类型列内。

此部分为示例使用 row 类型 zip_t、address_t 和 employee_t，以及 类型的表 employee。下图展示创建 row 类型和表的 SQL 语法。

图: 创建 row 类型和表的 SQL 语法。

```
CREATE ROW TYPE zip_t
            (
            z_code      CHAR(5),
            z_suffix    CHAR(4)
            );
```

```
CREATE ROW TYPE address_t
(
street      VARCHAR(20),
city        VARCHAR(20),
state       CHAR(2),
zip         zip_t
);

CREATE ROW TYPE employee_t
(
name         VARCHAR(30),
address     address_t,
salary      INTEGER
);

CREATE TABLE employee OF TYPE employee_t;
```

7.3.3 在列上插入的语法规则

下列语法规则适用于那些定义在命名了的 ROW 类型或未命名的 ROW 类型上的列上的插入：

在要插入字段值之前，指定 ROW 构造函数。

将 ROW 类型的字段值括在圆括号中。

将 ROW 表达式强制转型为适当的命名了的 ROW 类型（对于命名了的 ROW 类型）。

包含命名了的 row 类型的行

下列语句展示您如何将一行插入到在 图 1 中的 employee 内：

```
INSERT INTO employee
        VALUES ('Poole, John',
        ROW('402 High St', 'Willits', 'CA',
        ROW(69055,1450))::address_t, 35000 );
```

由于 employee 表的 address 列是命名了的 ROW 类型，因此您必须使用强制转型运算符和 ROW 类型的名称（address_t）来插入类型 address_t的值。

包含未命名的 row 类型的行

假设您创建下图所示的表。student 表定义 s_address 列为一未命名的 row 类型。

图: 创建 student 表。

```
CREATE TABLE student
        (
        s_name      VARCHAR(30),
        s_address   ROW(street VARCHAR (20), city VARCHAR(20),
        state CHAR(2), zip VARCHAR(9)),
        grade_point_avg DECIMAL(3,2)
        );
```

下列语句展示您如何向 student 表添加一行。要插入到未命名的 row 类型列 s_address 内，请使用 ROW 构造函数，但不要对该 row 类型值进行强制转型。

```
INSERT INTO student
        VALUES ('Keene, Terry',
        ROW('53 Terra Villa', 'Wheeling', 'IL', '45052'),
        3.75);
```

为 row 类型指定 NULL 值

row 类型列的字段可包含 NULL 值。您可在列级别或在字段级别指定 NULL 值。

下列语句在列级别指定 NULL 值，来为 s_address 列的所有字段插入 NULL 值。当您在列级别插入 NULL 值时，请不要包括 ROW 构造函数。

```
INSERT INTO student VALUES ('Brauer, Howie', NULL, 3.75);
```

当您为 ROW 类型的特定字段插入 NULL 值时，必须包括 ROW 构造函数。下列 INSERT 语句展示您可以如何将 NULL 值插入到 employee 表的address 列的特定字段内。（address 列被定义为命名了的 ROW 类型。）

```
INSERT INTO employee
        VALUES (
        'Singer, John',
        ROW(NULL, 'Davis', 'CA',
        ROW(97000, 2000))::address_t, 67000
        );
```

当您为 ROW 类型的字段指定 NULL 值时，当该 ROW 类型出现在 INSERT 语句、UPDATE 语句或程序变量赋值中时，无需显式地强制转型该 NULL 值。

下列 INSERT 语句展示您如何为 student 表的 s_address 列的 street 和 zip 字段插入 NULL 值：

```
INSERT INTO student
        VALUES(
```

```
'Henry, John',
ROW(NULL, 'Seattle', 'WA', NULL), 3.82
);
```

7.3.4 将行插入到超级表内

当您将行插入到超级表内时，不存在特殊的注意事项。INSERT 语句仅适用于在该语句中指定的表。例如，下列语句将值插入到超级表内，但不将值插入到任何子表内：

```
INSERT INTO person
        VALUES (
        'Poole, John',
        ROW('402 Saphire St.', 'Elmondo', 'CA', '69055'),
        345605900
        );
```

7.3.5 将集合值插入到列内

此部分描述如何使用 DB-Access 将集合值插入到列内。它未讨论如何将个别元素插入到集合列内。要访问或修改集合的个别元素，请使用 GBase 8s ESQL/C 程序或 SPL 例程。要获取关于如何创建 GBase 8s ESQL/C 程序来插入到集合内的信息，请参阅《GBase 8s ESQL/C 程序员手册》。要获取关于如何创建 SPL 例程来插入到集合内的信息，请参阅 创建和使用 SPL 例程。

本部分提供的这些示例是基于下图中的 manager 表。manager 表同时包含简单的和嵌套的集合类型。

图: 创建 manager 表。

```
CREATE TABLE manager
        (
        mgr_name            VARCHAR(30),
        department          VARCHAR(12),
        direct_reports    SET(VARCHAR(30) NOT NULL),
        projects            LIST(ROW(pro_name VARCHAR(15),
        pro_members SET(VARCHAR(20) NOT NULL))
        NOT NULL)
        );
```

将值插入到简单的集合和嵌套的集合内

当您将值插入包含集合列的行内时，您插入集合列包含的所有元素的值以及其他列的值。例如，下列语句将单个行插入到 manager 表内，该表同时包括简单的集合和嵌套的集合列:

```
INSERT INTO manager(mgr_name, department,
        direct_reports, projects)
        VALUES
        (
        'Sayles', 'marketing',
        "SET{'Simonian', 'Waters', 'Adams', 'Davis', 'Jones'}",
        LIST{
        ROW('voyager_project', SET{'Simonian', 'Waters',
        'Adams', 'Davis'}),
        ROW ('horizon_project', SET{'Freeman', 'Jacobs',
        'Walker', 'Smith', 'Cannan'}),
        ROW ('saphire_project', SET{'Villers', 'Reeves',
        'Doyle', 'Strongin'})
        }
        );
```

将 NULL 值插入到包含 row 类型的集合内

要将值插入到 ROW 类型的集合内，您必须为 ROW 类型中的每一字段指定值。

通常，在集合中不允许 NULL 值。然而，如果集合的元素类型为 ROW 类型，则您可将 NULL 值插入到 row 类型的个别字段内。

您还可指定空集合。空集合是不包含任何元素的集合。要指定空集合，请使用大括号（{}）。例如，下列语句将数据插入到 manager 表中的行内，但指定 direct_reports 和 projects 列为空集合：

```
INSERT INTO manager
        VALUES ('Sayles', 'marketing', "SET{}",
        "LIST{ROW(NULL, SET{})}"
        );
```

集合列不可包含 NULL 元素。由于指定 NULL 值作为集合的元素，因此下列语句返回一个错误：

```
INSERT INTO manager
        VALUES ('Cole', 'accounting', "SET{NULL}",
        "LIST{ROW(NULL, ""SET{NULL}"")}"
```

下列语法规则适用于对集合类型执行插入和更新：

使用大括号（{}）来划分每一集合包含的元素。

如果该集合为嵌套的集合，则使用大括号（{}）来同时划分内部集合和外部集合的元素。

7.3.6 插入智能大对象

当您使用 INSERT 语句来将对象插入到 BLOB 或 CLOB 列时，数据库服务器在 sbspace 中，而不是在表中，存储该对象。数据库服务器提供您可从 INSERT 语句之内调用的 SQL 函数来导入和导出 BLOB 或 CLOB 数据，这些数据又称为智能大对象。要了解这些函数的描述，请参阅 智能大对象函数。

下列 INSERT 语句使用 filetoblob() 和 filetoclob() 函数来插入 inmate 表的行。（图 1 定义 inmate 表。）

```
INSERT INTO inmate
        VALUES (437, FILETOBLOB('datafile', 'client'),
        FILETOCLOB('tmp/text', 'server'));
```

在前一示例中，FILETOBLOB() 和 FILETOCLOB() 函数的第一个参数分别指定要复制到 inmate 表的 BLOB 和 CLOB 列内的源文件的路径。每一函数的第二个参数指定该源文件是位于客户机计算机（'client'），还是位于服务器计算机（'server'）。要在该函数参数中指定文件名称的路径，请应用下列规则：

1. 如果源文件驻留在服务器计算机上，则您必须指定该文件的完全路径名称（不是相对于当前工作目录的路径名称）。
2. 如果源文件驻留在客户机计算机上，则您可指定该文件的完全路径或相对路径。

7.3.7 多个行和表达式

INSERT 语句的其他主要形式以 SELECT 语句替代 VALUES 子句。此特性允许您插入下列数据：

仅用一条语句处理多个行（每当 SELECT 语句返回一行，就插入一行）

计算值（VALUES 子句仅允许常量），由于 projection 列表可包含表达式

例如，假设对于已付款但尚未装运的每个订单都需要电话跟进。下列示例中的 INSERT 语句找到那些订单并为每一订单在 cust_calls 中插入一行：

```
INSERT INTO cust_calls (customer_num, call_descr)
        SELECT customer_num, order_num FROM orders
        WHERE paid_date IS NOT NULL
        AND ship_date IS NULL;
```

此 SELECT 语句返回两列。将来自这些列的数据（在每一选择了的行中）插入到 cust_calls 表的命名了的列内。然后，将（来自 SERIAL 列order_num 的）订单编号插入

到呼叫描述，这是一个字符列。请记住，数据库服务器允许您将整数值插入到字符列内。它自动地将序列编号转换为十进制数字的字符串。

7.3.8 对插入选择的限制

下列列表包含对插入行的 SELECT 语句的限制：

- 它不可包含 INTO 子句。
- 它不可包含 INTO TEMP 子句。
- 它不可包含 ORDER BY 子句。
- 它不可引用您正在向其内插入行的表。

INTO、INTO TEMP 和 ORDER BY 子句限制较小。在此上下文中，INTO 子句没什么用处。（要获取更多信息，请参阅 SQL 编程。）要绕开 INTO TEMP 子句限制，请先选择您想要插入到临时表内的数据，然后使用 INSERT 语句从临时表插入该数据。同样，缺少 ORDER BY 子句也无关紧要。如果您需要确保这些新行在表中物理地排序，则您可首先将它们选择到临时表内并对它排序，然后从该临时表插入。在所有插入完成之后，您还可使用集群的索引来对该表进行物理排序。

重要： 最后一个限制更为严重，因为它同时阻止在 INSERT 语句的 INTO 子句与 SELECT 语句的 FROM 子句中命名同一个表。同时在 INSERT 语句的 INTO 子句与 SELECT 语句的 FROM 子句中命名同一个表，导致数据库服务器进入无限循环，每一插入了的行都会在其中被重新选择和重新插入。

然而，在某些情况下，您可能想要从您必须向其内插入数据的同一个表进行选择。例如，假设您已了解 Nikolus 公司与 Anza 公司供应相同的产品，但仅以一半的价格供应。您想要向 stock 表添加一些行来反映两家公司之间的差异。理想情况下，您想要从所有 Anza 库存行选择数据，并使用 Nikolus 制造商代码重新插入它。然而，您不可从您正在向其内插入的同一个表进行选择。

要避开此限制，请选择您想要插入到临时表内的数据。然后在 INSERT 语句中从那个临时表进行选择，如下例所示：

```
SELECT stock_num, 'NIK' temp_manu, description, unit_price/2
        half_price, unit,   unit_descr FROM stock
        WHERE manu_code = 'ANZ'
        AND stock_num < 110
        INTO TEMP anzrows;

        INSERT INTO stock SELECT * FROM anzrows;

        DROP TABLE anzrows;
```

此 SELECT 语句从 stock 得到现有的行，并替换制造商代码的文字值以及单价的计算得到的值。然后将这些行保存在临时表 anzrows 中，立即将该表插入到 stock 表内。

当您插入多个行时，存在一种风险：其中一行包含无效的数据，就可能导致数据库服务器报告一个错误。当发生这样一个错误时，该语句提早终止。即使未发生错误，也存在一个小风险：在执行该语句时可能发生硬件或软件故障（例如，磁盘可能写满）。

在任何一种事件中，您都不可轻易地知道插入了多少新行。如果全部重复该语句，则您可能创建重复的行，也可能不会。由于数据库处于未知状态，因此您无所适从。解决方案在于使用事务，如 中断了的修改 讨论的那样。

7.4 更新行

根据 SET 子句的规范，使用 UPDATE 语句来更改表的一个或多个现有行的内容。此语句采用两种根本不同的形式。一种允许您按名称将特定的值指定给列；另一种允许您将（可能是通过 SELECT 语句返回的）值的列表指定给列的列表。在任一情况下，如果您正在更新行，且某些列有数据完整性约束，则您更改的数据必须符合对那些列的限制。要获取更多信息，请参考 数据完整性。

注： MERGE 语句是 UPDATE 语句的一种替代，可使用与 UPDATE 语句一样的 SET 子句语法来修改表的现有行中的一个或多个值。MERGE 语句执行源表与目标表的外部连接，然后以来自于连接的结果集的值更新目标表中的行，其连接谓词求值为 TRUE。MERGE 语句不更改源表中的值。除了更新行之外，MERGE 语句可可选地同时组合 UPDATE 与 INSERT 操作，或可同时组合 DELETE 与 INSERT 操作而不更新任何行。要获取关于 Update 合并、Delete 合并和 Insert 合并的语法和限制的更多信息，请参阅《GBase 8s SQL 指南：语法》 中 MERGE 语句的描述。

7.4.1 选择要更新的行

UPDATE 语句的任一形式都可以确定修改那些行的 WHERE 子句结尾。如果您省略 WHERE 子句，则修改所有行。要选择在 WHERE 子句中需要更改的精确行集可能非常复杂。对 WHERE 子句的唯一限制是，不可在子查询的 FROM 子句中命名您更新的表。

UPDATE 语句的第一种形式是，使用一系列赋值子句来指定新的列值，如下例所示：

```
UPDATE customer
        SET fname = 'Barnaby', lname = 'Dorfler'
        WHERE customer_num = 103;
```

WHERE 子句选择您想要更新的行。在演示数据库中，customer.customer_num 列是那个表的主键，因此，此语句最多可更新一行。

您还可在 WHERE 子句中使用子查询。假设 Anza 公司对他们的网球发出安全召回。结果是，包括来自制造商 ANZ 的库存编号 6 的任何未装运的订单都必须设定为延期交货，如下例所示：

```
UPDATE orders
        SET backlog = 'y'
        WHERE ship_date IS NULL
        AND order_num IN
        (SELECT DISTINCT items.order_num FROM items
        WHERE items.stock_num = 6
        AND items.manu_code = 'ANZ');
```

此子查询返回一订单编号（零个或多个）的列。然后，该 UPDATE 操作针对该列表测试 orders 的每一行，如果那一行相匹配，则执行更新。

7.4.2 以统一值进行更新

关键字 SET 之后的每一赋值都为列指定新的值。那个值统一地应用于您更新的每一行。在前面部分中的示例中，新的值为常量，但您可指定任意表达式，包括基于列值本身的表达式。假设制造商 HRO 已将所有价格提高百分之五，且您必须更新 stock 表来反映此提价。请使用下列语句：

```
UPDATE stock
        SET unit_price = unit_price * 1.05
        WHERE manu_code = 'HRO';
```

您还可使用子查询作为指定的值的一部分。当使用子查询作为表达式的元素时，它必须恰好返回一个值（一列和一行）。对于任何库存编号，或许您决定必须收取比那种产品的任何制造商都更高的价格。您需要更新所有未装运的订单的价格。下列示例中的 SELECT 语句指定该标准：

```
UPDATE items
        SET total_price = quantity *
        (SELECT MAX (unit_price) FROM stock
        WHERE stock.stock_num = items.stock_num)
        WHERE items.order_num IN
        (SELECT order_num FROM orders
        WHERE ship_date IS NULL);
```

第一个 SELECT 语句返回单个值：在 stock 表中，对于某个特定的产品的最高价格。第一个 SELECT 语句是一个相关联的子查询，因为当来自items 的值出现在第一个 SELECT 语句的 WHERE 子句中时，您必须为您更新的每一行都执行该查询。

第二个 SELECT 语句产生未装运的订单的订单编号的一个列表。它是一个执行一次的非相关的子查询。

7.4.3 对更新的限制

当您修改数据时，对子查询的使用存在限制。特别是，您不可查询正在修改的表。您可在表达式中引用列的当前值，如同在 unit_price 列增大百分之五的示例中那样。您还可引用在子查询中的 WHERE 子句中的列的值，如同在更新了 stock 表的示例中那样，其中，更新 items 表，且在连接表达式中使用 items.stock_num。

在设计良好的数据库中，不会经常发生同时更新与查询一个表的需要。然而，当您首次开发数据库时，尚未认真全面地考虑它的设计之前，您可能想要同时更新和查询。当无意中且错误地在表的应为唯一的列中包含了带有重复值的几行时，会发生一个典型的问题。您可能想要删除重复的行，或仅更新重复的行。不论是哪种方式，不可避免地需要对您想要修改的同一表上的子查询进行重复的行的测试。这在 UPDATE 语句或 DELETE 语句中是不允许的。通过 SQL 程序修改数据 讨论如何使用更新游标来执行此种修改。

7.4.4 用选择了的值更新

第二种形式的 UPDATE 语句以单个批量赋值替代赋值列表，其中设置列的列表与值的列表相同。当这些值是简单的常量时，这种形式与前面的示例的形式没什么不同，只是重新安排它的某些部分，如下例所示：

```
UPDATE customer
        SET (fname, lname) = ('Barnaby', 'Dorfler')
        WHERE customer_num = 103;
```

以此方式编写该语句，不存在任何优势。实际上，它更难于阅读，因为将哪些值指定给哪些列并不明显。

然而，当这些要指定的值来自单个 SELECT 语句时，这种形式就很合理。假设要将地址的更改应用于几个客户。不是每次更新 customer 表都报告更改，而是在名为 newaddr 的单个临时表中收集新的地址。现在，一次性应用所有新地址的时候到了。

```
UPDATE customer
        SET (address1, address2, city, state, zipcode) =
        ((SELECT address1, address2, city, state, zipcode
        FROM newaddr
        WHERE newaddr.customer_num=customer.customer_num))
        WHERE customer_num IN (SELECT customer_num FROM newaddr);
```

单个 SELECT 语句产生多个列的值。如果您以其他形式重新编写此示例，为每一更新了的列进行赋值，您必须编写五个 SELECT 语句，为每个要更新的列编写一个。这样的语句不仅更难编写，而且它会花费更长的执行时间。

提示：在 SQL API 程序中，您可使用记录或主变量来更新值。要获取更多信息，请参考 SQL 编程。

7.4.5 更新 row 类型

依赖于该列是命名了的 ROW 类型还是未命名的 ROW 类型，您用于更新 row 类型值的语法会不同。本部分描述那些差异，还描述如何为 ROW 类型的字段指定 NULL 值。

更新包含命名了的 row 类型的行

要更新在命名了的 ROW 类型上定义的列，您必须指定所有 ROW 类型的字段。例如，下列语句仅更新 employee 表中 address 列的 street 和 city字段，但每一 ROW 类型的字段必须包含一个值（允许 NULL 值）：

```
UPDATE employee
        SET address = ROW('103 California St',
        San Francisco', address.state, address.zip)::address_t
        WHERE name = 'zawinul, joe';
```

在此示例中，从该行中读取 state 和 zip 字段的值，然后立即重新插入到该行内。仅更新 address 列的 street 和 city 字段。

当您更新在命名了的 ROW 类型上定义的列的字段时，您必须使用 ROW 构造函数，并将该行值强制转型为适当的命名了的 ROW 类型。

更新包含未命名的 row 类型的行

要更新在未命名的 ROW 类型上定义的列，您必须指定该 ROW 类型的所有字段。例如，下列语句仅更新 student 表中 address 列的 street 和 city字段，但 ROW 类型的每一字段都必须包含一个值（允许 NULL 值）：

```
UPDATE student
        SET s_address = ROW('13 Sunset', 'Fresno',
        s_address.state, s_address.zip)
        WHERE s_name = 'henry, john';
```

要更新在未命名的 ROW 类型上定义的列的字段，请始终在插入该字段值之前，指定 ROW 构造函数。

为 row 类型的字段指定 Null 值

row 类型列的字段可包含 NULL 值。当您以 NULL 值插入到 row 类型字段内或更新 row 类型字段时，您必须将该值强制转型为那个字段的数据类型。

下列 UPDATE 语句展示您可以如何为命名了的 row 类型列的特定字段指定 NULL 值：

```
UPDATE employee
        SET address = ROW(NULL::VARCHAR(20), 'Davis', 'CA',
```

```
        ROW(NULL::CHAR(5), NULL::CHAR(4)))::address_t)
        WHERE name = 'henry, john';
```

下列 UPDATE 语句展示您如何为 student 表的 address 列的 street 和 zip 字段指定 NULL 值。

```
UPDATE student
        SET address = ROW(NULL::VARCHAR(20), address.city,
        address.state, NULL::VARCHAR(9))
        WHERE s_name = 'henry, john';
```

重要：您不可为 row 类型列指定 NULL 值。您仅可为 row 类型的个别的字段指定 NULL 值。

7.4.6 更新集合类型

当您使用 DB-Access 来 更新集合类型时，您必须更新整个集合。下列语句展示如何更新 projects 列。 要定位需要更新的行，请使用 IN 关键字在direct_reports 列上执行搜索。

```
UPDATE manager
        SET projects = "LIST
        {
        ROW('brazil_project', SET{'Pryor', 'Murphy', 'Kinsley',
        'Bryant'}),
        ROW ('cuba_project', SET{'Forester', 'Barth', 'Lewis',
        'Leonard'})
        }"
        WHERE 'Williams' IN direct_reports;
```

在前一语句中第一次出现的 SET 关键字是 UPDATE 语句语法的一部分。

重要：请不要将 UPDATE 语句的 SET 关键字与表明集合为 SET 数据类型的 SET 构造函数相混淆。

虽然您可使用 IN 关键字来定位简单集合的特定元素，但您不可从 DB-Access 更新集合列的个别元素。然而，您可创建 GBase 8s ESQL/C 程序和 SPL 例程来更新集合内的元素。要获取关于如何创建 GBase 8s ESQL/C 程序来更新集合的信息，请参阅《GBase 8s ESQL/C 程序员手册》。要获取关于如何创建 SPL 例程来更新集合的信息，请参阅 处理集合 部分。

7.4.7 更新超级表的行

当您更新超级表的行时，更新的作用域是超级表及其子表。

当您对超级表构造 UPDATE 语句时，您可更新该超级表中的所有列，以及从该超级表继承的子表的列。例如，下列语句更新来自 employee 和sales_rep 表的行，它们是超级表 person 的子表：

```
UPDATE person
        SET salary=65000
        WHERE address.state = 'CA';
```

然而，对超级表的更新不允许您更新不在该超级表内的子表的列。例如，在前面的更新语句中，您不可更新 sales_rep 表的 region_num 列，因为region_num 列未出现在 employee 表中。

当您对超级表执行更新时，请注意该更新的作用域。例如，对 person 表的 UPDATE 语句未包括 WHERE 子句来限定要更新的行，该语句修改person、employee 和 sales_rep 表的所有行。

要限定为仅对超级表的行更新，您必须在 UPDATE 语句中使用 ONLY 关键字。例如，下列语句仅更新 person 表的行：

```
UPDATE ONLY(person)
        SET address = ROW('14 Jackson St', 'Berkeley',
        address.state, address.zip)
        WHERE name = 'Sallie, A.';
```

重要： 当您更新超级表的行时，请小心使用，因为对超级表的更新的作用域包括该超级表及其所有子表。

7.4.8 更新列的 CASE 表达式

CASE 表达式允许语句返回几个可能的结果之一，这依赖于若干条件测试中哪个求值为 TRUE。

下列示例展示如何在 UPDATE 语句中使用 CASE 表达式来增加 stock 表中某些商品的单价：

```
UPDATE stock
        SET unit_price = CASE
        WHEN stock_num = 1
        AND manu_code = "HRO"
        THEN unit_price * 1.2
        WHEN stock_num = 1
        AND manu_code = "SMT"
        THEN unit_price * 1.1
        ELSE 0
```

```
        END
```

您必须在 CASE 表达式内包括至少一个 WHEN 子句；后续的 WHEN 子句和 ELSE 子句是可选的。如果无 WHEN 条件求值为真，则结果值为空。

7.4.9 更新智能大对象的 SQL 函数

您可使用可从 UPDATE 语句之内调用的 SQL 函数来导入和导出智能大对象。要了解这些函数的描述，请参阅 智能大对象函数 页。

下列 UPDATE 语句使用 LOCOPY() 函数来将 BLOB 数据从 fbi_list 表的 mugshot 列复制到 inmate 表的 picture 列内。（图 1 定义 inmate 和fbi_list 表。）

```
UPDATE inmate (picture)
        SET picture = (SELECT LOCOPY(mugshot, 'inmate', 'picture')
        FROM fbi_list WHERE fbi_list.id = 669)
        WHERE inmate.id_num = 437;
```

LOCOPY() 的第一个参数指定从其导出该对象的列（mugshot）。第二个和第三个参数指定该新创建的对象将使用其存储特性的表（inmate）和列（picture）的名称。该 UPDATE 语句执行之后，picture 列包含来自 mugshot 列的数据。

当您在该函数参数中指定文件名称的路径时，请应用下列规则：

- 如果源文件驻留在服务器计算机上，则您必须指定该文件的完全路径名称（而不是相对于当前工作目录的路径名称）。
- 如果源文件驻留在客户机计算机上，则您可指定该文件的完全路径名称或相对路径名称。

7.4.10 更新表的 MERGE 语句

MERGE 语句允许您对源表与目标表的一个外部连接的结果应用布尔条件。如果 MERGE 语句包括 Update 子句，则对目标在 UPDATE 操作中使用那些满足您在 ON 关键字之后指定的连接条件的行。MERGE 语句的 SET 子句支持与 UPDATE 语句的 SET 子句相同的语法，并指定要更新的目标表的哪些列。

下例示例展示您可如何使用 MERGE 语句的 Update 子句来更新目标表：

```
MERGE INTO t_target AS t USING t_source AS s ON t.col_a = s.col_a
        WHEN MATCHED THEN UPDATE
        SET t.col_b = t.col_b + s.col_b ;
```

在前一示例中，目标表的名称为 t_target，源表的名称为 t_source。对于在源表与目标表中其 col_a 都有相同的值的连接结果的行，MERGE 语句通过将源表中 col_b 列的值添加到 t_target 表中 col_b 列的当前值来更新 t_target 表。

MERGE 语句的 UPDATE 操作不修改源表，且不可更新目标表中的任何行超过一次。

单个 MERGE 语句可同时组合 UPDATE 与 INSERT 操作，或可同时组合 DELETE 与 INSERT 操作而不需要删除子句。要了解不包括 Update 子句的 MERGE 的不同的示例，请参阅主题 MERGE 的 Delete 子句

7.5 对数据库级对其对象的权限

您可使用下列数据库权限来控制谁可访问数据库：

- 数据库级别权限
- 表级别权限
- 例程级别权限
- 语言级别权限
- 类型级别权限
- 序列级别权限
- 分片级别权限

本部分简要地描述数据库级别和表级别权限。要了解权限的列表以及 GRANT 和 REVOKE 语句的描述，请参阅《GBase 8s SQL 指南：语法》。

7.5.1 数据库级别权限

当您创建数据库时，您是唯一可访问它的人，直到您作为该数据库的所有者或数据库管理员（DBA），将数据库级别权限授予其他人。下表展示数据库级别权限。

权限	影响
Connect	允许您打开数据库、发出查询以及在临时表上创建和放置索引。
Resource	允许您创建永久表。
DBA	允许您作为 DBA 执行若干附加的函数。

7.5.2 表级别权限

当您在不符合 ANSI 的数据库中创建表时，所有用户都有访问该表的权限，直到您作为该表的所有者取消特定用户的表级别权限为止。下表介绍控制用户可如何访问表的四种权限。

权限	用途

权限	用途
Select	逐表授予权限，并允许您从表选择行。（此权限可限定于表中的特定列。）
Delete	允许您删除行。
Insert	允许您插入行。
Update	允许您更新现有的行（即，更改其内容）。

创建数据库和表的人们经常将 Connect 和 Select 权限授予 public，以便所有用户都拥有它们。如果您可查询表，则您至少具有对那个数据库和表的 Connect 和 Select 权限。

您需要其他的表级别权限来修改数据。表的所有者经常保留这些权限，或仅将它们授予特定的用户。因此，您可能无法修改您可自由地查询的一些表。

例如，由于这些权限都是逐表授予的，因此您仅可拥有对一个表的 Insert 权限，以及仅拥有对另一表的 Update 权限。甚至可进一步将 Update 权限限定于表中的特定列。

7.5.3 显示表权限

如果您是表的所有者（即，如果您创建了它），则您拥有对那个表的所有权限。否则，您可通过查询系统目录来确定您对于某个表拥有的权限。系统目录由描述数据库结构的系统表构成。对每一表所授予的权限都记录在 systabauth 系统表中。要显示这些权限，您还必须知道该表的唯一标识符编号。在 systables 系统表中指定此编号。要显示对 orders 表授予的权限，您可输入下列 SELECT 语句：

```
SELECT * FROM systabauth
            WHERE tabid = (SELECT tabid FROM systables
            WHERE tabname = 'orders');
```

该查询的输出类似于下列示例：

```
grantorgrantee tabid                    tabauth

            tfecitmutator   101         su-i-x--
            tfecitprocrustes101         s--idx--
            tfecitpublic    101         s--i-x--
```

授权者是授予权限的用户。授权者通常是表的所有者，但所有者可为授权者授权了的另一用户。被授权者是将权限授予其的用户，被授权者 public意味着有 Connect 权限的任何用户。如果您的用户名未出现，则您仅拥有授予给了 public 的那些权限。

tabauth 列指定授予的权限。此列的每一行中的字母是权限名称的首字母，除了 i 表示 Insert 以及 x 表示 Index 之外。在此示例中，public 具有 Select、Insert 和 Index 权限。仅用户 mutator 具有 Update 权限，仅用户 procrustes 具有 Delete 权限。

在数据库服务器为您执行任何操作（例如，执行 DELETE 语句）之前，它都执行类似于前一查询的查询。如果您不是该表的所有者，且如果数据库服务器找不到您的用户名或 public 对该表的必要权限，则它拒绝执行该操作。

7.5.4 将权限授予角色

作为 DBA，您可创建角色来使得给予一类用户的权限标准化。当您将权限指定给那个角色时，那个角色的每个用户都拥有那些权限。用于定义和操纵角色的 SQL 语句包括：CREATE ROLE、DROP ROLE、GRANT、REVOKE 和 SET ROLE。要获取关于定义和操纵角色的 SQL 语句的语法的更多信息，请参阅《GBase 8s SQL 指南：语法》。

在连接到数据库时，缺省的角色自动地应用于特定的用户和组，而不要求该用户发出 SET ROLE 语句。例如：

```
GRANT DEFAULT ROLE manager TO larry;
```

要获取关于角色与缺省角色的更多信息，请参阅 控制数据库使用 或参阅《GBase 8s 管理员指南》。

要获取关于授予和撤销权限的更多信息，请参阅 授予和撤销应用程序中的权限。

7.6 数据完整性

INSERT、UPDATE 和 DELETE 语句修改现有的数据库中的数据。每当您修改现有的数据时，就可影响数据的完整性。例如，可能会将不存在的产品的订单输入到 orders 表内，可能从 customer 表中删除一个有未完成订单的客户，或者可能在 orders 表中更新订单编号，但未在 items 表中更新。在每一这些情况下，都会失去存储的数据的完整性。

数据完整性实际由下列部分组成：

实体完整性

表的每一行都有唯一的标识符。

语义完整性

列中的数据正确地反映设计了该列来保存的信息的类型。

引用完整性

强制执行表之间的关系。

设计良好的数据库体现了这些原则，因此当您修改数据时，数据库本身防止您执行可能损坏数据完整性的任何操作。

7.6.1 实体完整性

实体是要记录在数据库中的任何人、位置或事物。每一表都表示一个实体，且表的每一行都表示那个实体的一个实例。例如，如果 order 是一个实体，则 orders 表表示订单的概念，表中的每一行表示一特定的订单。

要标识表中的每一行，该表必须有一主键。主键是标识每一行的一个唯一值。此要求称为实体完整性约束。

例如，orders 表的主键是 order_num。order_num 列为表中的每一行保存一个唯一的系统生成的订单编号。要访问 orders 表中的一行数据，请使用下列 SELECT 语句：

```
SELECT * FROM orders WHERE order_num = 1001;
```

在此语句的 WHERE 子句中使用该订单编号使得您能够容易地访问行，因为该订单编号唯一地标识那一行。如果该表允许重复的订单编号，则它几乎不可能访问单个一行，因为此表的所有其他列都允许重复的值。

7.6.2 语义完整性

语义完整性确保输入到行内的数据反映那一行的允许的值。该值必须在那一行的域或允许的值集之内。例如，items 表的 quantity 列仅允许数值。如果可将该域之外的值输入到列内，则违反该数据的语义完整性。

下列约束强制语义完整性：

数据类型

数据类型定义您可存储在列中的值的类型。例如，数据类型 SMALLINT 允许您将从 -32,767 至 32,767 的值输入到列内。

缺省值

缺省值是当未指定显式的值时插入到该列内的值。例如，如果未输入名称，则 cust_calls 表的 user_id 列的缺省值为该用户的登录名称。

检查约束

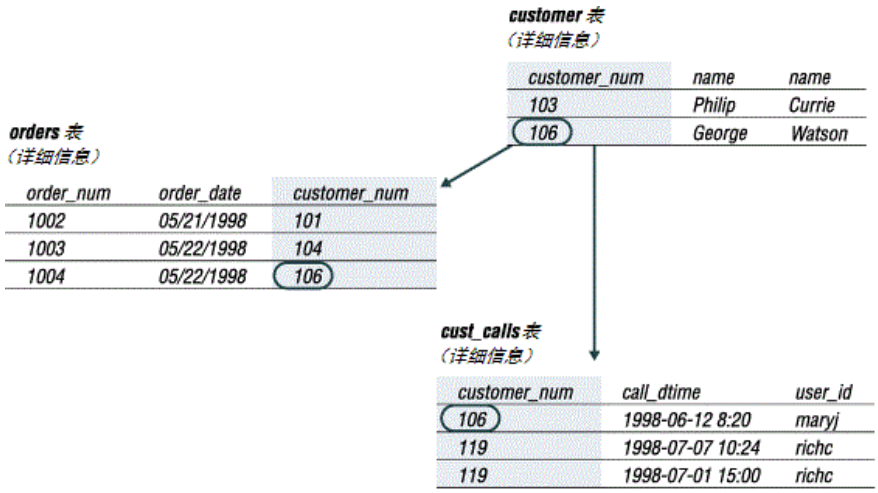
检查约束指定对插入到列内的数据的条件。输入到表内的每一行都必须满足这些条件。例如，items 表的 quantity 列可能检查大于或等于 1 的数量。

7.6.3 引用完整性

引用完整性指的是表之间的关系。由于数据库中每一表都必须具有主键，因此此主键可出现在其他的表中，这是因为它与那些表内数据的关系。当来自一个表的主键出现在另一表中时，将它称之为外键。

外键连接表并在表之间创建依赖。若干表可形成依赖的层级结构，这样，如果您更改或删除一个表中的行，则您破坏在其他表中行的含义。例如，下图展示 customer 表的 customer_num 列是那个表的主键，以及 orders 和 cust_call 表中的外键。
在 orders 和 cust_calls 表中都引用客户编号 106，George Watson™。如果从 customer 表删除客户 106，则破坏三个表以及此特定的客户之间的链接。

图: 演示数据库中的引用完整性



当您删除包含主键的行，或以不同的主键更新它时，您破坏了包含那个值作为外键的任何行的含义。引用完整性是外键对主键的逻辑依赖。包含外键的行的完整性依赖于它引用的那行的完整性—包含相匹配的主键的行。

在缺省情况下，数据库服务器不允许您违反引用完整性，且如果在您从子表删除行之前，您尝试从父表删除行，则向您提示错误消息。然而，您可使用 ON DELETE CASCADE 选项来在从父表删除的同时对相应的子表进行删除。请参阅 ON DELETE CASCADE 选项。

要定义主键和外键以及它们之间的关系，请使用 CREATE TABLE 和 ALTER TABLE 语句。要获取关于这些语句的更多信息，请参阅《GBase 8s SQL 指南：语法》。

ON DELETE CASCADE 选项

当您从主键为表删除行时，要保持引用完整性，请使用 CREATE TABLE 和 ALTER TABLE 语句的 REFERENCES 子句中的 ON DELETE CASCADE 选项。此选项允许您使用单个删除命令从父表删除一行以及在相匹配的子表中它的对应行。

在级联删除期间锁定

在删除期间，保持父表和子表上的所有符合条件的行上的锁定。当您指定删除时，在执行任何引用操作之前，执行从父表请求的删除。

多个子表的情况

如果您具有带有两个子约束的父表，一个子表指定了级联删除，另一个子表没有级联删除，且您尝试从同时应用于两个子表的父表删除一行，则 DELETE 语句失败，且从父表和子表都不删除行。

必须打开日志记录

为了使级联删除起作用，您必须在您的当前数据库中打开日志记录。在 事务日志记录 中讨论日志记录和级联删除。

级联删除的示例

假设您有应用了引用完整性规则的两个表，父表 accounts，以及子表 sub_accounts。下列 CREATE TABLE 语句定义引用约束：

```
CREATE TABLE accounts (
        acc_num SERIAL primary key,
        acc_type INT,
        acc_descr CHAR(20));

        CREATE TABLE sub_accounts (
        sub_acc INTEGER primary key,
        ref_num INTEGER REFERENCES accounts (acc_num)
        ON DELETE CASCADE,
        sub_descr CHAR(20));
```

accounts 表的主键，acc_num 列，使用 SERIAL 数据类型，sub_accounts 表的外键，ref_num 列，使用 INTEGER 数据类型。允许组合主键上的 SERIAL 与外键上的 INTEGER 数据类型。仅在此条件下，您可混合并匹配数据类型。SERIAL 数据类型是 INTEGER，且数据库自动地为该列生成值。所有其他主键与外键组合都必须显式地相匹配。例如，定义为 CHAR 的主键必须与定义为 CHAR 的外键相匹配。

sub_accounts 表的外键的定义，ref_num 列，包括 ON DELETE CASCADE 选项。此选项指定在父表 accounts 中任何行的删除都将自动地导致删除子表 sub_accounts 的对应行。

要从将级联删除 sub_accounts 表的 accounts 表删除一行，您必须打开日志记录。打开日志记录之后，您可从两个表都删除账户编号 2，如下例所示：

```
DELETE FROM accounts WHERE acc_num = 2;
```

对级联删除的限制

对于大多数删除，包括自引用的删除和循环查询的删除，您可使用级联删除。唯一的例外是相关的子查询，相关的子查询是嵌套的 SELECT 语句，子查询（或内部 SELECT）在其中产生的值依赖于包含它的外部 SELECT 语句所产生的值。如果您已实施了级联删除，则您不可在相关的子查询中编写使用子表的删除。当您尝试从相关的子查询删除时，您会收到错误。

限制： 如果表使用 ON DELETE CASCADE 定义引用约束，则您不可在该表上定义 DELETE 触发器事件。

7.6.4 对象模式和违反检测

数据库的对象模式和违反检测可帮助您监视数据完整性。在模式更改期间，或当对于短期内大批量数据执行插入、删除和更新操作时组合这些特性，这些特性特别有效。

在对象模式特性的讨论的上下文之内，数据库对象是约束、索引和触发器，且它们中的每一个都有不同的模式。请不要将与对象模式特性相关的数据库对象与一般的数据库对象相混淆。一般的数据库对象是诸如表和同义词之类的对象。

对象模式的定义

您可为约束或唯一索引设置禁用、启用或过滤模式。您可为触发器或重复索引设置启用或禁用模式。您可使用数据库对象模式来控制 INSERT、DELETE 和 UPDATE 语句的效果。

启用模式

在缺省情况下，约束、索引和触发器是启用的。

当数据库对象是启用的时，数据库服务器识别该数据库对象的存在，并在它执行 INSERT、DELETE 或 UPDATE 语句时考虑该数据库对象。因此，当触发器事件发生时，强制执行启用的约束，更新启用的索引，并执行启用的触发器。

当您启用约束和唯一索引时，如果存在违反的行，则该数据处理语句失败（即，不更改行）且数据库服务器返回错误消息。

当您分析违反表和诊断表中的信息时，您可标识该失败的原因。然后，您可采取更正活动或回滚该操作。

禁用模式

当数据库对象是禁用的时，在执行 INSERT、DELETE 或 UPDATE 语句时，数据库服务器不考虑它。当触发器事件发生时，不强制执行禁用的约束，不更新禁用的索引，也不执行禁用的触发器。当您禁用约束和唯一索引时，违反该约束或唯一索引的限制的任何数据操纵语句都成功，（即，更改目标行），且数据库服务器不返回错误消息。

过滤模式

当约束或唯一索引处于过滤模式时，该语句成功，且在 INSERT、DELETE 或 UPDATE 语句期间，通过将失败了的行写到与该目标表相关联的违反表，数据库服务器强制满足约束或唯一索引需求。将关于该约束违反的诊断信息写到与目标表相关联的诊断表。

使用数据操纵语句的模式的示例

一个使用 INSERT 语句的示例可说明启用模式、禁用模式与过滤模式之间的差异。请考虑这样一条 INSERT 语句，其中一个用户试图在表上添加不满足完整性约束的一行。例如，假设用户 joe 创建了名为 cust_subset 的表，且此表由下列列构成：ssn（客户的社会保险编号）、fname（客户的名）、lname（客户的姓）以及 city（客户生活的城市）。ssn 列具有 INT 数据类型。其他三列有 CHAR 数据类型。

假设用户 joe 定义了 lname 列为非空，但尚未将名称指定给非空约束，于是，数据库服务器已隐式地将名称 n104_7 指定给此约束。最后，假设用户 joe 在 ssn 列上创建了名为 unq_ssn 的唯一索引。

现在，对 cust_subset 有 Insert 权限的用户 linda 在此表上输入下列 INSERT 语句：

```
INSERT INTO cust_subset (ssn, fname, city)
        VALUES (973824499, "jane", "los altos");
```

要更好地理解启用模式、禁用模式与过滤模式之间的区别，您可在下面三个部分中查看前面的 INSERT 语句的结果。

当约束为启用的时，插入操作的结果

如果在 cust_subset 表上的 NOT NULL 约束是启用的，则 INSERT 语句不能在此表中插入新行。而当用户 linda 输入该 INSERT 语句时，她收到下列错误消息:

```
-292 An implied insert column lname does not accept NULLs.
```

当约束为禁用时，插入操作的结果

如果在 cust_subset 表上 NOT NULL 约束是禁用的，则用户 linda 发出的 INSERT 操作成功地在此表中插入新行。cust_subset 表的新行有下列列值。

ssn	fname	lname	city
973824499	jane	NULL	los altos

当约束处于过滤模式时，插入的结果

如果将 cust_subset 表上的 NOT NULL 约束设置为过滤模式，则用户 linda 发出的 INSERT 命令不能在此表中插入新行。而将该新行插入到违反表内，并将描述完整性违反的诊断行添加到诊断表。

假设用户 joe 已为 cust_subset 表启动了违反表和诊断表。违反表名为 cust_subset_vio，诊断表名为 cust_subset_dia。当用户 linda 在cust_subset 目标表上发出 INSERT 语句时，添加到 cust_subset_vio 违反表的新行有下列列值。

ssn	fname	lname	city	gbasedbt_tupleid	gbasedbt_optype	gbasedbt_recowner
973824499	jane	NULL	los altos	1	I	linda

cust_subset_vio 违反表中的此新行有下列特征：

违反表的前四列恰好与目标表的列相匹配。这四列与目标表的对应列有相同的名称和相同的数据类型，它们具有用户 linda 输入的 INSERT 语句提供了的列值。

gbasedbt_tupleid 列中的值 1 是分配给不符合的行的唯一序列标识符。

gbasedbt_optype 列中的值 I 表示操作类型的代码，该操作已导致了创建此不符合的行。特别地，I 代表 INSERT 操作。

gbasedbt_recowner 列中的值 linda 标识发出了导致创建此不符合行的用户。

用户 linda 在 cust_subset 目标表上发出的 INSERT 语句还导致将诊断行添加到 cust_subset_dia 诊断表。添加到诊断表的诊断行有下列列值。

gbasedbt_tupleid	objtype	objowner	objname
1	C	joe	n104_7

cust_subset_dia 诊断表中的此新诊断行有下列特征：

通过同时出现在两表中的 gbasedbt_tupleid 列，将此诊断表的行连接到违反表的对应行。值 1 同时出现在两表中的此列。

objtype 列中的值 C 标识违反表中对应行导致的完整性违反的类型。特别地，值 C 代表约束违反。

objowner 列中的值 joe 表示检测到违反完整性的约束的所有者。

objname 列中的值 n104_7 给出对其检测到了完整性违反的约束的名称。

通过连接违反表与诊断表，用户 joe（其拥有 cust_subset 目标表及其相关联的特殊表）或 DBA 可在违反表中发现在 INSERT 语句之后创建了的其 gbasedbt_tupleid 值为 1 的行，且此行违反约束。表的所有者或 DBA 可查询 sysconstraints 系统目录表来确定此约束为 NOT NULL 约束。既然知道该 INSERT 语句失败的原因，用户 joe 或 DBA 便可采取更正行动。

对于一个违反行的多个诊断行

在前面的示例中，仅诊断表中的一行对应于违反表中的新行。然而，当将单个新行添加到违反表时，可将多个诊断行添加到诊断表。例如，如果用户linda 在 INSERT 语句中输入了的 ssn 值（973824499）与 cust_subset 目标表的 ssn 列中现有的值相同，则在违反表中仅会出现一个新行，但会在cust_subset_dia 诊断表中出现下列两个诊断行。

gbasedbt_tupleid	objtype	objowner	objname
1	C	joe	n104_7
1	I	joe	unq_ssn

诊断表中的两行同时对应于违反表的同一行，因为这两行在 gbasedbt_tupleid 列中都有值 1。然而，第一个诊断行标识用户 linda 发出了的 INSERT 语句导致约束违反，而第二个诊断行标识同一 INSERT 语句导致唯一索引违反。在此第二个诊断行中，objtype 中的值 I 代表唯一索引违反，且objname 列中的值 unq_ssn 给出检测出了完整性违反的索引的名称。

要获取关于如何设置数据库对象模式的更多信息，请参阅《GBase 8s SQL 指南：语法》中的 SET Database Object Mode 语句。

违反表和诊断表

当您为目标表启动违反表时，在对目标表的 INSERT、UPDATE 和 DELETE 操作期间，违反约束和唯一索引的任何行都不会导致整个操作失败，但会被过滤到违反表。诊断表包含关于由违反表中每一行导致的完整性违反的信息。通过检查这些表，您可标识失败的原因，并通过修正违反或回滚操作来采取更正行动。

在您为目标表创建违反表之后，您不可改变基础表或违反表的列或分片。在您已启动了违反表之后，如果您改变目标表上的约束，则将不符合的行过滤到违反表。

要获取关于如何启动和停止违反表的信息，请参阅《GBase 8s SQL 指南：语法》中的 START VIOLATIONS TABLE 和 STOP VIOLATIONS TABLE 语句。

违反表与数据库对象模式的关系

如果您将定义在表上的约束和唯一索引设置为过滤模式，但您未为此目标表创建违反表和诊断表，则在插入、更新或删除操作期间，违反约束或唯一索引要求的任何行都不过滤到违反表。相反，您会收到错误消息，指示您必须为目标表启动违反表。

类似地，如果您将禁用的约束或禁用的唯一索引设置为启用模式或过滤模式，且您想要能够标识不满足约束或唯一索引要求的现有的行，则在您发出 SET Database Object Mode 语句之前必须创建违反表。

START VIOLATIONS TABLE 语句的示例

下列示例展示执行 START VIOLATIONS TABLE 语句的不同方式。

启动违反表和诊断表而不指定它们的名称

要为演示数据库中名为 customer 的目标表启动违反表和诊断表，请输入下列语句：

```
START VIOLATIONS TABLE FOR customer;
```

由于您的 START VIOLATIONS TABLE 语句不包括 USING 子句，因此违反表的缺省名称为 customer_vio，诊断表的缺省名称为customer_dia。customer_vio 表包括下列列：

```
customer_num
        fname
        lname
        company
        address1
        address2
        city
        state
        zipcode
        phone
        gbasedbt_tupleid
        gbasedbt_optype
        gbasedbt_recowner
```

customer_vio 表与 customer 表有相同的表定义，除了 customer_vio 表有包含关于导致了坏行的操作的信息的三个附加列之外。

customer_dia 表包括下列列：

```
gbasedbt_tupleid
        objtype
        objowner
        objname
```

对于目标表，此列的列表展示诊断表与违反表之间的重要差异。尽管对于目标表中的每列，违约表都有相匹配的列，但诊断表的列不依赖于目标表的模式。由任何 START

VIOLATIONS TABLE 语句创建的诊断表始终有以上列表中的四列，带有相同的列名称和数据类型。

启动违反表和诊断表并指定它们的名称

下列语句为名为 items 的目标表启动违反表和诊断表。USING 子句为违反表和诊断表声明显式的名称。违反表将命名为 exceptions，诊断表将命名为 reasons。

```
START VIOLATIONS TABLE FOR items
        USING exceptions, reasons;
```

指定诊断表中的最大行数

下列语句为名为 orders 的目标表启动违反表和诊断表。当在目标表上执行诸如 INSERT、MERGE 或 SET Database Object Mode 之类的单个语句时，MAX ROWS 子句指定可插入到 orders_dia 诊断表内的最大行数。

```
START VIOLATIONS TABLE FOR orders MAX ROWS 50000;
```

如果您未为 START VIOLATIONS TABLE 语句中的 MAX ROWS 指定值，则对诊断表中的行数没有缺省的限制，除了可用的磁盘空间之外。

MAX ROWS 子句仅对于表函数在其中作为诊断表的操作限定行的数目。

对违反表的权限的示例

下列示例说明如何从对目标表的权限的当前集派生对违反表的权限的初始集。

例如，假设我们创建了名为 cust_subset 的表，且此表由下列列组成：ssn（客户的社会保险号码）、fname（客户的名）、lname（客户的姓）以及 city（客户生活的城市）。

在 cust_subset 表上存在下列权限集：

- 用户 alvin 是该表的所有者。
- 用户 barbara 具有对该表的 Insert 和 Index 权限。她还具有对 ssn 和 lname 列的 Select 权限。
- 用户 carrie 具有对 city 列的 Update 权限。她还具有对 ssn 列的 Select 权限。
- 用户 danny 具有对该表的 Alter 权限。

现在，用户 alvin 为 cust_subset 表启动名为 cust_subset_viols 的违反表和名为 cust_subset_diags 的诊断表，如下：

```
START VIOLATIONS TABLE FOR cust_subset
    USING cust_subset_viols, cust_subset_diags;
```

数据库服务器授予对 cust_subset_viols 违反表的下列初始权限集：

- 用户 alvin 是该违反表的所有者，因此他具有对该表的所有表级别权限。

- 用户 barbara 具有对违反表的 Insert、Delete 和 Index 权限。她还具有对违反表的下列列的 Select 权限：ssn 列、lname列、gbasedbt_tupleid 列、gbasedbt_optype 列和 gbasedbt_recowner 列。
- 用户 carrie 具有对违反表的 Insert 和 Delete 权限。她还具有对违反表的下列列的 Update 权限：city 列、gbasedbt_tupleid列、gbasedbt_optype 列和 gbasedbt_recowner 列。她具有对违反表的下列列的 Select 权限：ssn 列、gbasedbt_tupleid列、gbasedbt_optype 列和 gbasedbt_recowner 列。
- 用户 danny 不具有对违反表的任何权限。

对诊断表的权限的示例

下列示例说明如何从对目标表的当前权限集派生对诊断表的初始权限集。

例如，假设名为 cust_subset 的表由下列列组成：ssn（客户的社会保险编号）、fname（客户的名）、lname（客户的姓）以及 city（客户生活的城市）。

对 cust_subset 表存在下列权限集：

- 用户 alvin 为该表的所有者。
- 用户 barbara 具有对该表的 Insert 和 Index 权限。她还具有对 ssn 和 lname 列的 Select 权限。
- 用户 carrie 具有对 city 列的 Update 权限。她还具有对 ssn 列的 Select 权限。
- 用户 danny 具有对该表的 Alter 权限。

现在，用户 alvin 为 cust_subset 表启动名为 cust_subset_viols 的违反表和名为 cust_subset_diags 的诊断表，如下：

```
START VIOLATIONS TABLE FOR cust_subset
        USING cust_subset_viols, cust_subset_diags;
```

数据库服务器对于 cust_subset_diags 诊断表授予下列初始的权限集：

- 用户 alvin 为诊断表的所有者，因此他具有对该表的所有表级别权限。
- 用户 barbara 具有对诊断表的 Insert、Delete、Select 和 Index 权限。
- 用户 carrie 具有对诊断表的 Insert、Delete、Select 和 Update 权限。
- 用户 danny 对诊断表没有权限。

7.7 中断了的修改

即使所有软件都没有错误且所有硬件都完全可靠，计算机外部的世界也可干扰它。闪电可能击中建筑物，中断供电并在您的 UPDATE 语句运行期间停止计算机。当磁盘已满或用

户提供不正确的数据时，更可能发生的情景是，导致您的多行插入过早停止并产生错误。在任何情况下，每当您修改数据，您必须假设某种不可预测的事件可中断该修改。

当外部原因导致修改中断时，您不可确定该操作完成了多少。即使在单行操作中，您也不可知道是否正确地更新了到达了磁盘的数据或索引。

如果多行修改是一个问题，则多语句修改就更糟。通常在程序中嵌入它们，因此您看不到正在执行的个别 SQL 语句。例如，要在演示数据库中输入新的订单，请执行下列步骤：

1. 在 orders 表中插入一行。（此插入生成一个订单编号。）
2. 对于订购的每一商品，在 items 表中插入一行。

存在两种编制订单输入应用程序的方法。一种方法是使它完全是交互的，以便程序立即插入第一行，然后在用户输入时插入每一商品。但这种方法使得操作可能遭遇许多更不可预测的事件：客户的电话电线，用户按错键，用户的终端或计算机断电，等等。

下列列表描述构建订单输入应用程序的正确方法：

- 以交互方式接受所有数据。
- 验证数据并展开它（例如，在 stock 和 manufact 中查找代码）。
- 在屏幕上显示信息以进行检查。
- 等待操作人员进行最终的提交。
- 快速地执行插入。

即使使用这些步骤，不可预测的情况还可在它插入该订单之后，但在它完成插入商品之前停止该程序。如果发生那种情况，则数据库处于不可预测的状态：它的数据完整性受到损害。

7.7.1 事务

对所有这些潜在问题的解决方案称为事务。事务是必须或者全部完成，或者根本就不执行的修改的序列。数据库服务器保证在事务的范围内执行的操作或者完整并正确地提交到磁盘，或者将数据库恢复到事务开始之前的同一状态。

事务不仅是对不可预测的故障的保护；当程序检测到逻辑错误时，它还为程序提供一种规避的方法。

7.7.2 事务日志记录

数据库服务器可保持对在事务期间数据库服务器对数据库进行的每一更改的记录。如果发生了取消该事务的情况，则数据库服务器自动地使用这些记录来撤销更改。许多原因可导致事务失败。例如，发出 SQL 语句的程序可失败或被终止。数据库服务器一发现事务失

败，失败可能就在重新启动计算机和数据库服务器之后发生，它就使用来自该事务的记录来将数据库返回到之前的同一状态。

保存事务的记录的过程称为事务日志记录，或简称为日志记录。事务的记录，称为日志记录，保存在与数据库分开的磁盘空间部分中。此空间称为逻辑日志，因为该日志记录表示事务的逻辑单元。

GBase 8s 提供下列支持：

- 在日志记录数据库中创建无日志记录（raw）或日志记录（standard）的表。
- 使用 ALTER TABLE 语句将表从无日志记录改变为日志记录，或相反。

为了快速加载非常大的表，GBase 8s 支持无日志记录的表。建议您在事务内不使用无日志记录的表。要避免并发问题，在您在事务中使用表之前，请使用 ALTER TABLE 语句来使该表成为 standard（即，日志记录）。

要获取关于 GBase 8s 的无日志记录的表的更多信息，请参阅《GBase 8s 管理员指南》。要了解无日志记录的表的性能优势，请参阅《GBase 8s 性能指南》。要获取关于 ALTER TABLE 语句的信息，请参阅 GBase 8s SQL 指南：语法。

大多数 GBase 8s 数据库不会自动地生成事务记录。DBA 决定数据库是否使用事务日志记录。没有事务日志记录，您就不可回滚事务。

日志记录和级联删除

为了使级联删除起作用，必须在您的数据库中打开日志记录，因为当您指定级联删除时，首先在父表的主键上执行删除。如果在执行父表的主键的行删除之后，但在删除子表的外键的行之前，系统出现故障，则违反引用完整性。如果关闭日志记录，即使是临时地关闭，也不会级联删除。然而，在重新打开日志记录之后，又可级联删除。

GBase 8s 允许您使用 CREATE DATABASE 语句中的 WITH LOG 子句来打开日志记录。

7.7.3 指定事务

您可使用两种方法来用 SQL 语句指定事务的边界。在最常用的方法中，通过执行 BEGIN WORK 语句指定多语句事务的开始。在以 MODE ANSI 选项创建的数据库中，不存在标记事务的开始的需要。总会有一个起作用；您只要指明每一事务的结束。

在两种方法中，要指定成功的事务的结束，请执行 COMMIT WORK 语句。此语句告诉数据库服务器您达到了必须一起成功完成的一系列语句的结束。数据库服务器执行任何必要的操作来确保正确地完成了所有修改并提交到了磁盘。

程序还可通过执行 ROLLBACK WORK 语句来有意地取消事务。此语句请求数据库服务器取消当前事务并撤销任何更改。

当订单输入应用程序创建新订单时，它可以下列方式使用事务：

- 交互地接受所有数据。
- 验证并展开它。
- 等候操作人员进行最终的提交
- 执行 BEGIN WORK
- 在 orders 和 items 表中插入行，检查数据库服务器返回的错误代码
- 如果未发生错误，则执行 COMMIT WORK；否则，执行 ROLLBACK WORK

如果任何外部故障阻止事务的完成，在当系统重启时，部分事务回滚。在所有情况下，该数据库处于不可预测的状态。要么完全地输入新订单，要么根本未输入它。

7.8 使用 GBase 8s 数据库服务器来备份和记录日志

通过使用事务，您可确保数据库始终处于一致的状态，且将您的修改正确地记录在磁盘上。但磁盘本身不十分安全。对于机械故障以及洪水、火灾和地震，它都是脆弱的。唯一的保护措施是保存数据的多个副本。这些冗余的副本称为备份副本。

事务日志（也称为逻辑日志）补充数据库的备份副本。它的内容是自从上一次备份数据库以来发生了的所有修改的历史。如果您曾经需要从备份副本恢复数据库，则您可使用事务日志来将数据库向前滚到它最近的状态。

数据库服务器包含完善的特性来支持备份和日志记录。您的数据库服务器归档和备份指南描述这些特性。

数据库服务器对性能和可靠性有严格的要求（例如，它支持在正在使用数据库时制作备份副本。）

数据库服务器管理它自己的磁盘空间，这些空间专用于日志记录。

数据库服务器使用有限的日志文件集来并发地执行所有数据库的日志记录。在事务为活动的时，可将日志文件复制到另一介质（备份）。

数据库服务器从不需要考虑这些设置，因为 DBA 总是从中央位置管理它们。

GBase 8s 支持 onload 和 onunload 实用程序。使用 onunload 实用程序来制作单个数据库或表的个人备份副本。此程序将表或数据库复制到磁带。它的输出由磁盘页的二进制映像组成，如同在数据库服务器中存储了它们。因此，可快速地制作副本，且相应的 onload 程序可快速地恢复该文件。然而，对于任何其他程序，该数据格式没有意义。要获取关于如何使用 onload 和 onunload 实用程序的信息，请参阅《GBase 8s 迁移指南》。

如果您的 DBA 使用 ON-Bar 来创建备份并备份逻辑日志，则您还可能能够使用 ON-Bar 来创建您自己的备份副本。要获取更多信息，请参阅您的《GBase 8s 备份与恢复指南》。

7.9 并发和锁定

如果在单个用户工作站中包含您的数据库，而没有网络将它连接到其他计算机，则并发并不重要。在所有其他情况下，您必须允许这样的可能性：当您的程序正在修改数据时，另一程序也正在读取或修改同一数据。并发涉及在同一时间对同一数据的两个或多个独立使用。

在多用户数据库系统中，高级别的并发对良好的性能至关重要。然而，除非在数据的使用上存在控制，否则并发可导致各种负面的影响。程序可能读取过时的数据；即使表面上似乎成功地输入了修改，但修改可能丢失。

要防止此类错误，数据库服务器强加一个锁定系统。锁定是程序可在一块数据上放置的声明或保留。只要数据被锁定，数据库服务器就保证其他程序不可修改它。当另一程序请求该数据时，数据库服务器或者让该程序等待，或者向它返回错误。

要控制锁定对您的数据访问的影响，请使用 SQL 语句的组合：SET LOCK MODE 与或者 SET ISOLATION 或者 SET TRANSACTION。在从程序内阅读关于游标的使用的讨论之后，您可了解这些语句的详细信息。在 SQL 编程 和 通过 SQL 程序修改数据 中讨论游标。要获取关于锁定和并发的更多信息，请参阅 对多用户环境编程。

7.10 GBase 8s 数据复制

从更广义上讲，术语数据复制意味着在多个不同的站点，数据库对象多次出现。例如，有一种复制数据的方式是将数据库复制到不同的计算机上的数据库服务器，这样，报告可针对该数据运行，而不干扰正在使用原始数据库的客户机应用程序。

下列列表描述数据复制的优势：

- 与未复制的远程数据相对，在本地访问复制了的数据的客户机的性能提高，因为它们无需使用网络服务。
- 使用复制了的数据，提高所有站点的客户机的可用性，因为如果本地的复制了的数据不可用时，尽管是远程地，该数据的一个副本仍可用。

获得这些优势不是没有代价的。与非复制的数据相比，对于复制了的数据，数据复制显然需要更多的存储，且更新复制了的数据可比更新单个对象要花费更多处理时间。

通过显式地指定应发现和更新数据的位置，可在客户机应用程序的逻辑中实际地实现数据复制。然而，归档数据复制的这种方式成本高、容易出错且难以维护。相反，数据复制的概念常常伴随着复制透明。复制透明是在数据库服务器内自动地处理定位和维护数据副本的详细信息的内建功能。

在数据复制的大框架内，GBase 8s 数据库服务器几乎实现整个数据库服务器的透明的数据复制。复制一个数据库服务器管理的所有数据，并动态地在另一数据库服务器上更新，通常位于一远程站点。GBase 8s 数据库服务器的数据复制有时称为热站点备份，因为它提供一种维护整个数据库服务器的备份副本的方法，在发生灾难性故障时，可快速地使用它。

由于数据库服务器提供复制透明，因此您通常不需要关注或注意到数据复制；DBA 会处理它。然而，如果您的机构决定使用数据复制，则您应注意到，在数据复制环境中，存在对于客户机应用程序的特殊连接性事项。在 GBase 8s 管理员指南 中描述这些事项。

7.11 总结

通过数据库所有者授予您的权限来控制数据库访问。通常自动地授予您查询数据的权限，但通过特定的 Insert、Delete 和 Update 权限来控制修改数据的能力，以逐个表的方式授予这些权限。

如果对数据库施加数据完整性约束，则您的修改数据的能力受到那些约束的限制。您的数据库级别权限和表级别权限以及任何数据约束控制您可如何以及何时修改数据。此外，数据库的对象模式和违反检测特性也影响您可修改数据的方式，并有助于保持您的数据的完整性。

您可使用 DELETE 语句从表删除一行或多行。它的 WHERE 子句选择这些行；使用带有相同子句的 SELECT 语句来预览这些删除。

TRUNCATE 语句删除表的所有行。

使用 INSERT 语句将行添加到表。您可插入包含特定的列值的单个行，或可插入 SELECT 语句生成的一批行。

使用 UPDATE 语句来修改现有的行的内容。您使用可包括子查询的表达式来指定新的内容，以便您可使用基于其他表或更新了的表自身的数据。该语句有两种形式。在第一种形式中，您逐列地指定新值。在第二种形式中，SELECT 语句或记录变量生成一组新值。

使用 CREATE TABLE 和 ALTER TABLE 语句的 REFERENCES 子句来创建表之间的关系。REFERENCES 子句的 ON DELETE CASCADE 选项允许您使用一个 DELETE 语句来从父表和相关联的子表删除行。

使用事务来防止在修改过程中不可预测的中断，防止数据库处于不确定的状态。当在一事务内执行修改时，会在发生错误之后回滚它们。事务日志还扩展数据库的定期制作的备份副本。如果必须恢复数据库，则它可将数据库返回到最近的状态。

对用户为透明的数据复制提供另一种针对灾难性故障的保护。

8 在外部数据库中访问和修改数据

本部分总结访问不在当前数据库中的表和例程。

8.1 访问其他数据库服务器

通过限定数据库对象（表、视图、同义词或例程）的名称，您可访问外部数据库中的任何表或例程。

当外部表与当前数据库位于同一数据库服务器上时，您必须以数据库名称和冒号限定对象名称。例如，要引用不是本地数据库的数据库中的表，下列 SELECT 语句访问来自外部数据库的信息：

```
SELECT name, number FROM salesdb:contacts
```

在此示例中，查询从表 contacts 返回数据，该表在数据库 salesdb 中。

远程数据库服务器是不是当前数据库服务器的任何数据库服务器。当外部表在远程数据库服务器上时，您必须以数据库服务器名称和数据库名称来限定数据库对象的名称，如下例所示：

```
SELECT name, number FROM salesdb@distantserver:contacts
```

在此示例中，查询从表 contacts 返回数据，该表在远程数据库服务器 distantserver 上的数据库 salesdb 中。

要了解关于如何在外部数据库中指定数据库对象的语法和规则，请参阅《GBase 8s SQL 指南：语法》。

8.1.1 访问 ANSI 数据库

在 ANSI 数据库中，对象的所有者是对象名称的一部分： ownername.objectname。当当前数据库和外部数据库都是 ANSI 数据库时， 除非您是该对象的所有者，否则您必须包括所有者名称。 下列 SELECT 语句展示完全限定的表名称：

```
SELECT name, number FROM salesdb@aserver:ownername.contacts
```

提示： 您始终可“超限定”对象名。即，您可指定完全的对象名称，database@servername:ownername.objectname，即使在您不需要完全的对象名称的情况下。

8.1.2 在外部数据库服务器之间创建连接

您可在连接中使用相同的表示法。当您显式地指定数据库名称时，长的表名称可能会比较累赘，除非您使用别名来缩短它们，如下例所示：

```
SELECT O.order_num, C.fname, C.lname
        FROM masterdb@central:customer C, sales@boston:orders O
        WHERE C.customer_num = O.Customer_num
```

8.1.3 访问外部例程

要引用不是当前数据库服务器的数据库服务器上的例程，请以数据库服务器名称和数据库名称（以及所有者名称，如果远程数据库符合 ANSI 的话）来限定例程名称，如下列 SELECT 语句所示：

```
SELECT name, salesdb@boston:how_long()
        FROM salesdb@boston:contacts
```

8.2 对于远程数据库访问的限制

本部分总结对远程数据库访问的限制。

8.2.1 访问多个数据库的 SQL 语句

您可跨数据库和跨数据库服务器实例运行下列 SQL 语句：

- CREATE DATABASE
- CREATE SYNONYM
- CREATE VIEW
- DATABASE
- DELETE
- DROP DATABASE
- EXECUTE FUNCTION
- EXECUTE PROCEDURE
- INFO
- INSERT
- LOAD
- LOCK TABLE
- MERGE
- SELECT
- UNLOAD
- UNLOCK TABLE
- UPDATE

限制：

要跨数据库或跨数据库服务器成功地运行这些 SQL 语句中的每一个，本地数据库与外部数据库必须都具有相同的日志记录模式。例如，如果作为 MODE ANSI 创建了您从其发出

分布式查询的本地数据库，该查询访问的其他数据库都不可为无日志记录的，且不可使用显式的事务。

在跨数据库操作中返回数据类型

使用 SQL 语句或 UDR 来访问本地 GBase 8s 数据库服务器实例的其他数据库的分布式操作可访问这些数据类型的值：

- 非 opaque 的任何内建的原子数据类型
- BLOB、BOOLEAN、BSON、CLOB、JSON 和 LVARCHAR opaque 类型
- 基于内建的类型的 DISTINCT 类型
- 可强制转型为内建的类型的用户定义的数据类型（UDT）。

必须将上述 DISTINCT 或 UDT 值都显式地强制转型为内建的数据类型，且必须在所有参与的数据库中定义所有 DISTINCT 类型、UDT 和强制转型。

它们还可访问可强制转型为内建的类型的 UDT，假设显式地将 DISTINCT 或 UDT 值强制转型为内建的类型，以及在所有参与的数据库中定义的所有 DISTINCT 类型、UDT 和强制转型。

如果在所有参与的数据库中定义 UDR，则 SPL、C 和 Java 语言 UDR 可返回这些数据类型作为参数或作为返回值。必须跨所有参与的 GBase 8s 实例，来复制在这些数据类型之上定义的任何隐式的或显式的强制转型。DISTINCT 数据类型必须具有与在分布式查询中参与的所有数据库中定义的完全相同的数据类型层级结构。

跨数据库的分布式查询或其他访问本地 GBase 8s 数据库服务器的另一数据库的跨数据库 DML 操作将会失败并报错，如果它引用包括任何下列数据类型的列的表、视图或同义词的话：

- IMPEX
- IMPEXBIN
- LOLIST
- SENDRECV
- 以上列出的任何内建的 opaque 数据类型的 DISTINCT
- 复合的类型，包括 COLLECTION、LIST、MULTISET 或 SET，以及命名了的或未命名的 ROW 类型。

对于以这些内建的 opaque 或复合的数据类型访问表的跨数据库分布式操作的这一限制，也适用于访问两个或多个数据库服务器实例的数据库的操作，这在下一部分描述。

在跨服务器操作中的返回数据类型

跨两个或多个 GBase 8s 实例的分布式查询（或任何其他分布式 DML 操作或函数调用）不可返回复合的或大对象数据类型，也不可返回大部分 UDT 或 opaque 数据类型。跨服务器分布式查询、DML 操作和函数调用仅可返回下列数据类型：

- 任何非 opaque 的内建数据类型
- BOOLEAN
- BSON
- JSON
- LVARCHAR
- 非 opaque 的内建类型的 DISTINCT
- BOOLEAN 或 LVARCHAR 的 DISTINCT
- BSON 或 JSON 的 DISTINCT
- 在此列表中出现在上面的任何 DISTINCT 类型的 DISTINCT。

同样的跨数据库要求也适用于跨两个或多个 GBase 8s 数据库服务器实例的数据库的分布式 SQL 操作，即，在每个参与的数据库中，所有 UDR、强制转型和 DISTINCT 数据类型都要有相同的定义。

访问另一 GBase 8s 实例的数据库的跨服务器 DML 操作将会失败并报错。然而，如果它引用包括任意下列数据类型的表对象：

- BLOB
- CLIENTBINVAL
- CLOB
- IFX_LO_SPEC
- IFX_LO_STAT
- INDEXKEYARRAY
- POINTER
- RTNPARAMTYPES
- SELFUNCARGS
- STAT
- 用户定义的 OPAQUE 类型
- 罗列在上面的任何内建的 opaque 数据类型的 DISTINCT
- 复合的类型，包括 COLLECTION、LIST、MULTISET 或 SET，以及命名了的或未命名的 ROW 类型。

8.2.2 访问外部数据库对象

要访问外部数据库对象：

- 您必须持有对这些对象的适当的访问许可。

- 必须将两个数据库都设置为相同的语言环境。

重要： 分布式事务不可访问另一 GBase 8s 服务器实例的数据库中的对象，除非两个服务器实例都支持 TCP/IP 连接或 IPCSTR 连接，如同在它们的 DBSERVERNAME 或 DBSERVERALIASES 配置参数中以及在 sqlhosts 信息中定义的那样。此连接类型要求应用于 GBase 8s 数据库服务器实例之间的任何通信，即使两个数据库服务器都驻留在同一计算机上。

9 SQL 编程

前面的示例似乎将 SQL 作为一种交互的计算机语言；即，似乎您可以直接将 SELECT 语句输入数据库服务器内并查看回滚给您的数据行。

当然，情况并非如此。在您与数据库服务器之间存在许多软件层。在可显示数据之前，数据库服务器以必须格式化的二进制形式保留数据。它不会立即返回大量数据；当程序请求它时，它一次返回一行。

您可使用 DB-Access 通过交互的访问，通过以诸如 GBase 8s ESQL/C这样的 SQL API 编写的应用程序，或通过诸如 SPL 这样的应用程序语言，来访问您的数据库中的信息。

几乎所有程序都可包含 SQL 语句，执行它们，并从数据库服务器检索数据。本章节说明如何执行这些活动并指示您可如何编写执行它们的程序。

本章节介绍使用任何语言进行 SQL 编程的常见概念。在您可以特定的编程语言编写成功的程序之前，您必须先熟悉那种语言。然后，由于每种语言中处理的细节都不同，因此您必须熟悉特定于那种语言的 GBase 8s SQL API 的出版物。

9.1 程序中的 SQL

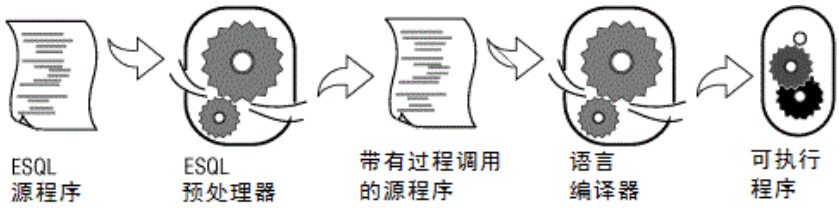
您可以任意几种语言编写程序，并将 SQL 语句混合在程序的其他语句之中，就如同它们是那种编程语言的一般语句似的。将这些 SQL 语句嵌入在程序中，且该程序包含嵌入式 SQL，其通常缩写为 ESQL。

9.1.1 SQL API 中的 SQL

ESQL 产品为 GBase 8s SQL API（应用程序编程接口）。 GBase 为 C 编程语言产生 SQL API。

下图展示 SQL API 产品如何工作。您编写您在其中将 SQL 语句处理作为可执行代码的源程序。嵌入式 SQL 预处理器处理您的源程序，它是一个定位嵌入式 SQL 语句并将它们转换为一系列过程调用和特殊的数据结构的程序。

图: 使用嵌入式 SQL 语句处理程序的概述



然后，转换了的源程序传递到编程语言编译器。在将它与静态的或动态的 SQL API 过程库相链接之后，编译器输出称为可执行的程序。当程序运行时，调用该 SQL API 库过程；它们与数据库服务器建立通信来执行 SQL 操作。

如果您将您的可执行程序链接到线程库包，则您可开发 GBase 8s ESQL/C 多线程应用程序。多线程应用程序可具有控制的许多线程。它将一个进程分割成多个执行线程，每一线程独立地运行。多线程的 GBase 8s ESQL/C 应用程序的主要优势在于，每一线程可同时具有与数据库服务器的许多活动的连接。而非线程的 GBase 8s ESQL/C 应用程序可创建与一个或多个数据库的许多连接，它一次仅可有一个连接是活动的。对于多线程的 GBase 8s ESQL/C 应用程序，每一线程可有一个活动的连接，且每个应用程序可有许多线程。

要获取关于多线程的应用程序的更多信息，请参阅《GBase 8s ESQL/C 程序员手册》。

9.1.2 应用程序语言中的 SQL

尽管 GBase 8s SQL API 产品允许您将 SQL 嵌入在主语言中，但某些语言包括 SQL 作为它们的语句集的固有部分。GBase 8s “存储过程语言”（SPL）使用 SQL 作为它的语句集的固有部分。您使用 SQL API 产品编写应用程序。您使用 SPL 编写例程，例程与数据库一起存储并从应用程序调用它。

9.1.3 静态的嵌入

您可通过静态的嵌入 或动态的嵌入将 SQL 语句引入程序。比较简单和常见的方式是通过静态的嵌入，其意味着编写 SQL 语句作为代码的一部分。该语句为静态的，是因为它们是源文本的固定部分。要获取关于静态的嵌入的更多信息，请参阅 检索单行 和 检索多行。

9.1.4 动态的语句

有些应用程序要求有动态地组合 SQL 语句的能力，以响应用户输入。例如，程序可能必须选择不同的列或将不同的标准应用到行，这依赖于用户想要什么。

使用动态的 SQL，程序在内存中将 SQL 语句组合成字符串，并将它传递给数据库服务器来执行。动态的语句不是代码的一部分；在执行期间在内存中构造它们。要了解更多信息，请参阅 动态 SQL。

9.1.5 程序变量和主变量

应用程序可在 SQL 语句内使用程序变量。在 SPL 中，当语法允许时，您可在 SQL 语句中放置程序变量。例如，DELETE 语句可在它的 WHERE 子句中使用程序变量。

下列代码示例展示 SPL 中的程序变量。

```
CREATE PROCEDURE delete_item (drop_number INT)
        ⋮
        DELETE FROM items WHERE order_num = drop_number
        ⋮
```

在使用嵌入式 SQL 语句的应用程序中，SQL 语句可引用程序变量的内容。在嵌入式 SQL 语句中命名的程序变量称为主变量，因为在程序中将该 SQL 语句认作是客人。

下列示例展示 DELETE 语句，当将它嵌入在 GBase 8s ESQL/C 源程序中时，它可能出现：

```
EXEC SQL delete FROM items
            WHERE order_num = :onum;
```

在此程序中，您看到常规的 DELETE 语句，如同 修改数据 描述的那样。当执行该 GBase 8s ESQL/C 程序时，删除 items 表的一行；还可删除多行。

该语句包含一个新的特性。它将 order_num 列与编写为 :onum 的一项相比较，这是主变量的名称。

SQL API 产品提供一种分隔主变量的名称的方式，当它们出现在 SQL 语句的上下文中时。在 GBase 8s ESQL/C 中，可使用美元符号（$）或冒号（:）来引入主变量。冒号是符合 ANSI 的格式。示例语句请求数据库服务器删除其中的订单编号等于名为 :onum 的主变量的当前内容的那些行。在程序中声明了此数值变量并提前分配了值。

在 GBase 8s ESQL/C 中，可使用前导的美元符号（$）或关键字 EXEC SQL 来引入 SQL 语句。

在前面的示例中说明的语法的差异很小； 重要的是，SQL API 和 SPL 语言使您执行下列任务：

- 在源程序中嵌入 SQL 语句，就好像它们是主语言的可执行语句一样。
- 以使用文字值的方式，在 SQL 表达式中使用程序变量。

如果您有编程经验，则您可立即看到这些可能性。在该示例中，将要删除的订单号码传到变量 onum 中。那个值来自程序可使用的任何源。可从文件读取它，程序可提示用户输入

它，或可从数据库读取它。DELETE 语句本身可为子例程的一部分（在此情况下，onum 可为该子例程的参数）；可一次或反复地调用该子例程。

总之，当您在程序中嵌入 SQL 语句时，您可对它们应用主语言的所有功能。您可将 SQL 语句隐藏在许多接口之下，且可以多种方式优化 SQL 函数。

9.2 调用数据库服务器

执行 SQL 语句本质上是作为子例程调用数据库服务器。必须将信息从程序传到数据库服务器，且必须将信息从数据库服务器返回到程序。

此通讯的部分是通过主变量完成的。您可将在 SQL 语句中命名的主变量视作对数据库服务器调用的过程的参数。在前面的示例中，主变量作为 WHERE 子句的参数。主变量收到数据库服务器返回的数据，如同 检索多行 描述的那样。

9.2.1 SQL 通信区域

数据库服务器始终在一个称为“SQL 通信区域”（SQLCA）的数据结构中返回结果代码，以及关于操作结果的其他可能信息。如果数据库服务器在用户定义的例程中执行 SQL 语句，则调用应用程序的 SQLCA 包含在该例程中 SQL 语句触发的值。

在从 表 1 至 表 1 中罗列 SQLCA 的主体字段。在编程语言之中，您用来描述诸如 SQLCA 这样的数据结构的语法，以及您用来应用其中字段的语法是不同的。要了解详细信息，请参阅您的 SQL API 出版物。

特别地，您通过其命名 SQLERRD 和 SQLWARN 数组的一个元素的下标是不同的。在 GBase 8s ESQL/C 中，数组元素从零开始编号，但在其他语言中，从一开始。在本讨论中，以诸如 third 这样的特定词命名字段，且您必须将这些词翻译成您的编程语言的语法。

您还可使用 GET DIAGNOSTICS 语句的 SQLSTATE 变量来检测、处理和诊断错误。请参阅 SQLSTATE 值。

9.2.2 SQLCODE 字段

SQLCODE 字段是数据库服务器的主要返回代码。在每个 SQL 语句之后，将 SQLCODE 设置为如下表所示的一个整数值。当那个值为零时，无误地执行该语句。特别地，当假定一个语句将数据返回到主变量内时，代码零意味着已返回了该数据且可使用它。任何非零代码都意味着相反的意思。未将有用的数据返回到了主变量。

表 1. SQLCODE 的值

返回值	解释

返回值	解释
值 < 0	指定一个错误代码。
值 = 0	指示成功。
0 < 值 < 100	在 DESCRIBE 语句之后，表示描述 SQL 语句的类型的一个整数值。
100	在未返回行的成功查询之后，指示 NOT FOUND 条件。在 INSERT INTO/SELECT、UPDATE、DELETE 或 SELECT... INTO TEMP 语句未能访问任何行之后，NOT FOUND 还可发生在符合 ANSI 的数据库中。

数据的结束

当正确地执行语句，但未找到行时，数据库服务器将 SQLCODE 设置为 100。在两种情况下可发生此情况。

第一种情况涉及使用游标的查询。(检索多行 描述使用游标的查询。)在这些查询中，FETCH 语句将来自活动集的每一值检索到内存内。检索最后一行之后，后续的 FETCH 语句不可返回任何数据。当发生此情况时，数据库服务器将 SQLCODE 设置为 100，指示数据的结束，找不到行。

第二种情况涉及不使用游标的查询。在此情况下，当没有行满足查询条件时，数据库服务器将 SQLCODE 设置为 100。在不符合 ANSI 的数据库中，仅不返回行的 SELECT 语句会导致将 SQLCODE 设置为 100。

在符合 ANSI 的数据库中，如果未返回行，则 SELECT、DELETE、UPDATE 和 INSERT 语句都将 SQLCODE 设置为 100。

负代码

在语句期间，当发生意外错误时，数据库服务器在 SQLCODE 中返回一个负数值来说明该问题。在联机错误消息文件中记录这些代码的含义。

9.2.3 SQLERRD 数组

在 SQLCODE 中可报告的某些错误代码反映一般的问题。数据库服务器可在 SQLERRD 的第二个字段中设置更详细的代码，显示数据库服务器 I/O 例程或操作系统遇到的错误。

将 SQLERRD 数组中的整数设置为跟在不同语句之后的不同值。在 GBase 8s ESQL/C 中，仅使用数组的第一个和第四个元素。下表展示如何使用这些字段。

表 1. SQLERRD 的字段

字段	解释
第一个	对于 SELECT、UPDATE、INSERT 或 DELETE 语句，在成功的 PREPARE 语句之后，或在打开 Select 游标之后，此字段包含估计的受影响的行数。
第二个	当 SQLCODE 包含一个错误代码时，此字段包含零或附加的错误代码，称为 ISAM 错误代码，说明主要错误的原因。在对单个行的成功的插入操作之后，此字段包含任何 SERIAL、BIGSERIAL 或 SERIAL8 值为那行生成的值。（然而，当通过表上的触发器，或通过视图上的 INSTEAD OF 触发器，将一序列列作为触发器的活动直接插入时，不更新此字段。）
第三个	在成功的多行插入、更新或删除操作之后，此字段包含处理了的行数。在以错误结束的多行插入、更新或删除操作之后，此字段包含在检测到该错误之前成功地处理了的行数。
第四个	在对于 SELECT、UPDATE、INSERT 或 DELETE 语句的成功的 PREPARE 语句之后，或在已打开了选择游标之后，此字段包含磁盘访问的与处理的全部行的估计加权总和。
第五个	在 PREPARE、EXECUTE IMMEDIATE、DECLARE 或静态的 SQL 语句中的语法错误之后，此字段包含检测到该错误的位置的语句文本的偏移量。
第六个	在对选择了的行的成功的访存之后，或成功的插入、更新或删除操作之后，此字段包含处理了的最后一行的 rowid（物理地址）。此 rowid 值是否对应于数据库服务器返回给用户的行，依赖于数据库服务器处理查询的方式，特别是对于 SELECT 语句。
第七个	保留。

这些附加的详细信息可是有用的。例如，您可使用第三个字段中的值来报告删除了或更新了多少行。当您的程序准备一个用户输入的 SQL 语句并发现错误时，第五个字段中的值使得您能够向用户显示错误的精确点。（当您在错误之后请求修改语句时，DB-Access 使用此特性来定位游标。）

9.2.4 SQLWARN 数组

将 SQLWARN 数组中的八个字符字段设置为空，或设置为 W 来指示各种特殊的情况。它们的含义依赖于刚刚执行的语句。

当数据库打开时，即，跟在 CONNECT、DATABASE 或 CREATE DATABASE 语句之后，出现一组警告标志。这些标志告诉您数据库的一些整体特征。

第二组标志跟在任何其他语句之后出现。这些标志反映在该语句期间发生的不寻常事件，这些事件通常没有严重到通过 SQLCODE 来反映的程度。

在下表中总结这两组 SQLWARN 值。

表 1. SQLWARN 的字段

字段	当打开或连接到数据库时	所有其他 SQL 操作
第一个	当将任何其他警告字段设置为 W 时，设置为 W。如果为空，则不需要检查其他的。	当将任何其他警告字段设置为 W 时，设置为 W。
第二个	当现在打开的数据库使用事务日志时，设置为 W。	如果截断列值，当使用 FETCH 或 SELECT...INTO 语句将它访存到主变量内时，设置为 W 。在 REVOKE ALL 语句上，当未取消全部七个表级别权限时，设置为 W。
第三个	当现在打开的数据库符合 ANSI 时，设置为 W。	当 FETCH 或 SELECT 语句返回为 NULL 的聚集函数（SUM、AVG、MIN、MAX）值时，设置为 W。
第四个	当数据库服务器为 GBase 8s 时，设置为 W。	在 SELECT ... INTO、FETCH ... INTO 或 EXECUTE ... INTO 语句上，当 projection 列表项数不同于在 INTO 子句中检索它们的主变量的数目时，设置为 W。在 GRANT ALL 语句上，当未授予全部七个表级别访问权限时，设置为 W。
第五个	当数据库服务器以 DECIMAL 形式存储 FLOAT 数据类型时，设置为 W。当主机系统缺乏对 FLOAT 类型的支持时，这样做。	如果准备好的对象包含不带有 WHERE 子句的 DELETE 语句或 UPDATE 语句时，设置为 W。
第六个	保留。	跟在不使用 ANSI 标准 SQL 语法的语句执行之后（假设设置了DBANSIWARN 环境变量），设置为 W。

字段	当打开或连接到数据库时	所有其他 SQL 操作
第七个	当将应用程序连接到在数据复制对中为辅助服务器的数据库服务器上时，设置为 W。即，该服务器仅对读取操作可用。	在查询处理期间(当 DATASKIP 特性为 on 时)，当已跳过了数据分片（dbspace）时，设置为 W。
第八个	当客户机 DB_LOCALE 与数据库语言环境不相匹配时，设置为W。要获取更多信息，请参阅《GBase 8s GLS 用户指南》。	当 SET EXPLAIN ON AVOID_EXECUTE 语句阻止查询执行时，设置为W。

9.2.5 SQLERRM 字符串

SQLERRM 可存储最多 72 字节的字符串。SQLERRM 字符串包含放置在错误消息里的标识符，诸如表名称。对于某些网络化的应用程序，它包含网络软件生成的错误消息。

如果由于违反约束导致 INSERT 操作失败，则将失败了的约束名称写到 SQLERRM。

提示： 如果错误字符串长于 72 字节，则静默地废弃溢出的部分。在某些上下文中，这可导致关于运行时错误的信息丢失。

9.2.6 SQLSTATE 值

某些 GBase 8s 产品，诸如 GBase 8s ESQL/C，支持符合 X/Open 和 ANSI SQL 标准的 SQLSTATE 值。在您运行 SQL 语句之后，GET DIAGNOSTICS 语句读取 SQLSTATE 值来诊断错误。数据库服务器以称为 SQLSTATE 的变量中存储的五个字符的字符串来返回结果代码。SQLSTATE 错误代码，或值，告诉您关于最近执行的 SQL 语句的下列信息：

- 该语句是否成功
- 该语句是否成功但生成了警告
- 该语句是否成功但未生成数据
- 该语句是否失败了

要获取关于 GET DIAGNOSTICS 语句、SQLSTATE 变量以及 SQLSTATE 返回代码的含义的更多信息，请参阅《GBase 8s SQL 指南：语法》中的 GET DIAGNOSTICS 语句。

提示： 如果您的 GBase 8s 产品支持 GET DIAGNOSTICS 和 SQLSTATE，则推荐您使用它们作为检测、处理和诊断错误的主要结构。使用 SQLSTATE 允许您检测多个错误，且它符合 ANSI。

9.3 检索单行

SELECT 语句返回的行集是它的活动集。单个 SELECT 语句返回单个行。您可使用嵌入式 SELECT 语句来从数据库将单个行检索到主变量内。然而，当 SELECT 语句返回多行数据时，程序必须使用游标来一次检索一行。在 检索多行 中讨论“多行”选择操作。

要检索单行数据，只要在您的程序中嵌入 SELECT 语句。下列示例展示您可如何使用 GBase 8s ESQL/C 来编写嵌入式 SELECT 语句：

```
EXEC SQL SELECT avg (total_price)
        INTO :avg_price
        FROM items
        WHERE order_num in
        (SELECT order_num from orders
        WHERE order_date < date('6/1/98') );
```

INTO 子句是将此语句与 编写 SELECT 语句 或 编写高级 SELECT 语句 中的任何示例区分开来的唯一细节。此子句指定要检索产生的数据的主变量。

当程序执行嵌入式 SELECT 语句时，数据库服务器执行该查询。示例语句选择聚集值，以便于它恰好产生一行数据。该行仅有单个列，且它的值存储在名为 avg_price 的主变量中。程序的后续行可使用那个变量。

您可使用此类语句来将单行数据检索到主变量内。单个行可有所期望的许多列。如果查询产生多行数据，则数据库服务器不可返回任何数据，而是返回一个错误代码。

您在 INTO 子句中罗列的主变量应与选择列表中的项一样多。如果这些列表的长度碰巧不一样，则数据库服务器返回尽可能多的值并在 SQLWARN 的第四个字段中设置警告标志。

9.3.1 数据类型转换

下列 GBase 8s ESQL/C 示例检索 DECIMAL 列的平均值，其自身是 DECIMAL 值。然而，将 DECIMAL 列的平均值放置其内的主变量不要求具有那种数据类型。

```
EXEC SQL SELECT avg (total_price) into :avg_price
        FROM items;
```

不展示在前面的 GBase 8s ESQL/C 代码示例中接收的变量 avg_price 的声明。该声明可为任一下列定义：

```
int avg_price;
        double avg_price;
        char avg_price[16];
        dec_t avg_price; /* typedef of decimal number structure */
```

注释语句中使用的每一主变量的数据类型，并使用该语句传到数据库服务器。数据库服务器尽量将列数据转换为接收的变量使用的形式。允许几乎任何转换，尽管某些转换会导致精度损失。依赖于接收的主变量的数据类型，前面的示例的结果会不同，如下表所示。

数据类型	结果
FLOAT	数据库服务器将十进制结果转换为 FLOAT，可能截断某些小数位。如果十进制的数量超过 FLOAT 格式的最大数量，则返回一个错误。
INTEGER	数据库服务器将结果转换为 INTEGER，如有必要会截断小数位。如果被转换的数值的整数部分与接收的变量不适合，则发生错误。
CHARACTER	数据库服务器将十进制值转换为 CHARACTER 字符串。如果对于接收的变量来说该字符串太长，则截断它。将 SQLWARN 的第二个字段设置为 W，且 SQLSTATE 变量中的值为 01004。

9.3.2 如果程序检索到 NULL 值，该怎么办？

可在数据库中存储 NULL 值，但编程语言支持的数据类型不识别 NULL 状态。程序必须采用某种方式来识别 NULL 项，以免将它作为数据来处理。

在 SQL API 中，指示符变量满足此需要。指示符变量是与可能收到 NULL 项的主变量相关联的一个附加的变量。当数据库服务器将数据放在主变量中时，它还在指示符变量中放置一个特殊的值来展示该数据是否为 NULL。在下列 GBase 8s ESQL/C 示例中，选择单个行，并将单个值检索到主变量 op_date 内：

```
EXEC SQL SELECT paid_date
        INTO :op_date:op_d_ind
        FROM orders
        WHERE order_num = $the_order;
        if (op_d_ind < 0) /* data was null */
        rstrdate ('01/01/1900', :op_date);
```

语句该值可能为 NULL，名为 op_d_ind 的指示符变量与该主变量相关联。（必须在程序中的其他地方将它声明为以短整数。）

跟在 SELECT 语句的执行之后，程序测试该指示符变量为负值。负值（通常为 -1）意味着检索到主变量内的值为 NULL。如果该变量为 NULL，则此程序使用 GBase 8s ESQL/C 库函数来将缺省的值指定给主变量。（函数 rstrdate 是 GBase 8s ESQL/C 产品的一部分。）

您用来将指示符变量与主变量相关联的语法，不同于您正在使用的语言，但在所有语言中，该原则是相同的。

9.3.3 处理错误

虽然数据库服务器自动地处理数据类型之间的转换，但使用 SELECT 语句仍可发生错误。在 SQL 编程中，如同在任何种类的编程中一样，您必须预见错误并随时为应对其做好准备。

数据的结束

通常会发生没有行满足查询的情况。在 SELECT 语句之后，通过 SQLSTATE 代码 02000 和 SQLCODE 中的代码 100 标志此事件。此代码指示一个错误或一般事件，这完全依赖于您的应用程序。如果您确信应有一行或多行满足查询（例如，如果您使用您刚从另一表的行读取的键值读取一行的话），则“数据的结束”代码表示在该程序的逻辑中的严重错误。在另一方面，如果您基于用户提供的键，或其他来源提供的不如程序可靠的键，则数据的缺乏可能是正常事件。

不符合 ANSI 的数据库的数据的结束

如果您的数据库不符合 ANSI，则跟在 SELECT 语句之后，仅在 SQLCODE 中设置“数据的结束”返回代码 100。此外，将 SQLSTATE 值设置为 02000。（诸如 INSERT、UPDATE 和 DELETE 这样的其他语句设置 SQLERRD 的第三个元素，来展示他们影响了多少行；通过 SQL 程序修改数据 讨论此主题。）

严重的错误

将 SQLCODE 设置为负值的错误，或将 SQLSTATE 设置为任何不以 00、01 或 02 开头的值的错误通常都很严重。您已开发好的程序或正在生产的程序几乎不应报告这些错误。然而，很难预料每一种有问题的情况，因此您的程序必须能够处理这些错误。

例如，查询可返回错误 -206，这意味着在该查询中指定的表不在数据库中。如果有人在编写了程序之后删除了该表，或如果该程序通过某种逻辑错误或输入错误打开了错误数据库，则会发生此情况。

使用聚集函数解释数据的结束

使用诸如 SUM、MIN 或 AVG 这样的聚集函数的 SELECT 语句总会成功地返回至少一行数据，即使当没有行满足 WHERE 子句时。基于行的空集的聚集值为空，但它仍然存在。

然而，如果聚集值是基于都包含空值的一行或多行，则聚集值也为空。如果您必须能够检测不基于任何行的聚集值与基于某些都是空的行的聚集值之间的差异，则您必须在该语句中包括 COUNT 函数和聚集值上的指示符变量。然后，您可得出下列情况。

计数值	指示符	具体情况

计数值	指示符	具体情况
0	-1	选择了零行
>0	-1	选择了某些行；全部为空
>0	0	选择了一些非空行

缺省值

您可以许多方式处理这些不可避免的错误。在某些应用程序中，使用比执行功能更多的行代码来处理错误。然而，在本部分的示例中，最简单的解决方案之一，缺省值，应奏效，如下例所示：

```
avg_price = 0; /* 设置错误的缺省值 */
        EXEC SQL SELECT avg (total_price)
        INTO :avg_price:null_flag
        FROM items;
        if (null_flag < 0) /* 可能没有行 */
        avg_price = 0; /* 设置 0 行的缺省值 */
```

前面的示例处理下列事项：

- 如果查询选择一些非空的行，则返回并使用正确的值。这是期望的并且最常用的结果。
- 如果查询没有选择行，或在发生的可能性很小的情况下，仅选择在 total_price 列（从不应为空的列）中有空值的那些行，则设置该指示符变量，并指定缺省值。
- 如果发生任何严重的错误，则保持主变量不变；它包含最初设置的缺省值。在程序中的此点，程序员无需捕获这类错误并报告它们。

9.4 检索多行

当存在查询可返回多行的任何可能时，程序必须以不同的方式执行该查询。分两个阶段处理多行查询。首先，程序启动查询。（不立即返回数据。）然后，程序一次请求一行数据。

使用称为游标的特殊数据对象来执行这些操作。游标是表示查询的当前状态的数据结构。下表展示程序操作的一般的顺序：

1. 程序声明游标及其相关联的 SELECT 语句，其只是分配持有该游标的存储。
2. 程序打开游标，其启动相关联的 SELECT 语句的执行，并检测其中的任何错误。

3. 程序将一行数据访存到主变量内，并处理它。
4. 在访存最后一行之后，程序关闭游标。
5. 当不再需要该游标时，程序释放游标来释放它使用的资源。

使用名为 DECLARE、OPEN、FETCH、CLOSE 和 FREE 的 SQL 语句执行这些操作。

9.4.1 声明游标

您使用 DECLARE 语句来声明游标。此语句给游标一个名称，指定它的使用，并将它与语句相关联。下列示例是用 GBase 8s ESQL/C 编写的：

```
EXEC SQL DECLARE the_item CURSOR FOR
        SELECT order_num, item_num, stock_num
        INTO :o_num, :i_num, :s_num
        FROM items
        FOR READ ONLY;
```

声明给游标一个名称（在此示例中为 the_item）并将它与 SELECT 语句相关联。（通过 SQL 程序修改数据 讨论还可如何与 INSERT 语句相关联。）

此示例中的 SELECT 语句包含 INTO 子句。INTO 子句指定哪个变量接收数据。您还可使用 FETCH 语句来指定哪个变量接收数据，如 定位 INTO 子句讨论的那样。

DECLARE 语句不是活动的语句；它仅仅创建游标的特性并为它分配存储。您可使用在前面示例中声明的游标来通读 items 表一次。可声明向后读和向前读游标（请参阅 游标输入模式）。由于此游标缺少 FOR UPDATE 子句，且由于指定 FOR READ ONLY，因此，它仅用于读取数据，不修改它。通过 SQL 程序修改数据 说明如何使用游标来修改数据。

9.4.2 打开游标

当程序准备使用游标时，它打开它。OPEN 语句激活游标。它将相关联的 SELECT 语句传给数据库服务器，其开始搜索相匹配的行。数据库服务器处理该查询至定位到或构造输出的第一行的位置。它并不真正地返回那行数据，但它确实在 SQLSTATE 中和在 SQLCODE 中为 SQL API 设置返回代码。下列示例展示 GBase 8s ESQL/C 中的 OPEN 语句：

```
EXEC SQL OPEN the_item;
```

由于数据库服务器正在第一次查看查询，因此，它可能检测到一些错误。在程序打开游标之后，它应测试 SQLSTATE 或 SQLCODE。如果 SQLSTATE 值大于 02000 或 SQLCODE 包含负值，则该游标不可用。在 SELECT 语句中可能出现错误，或某些其他问题可能阻止数据库服务器执行该语句。

如果 SQLSTATE 等于 00000，或 SQLCODE 包含零，则 SELECT 语句在语法上是有效的，且准备使用该游标。然而，此时，该程序不知道游标能否产生任何行。

9.4.3 访存行

程序使用 FETCH 语句来检索输出的每一行。此语句命名游标，且还可命名接收该数据的主变量。下列示例展示完整的 GBase 8s ESQL/C 代码：

```
EXEC SQL DECLARE the_item CURSOR FOR
        SELECT order_num, item_num, stock_num
        INTO :o_num, :i_num, :s_num
        FROM items;
        EXEC SQL OPEN the_item;
        while(SQLCODE == 0)
        {
        EXEC SQL FETCH the_item;
        if(SQLCODE == 0)
        printf("%d, %d, %d", o_num, i_num, s_num);
        }
```

检测数据的结束

在前面的示例中，WHILE 条件在 OPEN 语句返回错误时阻止执行循环。当将 SQLCODE 设置为 100 来标志数据的结束时，相同的条件会终止该循环。然而，该循环包含 SQLCODE 的测试。此测试是必需的，因为如果 SELECT 语句是有效的但找不到相匹配的行，则 OPEN 语句返回零，但第一次访存返回 100（数据的结束）并不返回任何数据。下列示例展示编写同一循环的另一种方式：

```
EXEC SQL DECLARE the_item CURSOR FOR
        SELECT order_num, item_num, stock_num
        INTO :o_num, :i_num, :s_num
        FROM items;
        EXEC SQL OPEN the_item;
        if(SQLCODE == 0)
        EXEC SQL FETCH the_item;        /* fetch 1st row*/
        while(SQLCODE == 0)
        {
        printf("%d, %d, %d", o_num, i_num, s_num);
        EXEC SQL FETCH the_item;
        }
```

在此版本中，早已处理了无返回行的情况，因此，在循环中不存在第二次 SQLCODE 测试。由于 SQLCODE 测试的时间成本是访存成本的很小一部分，因此这些版本在性能上没有多大差异。

定位 INTO 子句

INTO 子句命名要接收数据库服务器返回的数据的主变量。INTO 必须出现在 SELECT 或 FETCH 语句中。然而，它不可同时出现在两个语句中。下列示例指定 FETCH 语句中的主变量：

```
EXEC SQL DECLARE the_item CURSOR FOR
        SELECT order_num, item_num, stock_num
        FROM items;
        EXEC SQL OPEN the_item;
        while(SQLCODE == 0)
        {
        EXEC SQL FETCH the_item INTO :o_num, :i_num, :s_num;
        if(SQLCODE == 0)
        printf("%d, %d, %d", o_num, i_num, s_num);
        }
```

此形式允许您将不同的行访存到不同的位置内。例如，您可以使用此形式来将连续的行访存到数组的连续元素内。

9.4.4 游标输入模式

为了输入，游标以顺序的或滚动的两种模式中的一种运行。顺序的游标仅可访存序列中的下一行，因此，每一次打开游标，顺序的游标仅可通读表一次。滚动游标可访存下一行或任何输出行，因此，滚动游标可多次读取相同的行。下列示例展示在 GBase 8s ESQL/C 中声明的顺序的游标。

```
EXEC SQL DECLARE pcurs cursor for
        SELECT customer_num, lname, city
        FROM customer;
```

在打开游标之后，仅可使用检索下一行数据的顺序的访存来使用它，如下例所示：

```
EXEC SQL FETCH p_curs into:cnum, :clname, :ccity;
```

每一顺序的访存返回一个新行。

使用关键字 SCROLL CURSOR 声明滚动游标，如来自 GBase 8s ESQL/C 的下列示例所示的那样：

```
EXEC SQL DECLARE s_curs SCROLL CURSOR FOR
```

```
        SELECT order_num, order_date FROM orders
        WHERE customer_num > 104
```

使用不同的访存选项来使用滚动游标。例如，ABSOLUTE 选项指定要访存的行的绝对行位置。

```
EXEC SQL FETCH ABSOLUTE :numrow s_curs
        INTO :nordr, :nodat
```

此语句访存在主变量 numrow 中给出其位置的行。您还可在此访存当前的行，或您可访存第一行然后再次扫描所有行。然而，这些特性可能导致应用程序运行得更慢，如下一部分描述的那样。要了解适用于滚动游标的附加的选项，请参阅《GBase 8s SQL 指南：语法》中的 FETCH 语句。

9.4.5 游标的活动集

一旦打开游标，就意味着选择一些行。查询产生的所有行的集合称为该游标的活动集。可以简单地将活动集视为定义良好的行的集合，且将游标视为指向该集合的一行。只要没有其他程序正在并发地修改同一数据，就会发生此情况。

创建活动集

当打开游标时，数据库服务器进行有必要的任何操作来定位选择了的数据的第一行。依赖于该查询的叙述方式，此活动可很容易，或者它可需要大量的工作和时间。请考虑下列游标的声明：

```
EXEC SQL DECLARE easy CURSOR FOR
        SELECT fname, lname FROM customer
        WHERE state = 'NJ'
```

由于在简单的方式中此游标仅需要单个表，因此，数据库服务器快速地确定是否有任何行满足该查询，并标识第一行。第一行是该游标此次找到的唯一一行。该活动集中余下的行仍然未知。作为对比，请考虑下列游标的声明：

```
EXEC SQL DECLARE hard SCROLL CURSOR FOR
        SELECT C.customer_num, O.order_num, sum (items.total_price)
        FROM customer C, orders O, items I
        WHERE C.customer_num = O.customer_num
        AND O.order_num = I.order_num
        AND O.paid_date is null
        GROUP BY C.customer_num, O.order_num
```

通过连接三个表并将输出行分组，生成此游标的活动集。优化器可能能够使用索引来以正确的顺序产生这些行，但通常情况下，ORDER BY 或 GROUP BY 子句的使用要求在可确定哪一行标识第一行之前，数据库服务器生成所有行，将它们复制到临时表并对该表排序。

在活动集全部生成并保存在临时表中的情况下，数据库服务器可花费相当多的时间来打开游标。接着，数据库服务器可以确切地告诉程序活动集包含多少行。然而，此信息不可用。一个原因是您永远不可确定优化器会使用哪种模式。如果优化器可避免排序和临时表，则它会这样做；但在查询中、在表的大小方面或在可用的索引方面的小更改都可更改优化器的方式。

顺序游标的活动集

数据库服务器尝试使用尽可能少的资源来维护游标的活动集。如果它可这么做，则数据库服务器从不保留下次访存的单个行之外的行。它可为大部分顺序的游标这么做。对于每一访存，它都返回当前行的内容并定位下一行。

SCROLL 游标的活动集

必须保留 SCROLL 游标的活动集中的所有行，直到游标关闭为止，因为数据库服务器不可确定程序下一次会请求哪一行。

更常见的是，数据库服务器将滚动游标的活动集作为临时表来实现。然而，数据库服务器可能不立即填充此表（除非它创建了临时表来处理该查询）。通常当打开游标时，它创建临时表。然后，第一次访存行时，数据库服务器将它复制到临时表内并将它返回到程序。当第二次访存行时，可从临时表取得它。如果在程序访存所有行之前，它放弃该查询，则此方案使用最少的资源。不创建或保存从不访存的行。

活动集和并发

当仅一个程序正在使用数据库时，活动集的成员不可更改。大多数个人计算机都是这种情况，且是要考虑的最简单情况。但必须为了在多编程系统中使用来设计一些程序，在此，两个、三个或几十个不同的程序可同时在相同的表上工作。

在您的游标是打开的时，当其他程序可更新表时，活动集的思路用处不大了。您的程序在某一时刻仅看到一行数据，但表中的所有其他行可能正在更改。

在简单查询的情况下，当数据库服务器仅持有活动集的一行时，任何其他行都可更改。在您的程序访存行之后的那一刻，另一程序可删除同一行或更新它，于是，如果在此检查它，它不再是活动集的一部分。

当在临时表中保存活动集或它的一部分时，旧数据可出现问题。即，从其派生活动集行的实际的表中的行可更改。如果真是这样，某些活动集行不再反映当前的表内容。

最初这些想法令人不安，但只要您的程序仅读取数据，就不存在旧数据，更确切地说，所有数据都同样陈旧。活动集是数据在某一时刻的快照。第二天行就不一样了；如果它在下一毫秒也不一样，倒无所谓。换言之，在程序正在运行时发生的更改，与该程序终止的那一刻保存和应用的更改之间，没有实际的差异。

旧数据可导致问题的唯一时刻，是当程序打算使用输入的数据来修改同一数据库时；例如，当银行业应用程序必须读取账户余额、更改它并将它写回时。通过 SQL 程序修改数据 讨论修改数据的程序。

9.4.6 部件爆炸问题

当您使用由程序逻辑补充的游标时，您可解决普通的 SQL 不可解决的问题。这些问题之一就是部件爆炸问题，有时称为材料单处理。此问题的核心是对象之间的递归关系；一个对象包含其他对象，其又包含其他对象。

通常以制造库存为例来说明该问题。例如，公司制造各种部件。有些部件是分立的，但有些是其他部件的组合。

在可能称为 contains 的单个表中说明这些关系。列 contains.parent 持有系组合的部件的部件编号。列 contains.child 具有为父部件的组件的部件的部件编号。如果部件编号 123400 是九个部件的组合，则存在九行，123400 在第一列中，其他部件编号在第二列中。下图展示描述部件编号 123400 的多行中的一行。

图: 部件爆炸问题

CONTAINS

PARENT	CHILD
FKNN	FKNN
123400 432100	432100 765899

部件爆炸问题在于：给定一个部件编号，产生为那个部件的组件的所有部件的列表。下列示例是一种解决方案的概要，如以 GBase 8s ESQL/C 实现的那样：

```
int part_list[200];

boom(top_part)
int top_part;
{
    long this_part, child_part;
    int next_to_do = 0, next_free = 1;
    part_list[next_to_do] = top_part;

    EXEC SQL DECLARE part_scan CURSOR FOR
        SELECT child INTO child_part FROM contains
```

```
            WHERE parent = this_part;
    while(next_to_do < next_free)
    {
        this_part = part_list[next_to_do];
        EXEC SQL OPEN part_scan;
        while(SQLCODE == 0)
        {
            EXEC SQL FETCH part_scan;
            if(SQLCODE == 0)
            {
                part_list[next_free] = child_part;
                next_free += 1;
            }
        }
        EXEC SQL CLOSE part_scan;
        next_to_do += 1;
    }
    return (next_free - 1);
}
```

从技术上讲，contains 表的每一行都是有向无环图，或树，的头结点。该函数执行对该树的宽度优先搜索，树根是作为它的参数传递的部件编号。该函数使用名为 part_scan 的游标返回在 parent 列中带有特定的值的所有行。最内层的 while 循环打开 part_scan 游标，在选择集中访存每一行，并当已检索了每一组件的部件编号时，关闭该游标。

此函数解决部件爆炸问题的核心，但该函数不是完整的解决方案。例如，它不允许组件在树中出现多个级别。此外，实际的 contains 表还会有列count，给出在每一 parent 中使用的 child 部件的计数。返回每一组件部件的总计数的程序要复杂得多。

之前描述的迭代方法不是解决部件爆炸问题的唯一方法。如果代的数目有固定的限制，则您可使用嵌套的外部自连接，以单个 SELECT 语句解决该问题。

如果在一个最高级别部件内，可包含最多四代部件，则下列 SELECT 语句返回所有部件：

```
SELECT a.parent, a.child, b.child, c.child, d.child
    FROM contains a
        OUTER (contains b,
            OUTER (contains c, outer contains d) )
    WHERE a.parent = top_part_number
        AND a.child = b.parent
        AND b.child = c.parent
```

```
        AND c.child = d.parent
```

此 SELECT 语句为来源于指定为 top_part_number 的祖先的每一行返回一行。对于不存在的级别，返回 Null 值。（请使用指示符变量来检测它们。）要将此解决方案扩展到更多级别，请选择 contains 表的附加的嵌套外部连接。您还可修订此解决方案来返回每一级别上部件的数目的计数。

9.5 动态 SQL

虽然静态 SQL 是有用的，但在您编写程序的时候，它要求您知道每个 SQL 语句确切内容。例如，您必须确切说明在任何 WHERE 子句中测试哪些列，以及在任何选择列表中重命名哪些列。

当您编写程序来执行定义良好的任务时，不存在任何问题。但不可事先完善地定义某些程序的数据库任务。特别地，必须响应交互用户的程序可能需要根据用户输入的内容来组合 SQL 语句。

动态 SQL 允许程序在执行期间形成 SQL 语句，因此，用户输入决定该语句的内容。以下列步骤执行此活动：

1. 程序将 SQL 语句的文本组装为一个字符串，将该字符串存储在程序变量中。
2. 它执行 PREPARE 语句，请求数据库服务器测试该语句文本并为执行来准备它。
3. 它使用 EXECUTE 语句来执行该准备好的语句。

这样，基于任何种类的用户输入，程序可构造并然后使用任何 SQL 语句。例如，它可读取一个 SQL 语句的文件并准备和执行每一语句。

DB-Access 是一个您可用来交互地探索 SQL 的实用程序，它是一个动态地构造、准备和执行 SQL 语句的 GBase 8s ESQL/C 程序。例如，DB-Access 允许您使用简单的、交互式的菜单来指定表的行。当您完成时，DB-Access 动态地构建必需的 CREATE TABLE 或 ALTER TABLE 语句并准备和执行它。

9.5.1 准备语句

在形式上，动态 SQL 语句像任何其他写入程序的 SQL 语句一样，除了它不可包含任何主变量的名称之外。

准备好的 SQL 语句有两个限制。首先，如果它是 SELECT 语句，则它不可包括 INTO variable 子句。INTO variable 子句指定将列数据放入其内的主变量，而不允许在准备好的对象的文本中使用主变量。其次，不论主变量的名称通常出现在表达式中的任何位置，都将问号（?）写作 PREPARE 语句中的占位符。仅 PREPARE 语句可指定问号（?）占位符。

您可使用 PREPARE 语句以此形式为执行准备语句。使用 GBase 8s ESQL/C编写下列示例：

```
EXEC SQL prepare query_2 from
        'SELECT * from orders
        WHERE customer_num = ? and order_date > ?';
```

此示例中的两个问号指示当执行该语句时，在那两个位置使用主变量的值。

您可动态地准备几乎任何 SQL 语句。您唯一不可准备的语句就是与动态 SQL 和游标管理直接相关联的语句，诸如 PREPARE 和 OPEN 语句。在您准备 UPDATE 或 DELETE 语句之后，最好测试 SQLWARN 的第五个字段来查看您是否使用了 WHERE 子句（请参阅 SQLWARN 数组）。

准备语句的结果是表示该语句的数据结构。此数据结构与产生它的字符串不一样。在 PREPARE 语句中，您赋予该数据结构一个名称；它是前面示例中的 query_2。使用此名称来执行准备好的 SQL 语句。

PREPARE 语句不将字符串限制于一个语句。它可包含多个用分号分隔的 SQL 语句。下列示例展示用 GBase 8s ESQL/C 编写的相当复杂的事务：

```
strcpy(big_query, "UPDATE account SET balance = balance + ?
        WHERE customer_id = ?; \ UPDATE teller SET balance =
        balance + ? WHERE teller_id = ?;");
        EXEC SQL PREPARE big1 FROM :big_query;
```

当执行此语句的列表时，主变量必须为六个占位的问号提供值。虽然设置多语句列表更为复杂，但由于在程序与数据库服务器之间发生更少的交换，因此性能往往更好。

9.5.2 执行准备好的 SQL

在您准备语句之后，您可多次执行它。使用 EXECUTE 语句执行不是 SELECT 语句的那些语句，以及仅返回一行的 SELECT 语句。

下列 GBase 8s ESQL/C 代码准备并执行银行账户的多语句更新：

```
EXEC SQL BEGIN DECLARE SECTION;
        char bigquery[270] = "begin work;";
        EXEC SQL END DECLARE SECTION;
        stcat ("update account set balance = balance + ? where ", bigquery);
        stcat ("acct_number = ?;', bigquery);
        stcat ("update teller set balance = balance + ? where ", bigquery);
        stcat ("teller_number = ?;', bigquery);
        stcat ("update branch set balance = balance + ? where ", bigquery);
        stcat ("branch_number = ?;', bigquery);
        stcat ("insert into history values(timestamp, values);", bigquery);
```

```
EXEC SQL prepare bigq from :bigquery;

EXEC SQL execute bigq using :delta, :acct_number, :delta,
:teller_number, :delta, :branch_number;

EXEC SQL commit work;
```

EXECUTE 语句的 USING 子句提供主变量的列表，以其值替代准备好的语句中的问号。如果 SELECT（或 EXECUTE FUNCTION）仅返回一行，则您可使用 EXECUTE 的 INTO 子句来指定接收这些值的主变量。

9.5.3 动态主变量

支持动态地分配数据对象的 SQL API 进动态语句更进一步。它们允许您动态地分配接收列数据的主变量。

变量的动态分配使得有可能从程序输入中取到任意的 SELECT 语句，确定它产生多少个值及其数据类型，并分配适当类型的主变量来保存它们。

此功能的关键是 DESCRIBE 语句。它取到准备好的 SQL 语句的名称，并返回关于该语句及其内容的信息。它设置 SQLCODE 来指定语句的类型；即，它开头的动词。如果准备好的语句是 SELECT 语句，则 DESCRIBE 语句还返回关于选择了的输出数据的信息。如果准备好的语句是 INSERT 语句，则 DESCRIBE 语句返回关于输入参数的信息。DESCRIBE 语句返回信息的数据结构是预定义的数据结构，为此用途分配该数据结构并称为系统描述符区域。如果您正在使用 GBase 8s ESQL/C，则可使用系统描述符区域，或作为 sqlda 结构的替代方案。

DESCRIBE 语句返回的或为 SELECT 语句引用的数据结构包括结构的一个数组。每一结构描述为选择类表中一个项目的数据。程序可检测该数组并发现包括十进制值、某长度的字符值以及整数的一行数据。

利用此信息，程序可分配保存接收了的数据的内存，并为要使用的数据库服务器将必要的指针放置在该数据结构中。

9.5.4 释放准备好的语句

准备好的 SQL 语句占据内存中的空间。对于某些数据库服务器，它可消耗数据库服务器拥有的空间以及属于该程序的空间。当程序终止时，释放此空间，但通常您应在使用完此空间时就释放它。

您可使用 FREE 语句来释放此空间。FREE 语句采用语句的名称，或为语句名称声明了的游标的名称，并释放分配给准备好的语句的空间。如果在该语句上定义多个游标，则释放该语句不会释放游标。

9.5.5 快速执行

对于不需要游标或主变量的简单语句，您可将 PREPARE、EXECUTE 和 FREE 语句组合到单个操作内。下列示例展示 EXECUTE IMMEDIATE 语句如何获取字符串、准备它、执行它，并释放一个操作中的存储：

```
EXEC SQL execute immediate 'drop index my_temp_index';
```

此能力使得编写简单 SQL 语句更轻松。然而，由于不允许使用 USING 子句，EXECUTE IMMEDIATE 语句不可用于 SELECT 语句。

9.6 嵌入数据定义语句

数据定义语句是创建数据库和修改表的定义的 SQL 语句，通常将这些语句放在程序内。原因是很少执行它们。数据库只创建一次，但会多次查询和更新它。

使用 DB-Access，一般都是交互地完成数据库及其表的创建。还可从语句的文件运行这些工具，因此可使用操作系统命令来完成数据库创建。在 GBase 8s SQL 指南：语法中描述数据定义语句。

9.7 授予和撤销应用程序中的权限

反复地执行与数据定义相关的一个任务：授予和撤销权限。由于必须频繁地授予和撤销权限，有可能是由不熟悉 SQL 的用户操作，因此一种策略是将 GRANT 和 REVOKE 语句打包在程序中，来提供给他们更简单、更方便的用户接口。

GRANT 和 REVOKE 语句特别适合于动态 SQL。每一语句采用下列参数：

- 一个或多个权限的列表
- 表名称
- 用户的名称

您可能至少需要基于程序输入（来自于用户、命令行参数或文件）提供这些值的一部分，但都不可以主变量的形式提供。这些语句的语法不允许在任何位置使用主变量。

一个替代方案是将语句的各个部分组合到字符串内，并准备和执行组合好的语句。程序输入可作为字符串合并到准备好的语句内。

下列 GBase 8s ESQL/C 函数从参数组合 GRANT 语句，然后准备和执行它：

```
char priv_to_grant[100];
        char table_name[20];
        char user_id[20];

        table_grant(priv_to_grant, table_name, user_id)
        char *priv_to_grant;
        char *table_name;
        char *user_id;
        {
        EXEC SQL BEGIN DECLARE SECTION;
        char grant_stmt[200];
        EXEC SQL END DECLARE SECTION;

        sprintf(grant_stmt, " GRANT %s ON %s TO %s",
        priv_to_grant, table_name, user_id);
        PREPARE the_grant FROM :grant_stmt;
        if(SQLCODE == 0)
        EXEC SQL EXECUTE the_grant;
        else
        printf("Sorry, got error # %d attempting %s",
        SQLCODE, grant_stmt);

        EXEC SQL FREE the_grant;
        }
```

下列示例展示的该函数的打开语句指定它的名称及其三个参数。这三个参数指定要授予的权限、对其授予权限的表的名称，以及要接收它们的用户的 ID。

```
table_grant(priv_to_grant, table_name, user_id)
        char *priv_to_grant;
        char *table_name;
        char *user_id;
```

该函数使用下例中的语句来定义本地变量 grant_stmt，使用其来组合并保存 GRANT 语句：

```
EXEC SQL BEGIN DECLARE SECTION;
        char grant_stmt[200];
        EXEC SQL END DECLARE SECTION;
```

如下例所示，通过将该语句的常量部分与函数参数连起来，创建该 GRANT 语句：

```
sprintf(grant_stmt, " GRANT %s ON %s TO %s",priv_to_grant,
```

```
        table_name, user_id);
```

此语句将下列六个字符串连接起来：

- 'GRANT'
- 指定要授予的权限的参数
- 'ON'
- 指定表名称的参数
- 'TO'
- 指定用户的参数

结果是部分由程序输入组成的完整的 GRANT 语句。PREPARE 语句将组合的语句文本传给数据库服务器进行解析。

如果数据库服务器跟在 PREPARE 语句之后在 SQLCODE 中返回错误代码，则该函数显示错误消息。如果数据库服务器认可该语句的形式，则它设置零返回值。此活动并不保证正确地执行该语句；它仅意味着该语句的语法正确。它可能引用不存在的表或某些种类仅可在执行期间才能检测到的错误。该示例的下列部分在执行之前检查 the_grant 是否准备成功了：

```
if(SQLCODE == 0)
        EXEC SQL EXECUTE the_grant;
        else
        printf("Sorry, got error # %d attempting %s", SQLCODE, grant_stmt);
```

如果准备成功，则 SQLCODE = = 0，下一步执行准备好的语句。

9.7.1 指定角色

或者，DBA 可使用 CREATE ROLE 语句定义一个角色，并使用 GRANT 和 REVOKE 语句来将角色分配给用户或取消，以及授予和撤销角色权限。例如：

```
GRANT engineer TO nmartin;
```

需要 SET ROLE 语句来激活非缺省的角色。要获取关于角色和权限的更多信息，请参阅 访问管理策略 和 对数据库级对其对象的权限。要获取关于这些语句的语法的更多信息，请参阅《GBase 8s SQL 指南：语法》。

9.8 总结

可将 SQL 语句写入程序内，如同它们是不同的编程语言的语句那样。可在 WHERE 子句中使用程序变量，可将来自数据库的数据访存到它们之内。预处理器将 SQL 代码翻译为过程调用和数据结构。

编写不返回数据的语句，或仅返回一行数据的查询，就像该语言的普通命令语句一样。可返回多行的查询与表示当前数据行的游标相关联。通过游标，程序可根据需要访存数据的每一行。

将静态 SQL 语句写入程序的文本内。然而，程序在它运行时动态地形成新的 SQL 语句，并执行它们。在最先进的情况下，程序可获得关于查询返回的列的数目和类型，并动态地分配内存空间来保存它们。

10 通过 SQL 程序修改数据

前一章节描述如何将 SQL 语句插入或嵌入到其他语言编写的程序之内，特别是 SELECT 语句。嵌入式 SQL 使程序能够从数据库检索数据行。

本章节讨论当程序需要删除、插入或更新行来修改数据库时发生的问题。如在 SQL 编程中那样，此章节为您阅读您的 GBase 8s 嵌入式语言出版物做准备。

在 修改数据 中讨论 INSERT、UPDATE 和 DELETE 语句的常规用法。本章节从程序之内检验它们的使用。您可方便地将语句嵌入在程序中，但难以处理错误和处理来自多个程序的并发修改。

10.1 DELETE 语句

要从表删除行，程序执行 DELETE 语句。DELETE 语句可以常规方式以 WHERE 子句指定行，或它可引用单个行，通过指定的游标访存最后一行。

每当您删除行时，您必须考虑其他表中的行是否依赖于删除了的行。在 修改数据 中论述协调删除的这个问题。当从程序内删除时，问题是一样的。

10.1.1 直接删除

您可在程序中嵌入 DELETE 语句。下列示例使用 GBase 8s ESQL/C：

```
EXEC SQL delete from items
            WHERE order_num = :onum;
```

您还可动态地准备和执行同样形式的语句。在任一情况下，该语句直接作用于数据库来影响一行或多行。

示例中的 WHERE 子句使用名为 onum 的主变量的值。通常在该操作之后，将结果发布在 SQLSTATE 中以及在 sqlca 结构中。即使发生错误，SQLERRD 数组的第三个元素也包含删除的行的计数。SQLCODE 中的值展示操作的完全成功。如果该值不是负的，则未发生错误且 SQLERRD 的第三个元素是满足了 WHERE 子句并被删除了的所有行的计数。

直接删除期间的错误

当发生错误时，该语句提前终止。SQLSTATE 中的值和 SQLCODE 中的值以及 SQLERRD 的第二个元素说明它的原因，且行的计数显示删除了的行数。对于许多错误，由于错误阻止数据库服务器开始操作，因此那个计数为零。例如，如果不存在命名的表，或如果重命名 WHERE 子句中测试的列，则不会尝试任何删除。

然而，在操作开始且处理某些行之后，可发现某些错误。这些错误中最常见的是锁冲突。在数据库服务器可删除那行之前，它必须获取行上的排他锁。其他程序可能正在使用来自该表的行，阻止该数据库服务器锁定行。由于锁定的问题影响所有类型的修改，因此在 对多用户环境编程 中讨论它。

另外，在删除开始之后，可出现较少见的错误类型。例如，在更新数据库时发生硬件错误。

事务日志记录

在修改期间为任何种类的错误做准备的最好方式是使用事务日志记录。万一发生错误，您可告诉数据库服务器将数据库恢复原样。下列示例是基于直接删除 部分中的示例的，将其扩展为使用事务：

```
EXEC SQL begin work;                    /* 开启事务 */
        EXEC SQL delete from items
        where order_num = :onum;
        del_result = sqlca.sqlcode;          /* 保存两个错误 */
        del_isamno = sqlca.sqlerrd[1];       /* 代码编号 */
        del_rowcnt = sqlca.sqlerrd[2];       /* 以及行的计数 */
        if (del_result < 0)                  /* 发现的问题：*/
        EXEC SQL rollback work;              /* 恢复一切 */
        else                                    /* 一切正常：*/
        EXEC SQL commit work;                /* 结束事务 */
```

此示例中的关键在于，在程序结束该事务之前，它将重要的返回值保存在 sqlca 结构中。像其他 SQL 语句一样，ROLLBACK WORK 和 COMMIT WORK 语句都在 sqlca 结构中设置返回代码。然而，如果您想要报告错误生成的代码，则必须在执行 ROLLBACK WORK 之前保存它们。ROLLBACK WORK 语句移除所有暂挂的事务，包括它的错误代码。

使用事务的优势在于，不管发生什么错误，数据库都处于已知的、可预测的状态。不存在修改完成了多少的问题；要不就是都完成了，要不就是都没完成。

在带有日志记录的数据库中，如果用户未启动一个显式的事务，则数据库服务器在语句执行之前初始化一个内部的事务，并在执行完成或失败后终止该事务。如果语句执行成功，则提交该内部的事务。如果语句失败，则回滚该内部的事务。

协调的删除

当您必须修改多个表时，事务日志记录的用处特别明显。例如，考虑从演示数据库删除一个订单的问题。在该问题的最简单的形式中，您必须从两个表 orders 和 items 同时删除行，如下列 GBase 8s ESQL/C 的示例所示：

```
EXEC SQL BEGIN WORK;
            EXEC SQL DELETE FROM items
            WHERE order_num = :o_num;
            if (SQLCODE >= 0)
            {
            EXEC SQL DELETE FROM orders
            WHERE order_num == :o_num;
{
              if (SQLCODE >= 0)
              EXEC SQL COMMIT WORK;
{
                 else
                 {
                 printf("Error %d on DELETE", SQLCODE);
                 EXEC SQL ROLLBACK WORK;
                 }
```

不论是否使用事务，此程序的逻辑都很相似。如果未使用它们，则看到错误消息的人员更难作出决定。依赖于错误发生的时间，下列情况中的一种适用：

- 未执行删除；此订单的所有行都保留在数据库中。
- 删除了某些商品行，但不是全部；仅保留某些商品的订单记录。
- 删除了所有商品行，但保留订单行。
- 删除了所有行。

在第二种和第三种情况下，数据库受到一定程度的损害；它包含可导致某些查询产生错误结果的部分信息。您必须小心行事来恢复信息的一致性。当使用事务时，会防止所有这些不确定性。

10.1.2 使用游标删除

您还可使用游标编写 DELETE 语句来删除最后访存了的行。以此方式删除行允许您的程序基于在 WHERE 子句中不可测试的条件进行删除。由于设置事务的开始和结束的方式的缘故，下列示例仅适用于不符合 ANSI 的数据库

警告： 此示例中的 GBase 8s ESQL/C 函数的设计是不安全的。它依赖于正确的操作的当前隔离级别。该章节稍后讨论隔离级别。要获取关于隔离级别的更多信息，请参阅 对多用户环境编程。即使当该函数按期望的方式运行时，它的影响也依赖于表中行的物理顺序，这样做通常并不理想。

```
int delDupOrder()
        {
        int ord_num;
        int dup_cnt, ret_code;

        EXEC SQL declare scan_ord cursor for
        select order_num, order_date
        into :ord_num, :ord_date
        from orders for update;
        EXEC SQL open scan_ord;
        if (sqlca.sqlcode != 0)
        return (sqlca.sqlcode);
        EXEC SQL begin work;
        for(;;)
        {
        EXEC SQL fetch next scan_ord;
        if (sqlca.sqlcode != 0) break;
        dup_cnt = 0; /* default in case of error */
        EXEC SQL select count(*) into dup_cnt from orders
        where order_num = :ord_num;
        if (dup_cnt > 1)
        {
        EXEC SQL delete from orders
        where current of scan_ord;
        if (sqlca.sqlcode != 0)
        break;
        }
        }
        ret_code = sqlca.sqlcode;
        if (ret_code == 100)                    /* merely end of data */
```

```
        EXEC SQL commit work;
        else        /* error on fetch or on delete */
        EXEC SQL rollback work;
        return (ret_code);
        }
```

该函数的目的是删除包含重复的订单号码的行。实际上，在演示数据库中，orders.order_num 列有唯一约束，因此，其中不可出现重复的行。然而，可为另一数据库编写一个类似的函数；这一个使用熟悉的列名称。

该函数声明游标 scan_ord 来扫描 orders 表中的所有行。使用 FOR UPDATE 子句声明它，说明该游标可修改数据。如果该游标正确地打开，则该函数开始一个事务，然后对表的行进行循环。对于每一行，它使用嵌入式 SELECT 语句来确定该表的多少行具有当前行的订单编号。如果没有正确的隔离级别，此步骤失败，如 对多用户环境编程 描述的那样。）

在演示数据库中，使用它在此表上的唯一约束，返回到 dup_cnt 的计数始终为一。然而，如果它更大，则该函数删除表的当前行，将重复的计数减少一个。

有时需要这类清理函数，但它们一般需要更复杂的设计。此函数删除所有重复的行，除了数据库服务器返回的最后一行之外。那个顺序对这些行的内容及其含义没有任何关系。您或许可通过将 ORDER BY 子句添加到游标声明来提升前面例子中函数的性能。然而，您不可同时使用 ORDER BY 与 FOR UPDATE。插入示例 提供一种更好的方法。

10.2 INSERT 语句

您可在程序中嵌入 INSERT 语句。它的形式和在程序中的使用与 修改数据 中描述的一样，带有您可在表达式中使用主变量的附加的特性，在 VALUES 和 WHERE 子句中都一样。此外，您在程序中有使用游标来插入行的附加能力。

10.2.1 插入游标

DECLARE CURSOR 语句有许多种变体。大部分用于为不同种类的数据扫描创建游标，但有一种变体创建特殊种类的游标，称为插入游标。您可使用 PUT 和 FLUSH 语句来插入游标，以便高效地将行批量插入到表内。

声明插入游标

要创建插入游标，请为 INSERT 语句而不是 SELECT 语句声明游标。您不可使用这样的游标来访存数据行；您仅可使用它来插入它们。

当您打开插入游标时，在内存中创建缓冲区来保存一块行。当程序产生数据行时，该缓冲区接收它们；然后，当缓冲区满时，将它们以块的形式传到数据库服务器。该缓冲区减小

程序与数据库服务器之间的通信量，允许数据库服务器比较容易地插入行。因此，插入操作更快。

该缓冲区始终足够大，以保持至少两行插入了的值。当这些行比最小缓冲区大小还小时，它大到足以保存超过两行的值。

使用游标来插入

前面示例中的代码（声明插入游标）为使用准备插入游标。如下例所示，接下来演示如何使用该游标。为了简化起见，此示例假设名为 next_cust的函数或者返回关于新客户的信息，或者返回空数据来标志输入的结束。

```
EXEC SQL BEGIN WORK;
        EXEC SQL OPEN new_custs;
        while(SQLCODE == 0)
        {
        next_cust();
        if(the_company == NULL)
        break;
        EXEC SQL PUT new_custs;
        }
        if(SQLCODE == 0)                    /* 如果 PUT 没有问题 */
        {
        EXEC SQL FLUSH new_custs;           /* 写留下的任何行 */
        if(SQLCODE == 0)                    /* 如果 FLUSH 没有问题 */
        EXEC SQL COMMIT WORK;               /* 提交更改 */
        }
        else
        EXEC SQL ROLLBACK WORK;             /* 否则，取消更改 */
```

此示例中的代码反复地调用 next_cust，当它返回非空数据时，PUT 语句将返回的数据发送到该行缓冲区。当缓冲区填满时，自动地将它包含的行发送到数据库服务器。当 next_cust 没有更多数据返回时，该循环正常结束。然后，FLUSH 语句写入缓冲区中余下的任何行，之后，事务终止。

重新检查关于 INSERT 语句的信息。请参阅 INSERT 语句。该语句本身不是游标定义的一部分，它将单个行插入到 customer 表内。实际上，可从示例代码删除整个插入游标的装置，且可将 INSERT 语句写到 PUT 语句正所在的位置。不同之处在于，插入游标导致程序运行得更快些。

PUT 和 FLUSH 之后的状态代码

当程序执行 PUT 语句时，程序应测试是否成功地将该行放入缓冲区中。如果新的行适合该缓冲区，则 PUT 的唯一操作就是将该行复制到缓冲区。在此情况下不可发生错误。然而，如果该行不适合，则为了插入将整个缓冲区负载传到数据库服务器，可发生错误。

返回到“SQL 通信区域”（SQLCA）内的值为程序提供它需要的信息，来整理每一种情况。如果未发生错误，则在每个 PUT 语句之后，将 SQLCODE 和 SQLSTATE 设置为零，如果发生错误，则设置为负的错误代码。

数据库服务器将 SQLERRD 的第三个元素设置为实际插入到表内的行数，如下

- 零，如果仅将新行移至缓冲区
- 缓冲区中的行数，如果插入缓冲区负载而未发生错误
- 错误发生之前插入了的行数，如果发生错误

请再次阅读代码来了解如何使用 SQLCODE（请参阅前面的示例）。首先，如果 OPEN 语句发生错误，则由于 WHILE 条件失败，不执行该循环，则不执行 FLUSH 操作，且该事务回滚。其次，如果 PUT 语句返回一个错误，则由于 WHILE 条件的缘故结束该循环，不执行 FLUSH 操作，且该事务回滚。仅当该循环至少一次生成足够的行来填充缓冲区，此条件才可发生。否则，PUT 语句不可生成错误。

程序可能结束该循环，而这些行还在缓冲区中，可能未插入任何行。此时，SQL 状态为零，且发生 FLUSH 操作。如果 FLUSH 操作产生一错误代码，则该事务回滚。仅当成功地执行所有操作，才提交该事务。

10.2.2 常量行

插入游标机制支持一种特殊的情况，在此易于获得高性能。在此情况下，罗列在 INSERT 语句中的所有值都是常量：不罗列表达式和主变量，仅罗列文字数值和字符串。不管发生多少次这样的 INSERT 操作，它产生的行都是相同的。当这些行是相同的时，复制、缓冲和传输每一相同的行都是没有意义的。

但是，对于此类 INSERT 操作，PUT 语句除了增加计数器之外不进行任何操作。当最终执行 FLUSH 操作时，将该行的单个副本以及插入的计数传到数据库服务器。数据库服务器创建并在一个操作中插入许多行。

您通常不插入一些相同的行。当您首次建立此数据库来操作带有空数据的大型表时，您可插入相同的行。

10.2.3 插入示例

使用游标删除 包含一个 DELETE 语句的示例，其目的在于查找并删除表的重复的行。执行此任务的更好方式是选择期望的行，而不是删除不期望的行。下列 GBase 8s ESQL/C 示例中的代码展示执行此任务的一种方法：

```
EXEC SQL BEGIN DECLARE SECTION;
        long last_ord = 1;
        struct {
        long int o_num;
        date        o_date;
        long        c_num;
        char        o_shipinst[40];
        char        o_backlog;
        char        o_po[10];
        date        o_shipdate;
        decimal     o_shipwt;
        decimal     o_shipchg;
        date        o_paiddate;
        } ord_row;
        EXEC SQL END DECLARE SECTION;

        EXEC SQL BEGIN WORK;
        EXEC SQL INSERT INTO new_orders
        SELECT * FROM orders main
        WHERE 1 = (SELECT COUNT(*) FROM orders minor
        WHERE main.order_num = minor.order_num);
        EXEC SQL COMMIT WORK;

        EXEC SQL DECLARE dup_row CURSOR FOR
        SELECT * FROM orders main INTO :ord_row
        WHERE 1 < (SELECT COUNT(*) FROM orders minor
        WHERE main.order_num = minor.order_num)
        ORDER BY order_date;
        EXEC SQL DECLARE ins_row CURSOR FOR
        INSERT INTO new_orders VALUES (:ord_row);

        EXEC SQL BEGIN WORK;
        EXEC SQL OPEN ins_row;
        EXEC SQL OPEN dup_row;
        while(SQLCODE == 0)
        {
        EXEC SQL FETCH dup_row;
```

```
        if(SQLCODE == 0)
        {
        if(ord_row.o_num != last_ord)
        EXEC SQL PUT ins_row;
        last_ord = ord_row.o_num
        continue;
        }
        break;
        }
        if(SQLCODE != 0 && SQLCODE != 100)
        EXEC SQL ROLLBACK WORK;
        else
        EXEC SQL COMMIT WORK;
        EXEC SQL CLOSE ins_row;
        EXEC SQL CLOSE dup_row;
```

此示例以一个常规的 INSERT 语句开始，该语句查找该表的所有非重复的行，并将它们插入到另一表内，假定在程序启动之前已创建了该表。那个操作仅留下重复的行。（在演示数据库中，orders 表有唯一约束，不可有重复的行。假设此示例处理的是其他数据库。）

然后，前面示例中的代码声明两个游标。第一个称为 dup_row，返回表中的重复的行。由于 dup_row 仅用于输入，因此，它可使用 ORDER BY 子句来强制一些重复的顺序，而不是在 使用游标删除 页上的示例中使用的物理记录顺序。在此示例中，按重复的行的日期对它们排序，保留最早的日期，但您可基于该数据使用任何其他的顺序。

第二个游标 ins_row 是插入游标。此游标利用该能力来使用 C 结构 ord_row，以支持该行中所有列的值。

剩余的代码检查通过 dup_row 返回的行。它将来自每一组重复的行中的第一行插入到新表内，并忽略其余的。

为了简洁起见，前面的示例使用最简单的错误处理类型。如果在已处理了所有行之前发生错误，则该样例代码回滚活动的事务。

多少行受到了影响？

当您的程序使用游标来选择行时，它可测试 SQLCODE 是否为 100（或 SQLSTATE 是否为 02000），即“数据的结束”返回代码。设置此代码来指示没有行或没有更多的行满足该查询条件。对于不符合 ANSI 的数据库，仅跟在 SELECT 语句之后在 SQLCODE 或 SQLSTATE 中设置“数据的结束”返回代码；不跟在 DELETE、INSERT 或 UPDATE 语句之后使用它。对于符合 ANSI 的数据库，对于不影响任何行的更新、删除和插入操作，也将 SQLCODE 设置为 100。

找不到数据的查询是不成功的。然而，仍将碰巧未更新或插入行的 UPDATE 或 DELETE 语句视为成功。它更新了或删除了它的 WHERE 子句表明它应更新或产出的行集；然而，该集合为空。

同样地，即使当插入了的行的来源是 SELECT 语句，且该 SELECT 未选择任何行，INSERT 语句也不设置“数据的结束”返回代码。因为该 INSERT 语句插入了要求它插入的行数（即，零行），因此，该语句成功。

要了解插入了、更新了或删除了多少行，程序可测试 SQLERRD 的第三个元素。行的计数在那里，这与 SQLCODE 中的值（零还是负的）无关。

10.3 UPDATE 语句

您可以 修改数据 描述的任何形式将 UPDATE 语句嵌入在程序中，附加的特性是，您可同时在 SET 和 WHERE 子句中命名表达式中的主变量。此外，程序可更新游标找到的行。

10.3.1 更新游标

更新游标允许您删除或更新当前行；即，最近访存的行。使用 GBase 8s ESQL/C 编写的下列示例展示更新游标的声明：

```
EXEC SQL
        DECLARE names CURSOR FOR
        SELECT fname, lname, company
        FROM customer
        FOR UPDATE;
```

使用此游标的程序可以常规的方法访存行。

```
EXEC SQL
        FETCH names INTO :FNAME, :LNAME, :COMPANY;
```

如果该程序然后决定需要修改该行，则它可这么做。

```
if (strcmp(COMPANY, "SONY") ==0)
        {
        EXEC SQL
        UPDATE customer
        SET fname = 'Midori', lname = 'Tokugawa'
        WHERE CURRENT OF names;
        }
```

关键字 CURRENT OF names 替代 WHERE 子句中的常规测试表达式。在其他方面，UPDATE 语句保持不变，即使包括表名称的规范，在游标名称中其为隐式的，但仍然需要。

关键字 UPDATE 的用途

游标中的关键字 UPDATE 的用途是让数据库服务器可更新（或删除）它访存的任何行。数据库服务器在通过更新游标访存的行上放置较多的需求锁，而当它访存未使用那个关键字声明的游标的行时，放置较少的需求锁。此操作使常规的游标有较好的性能，以及在多处理系统中更高的并发使用级别。（对多用户环境编程 讨论锁和并发使用的级别。）

更新特定的列

下列示例已更新了前面的更新游标示例的特定的列：

```
EXEC SQL
        DECLARE names CURSOR FOR
        SELECT fname, lname, company, phone
        INTO   :FNAME,:LNAME,:COMPANY,:PHONE FROM customer
        FOR UPDATE OF fname, lname
        END-EXEC.
```

仅可通过此游标更新 fname 和 lname 列。作为错误拒绝如下语句：

```
EXEC SQL
        UPDATE customer
        SET company = 'Siemens'
        WHERE CURRENT OF names
        END-EXEC.
```

如果程序尝试这样的更新，则返回错误代码且不发生更新。由于删除影响所有列，因此，也拒绝使用 WHERE CURRENT OF 的删除尝试。

不总是需要的 UPDATE 关键字

SQL 的 ANSI 标准不提供游标定义中的 FOR UPDATE 子句。当程序使用符合 ANSI 的数据库时，它可使用任何游标来更新或删除。

10.3.2 清理表

如何使用更新游标的一个最终的假设示例提出一个问题，使用已建立的数据库不应出现该问题，但在应用程序的初始设计阶段可能出现。

在该示例中，创建并操纵一个名为 target 的大型表。字符列 dactyl 无意中获得一些空值。应删除这些行。此外，使用 ALTER TABLE 语句将新列serials 添加到表。此列将安装唯一的整数值。下列示例展示您用来完成这些任务的 GBase 8s ESQL/C 代码：

```
EXEC SQL BEGIN DECLARE SECTION;
```

```
char dcol[80];
short dcolint;
int sequence;
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE target_row CURSOR FOR
SELECT datcol
INTO :dcol:dcolint
FROM target
FOR UPDATE OF serials;
EXEC SQL BEGIN WORK;
EXEC SQL OPEN target_row;
if (sqlca.sqlcode == 0) EXEC SQL FETCH NEXT target_row;
for(sequence = 1; sqlca.sqlcode == 0; ++sequence)
{
if (dcolint < 0) /* null datcol */
EXEC SQL DELETE WHERE CURRENT OF target_row;
else
EXEC SQL UPDATE target SET serials = :sequence
WHERE CURRENT OF target_row;
}
if (sqlca.sqlcode >= 0)
EXEC SQL COMMIT WORK;
else EXEC SQL ROLLBACK WORK;
```

10.4 总结

程序可执行 INSERT、DELETE 和 UPDATE 语句，如同 修改数据 描述的那样。程序还可使用游标来扫描整个表，更新或删除选择了的行。它还可使用游标来插入行，这样做的好处是缓冲这些行，并以块为单位发送到数据库服务器。

在所有这些活动中，您必须确保当发生错误时，程序会检测错误并将数据库返回到一已知的状态。实现这一点的最重要的工具是事务日志记录。没有事务日志记录，更难以编写可从错误恢复的程序。

11 对多用户环境编程

本部分描述当您在多用户环境中工作时需要注意的几个编程问题。

如果您的数据库包含在单个用户工作站中，且不访问来自另一计算机的数据。则您的程序可任意修改数据。在所有其他情况下，您必须考虑一种可能性，即，在您的程序正在修改数据时，另一程序正在读取或修改同一数据。将这种情况描述为并发：同一时刻对相同数据的两处或多处独立的使用。本部分讨论并发、锁定和隔离级别。

本部分还描述语句高速缓存特性，它可减少每一会话的内存分配，并加速查询处理。语句高速缓存存储那些稍后在使用相同的 SQL 语句的不同的用户会话之中共享的数据。

11.1 并发和性能

在多编程系统中，要有好的性能，并发至关重要。为了保证某一时刻仅一个程序可使用数据，当访问序列化的数据时，处理会显著减慢。

11.2 锁定和完整性

除非对数据的使用作出控制，否则，并发可导致许多负面效果。程序可读取过时的数据，或可丢失所做的修改，即使表面上已经完成了它们。

要防止此类错误，数据库服务器强加一个锁定系统。锁定是程序可在数据块上放置的声明或保留。只要锁定数据，数据库服务器保证没有其他程序可修改它。当另一程序请求该数据时，数据库服务器或者让该程序等待，或者让其返回并报错。

11.3 锁定和性能

由于锁定序列化对一块数据的访问，因此，它减少并发；任何想要访问该数据的其他程序都必须等待。数据库服务器可在单个行、磁盘页、整个表或整个数据库上放置锁。（磁盘页可能保存多行，且一行可能需要多个磁盘页。）它越是放置锁，它锁定的对象越大，并发降得越低。锁越少，锁定的对象越小，并发和性能越高。

下列部分讨论您可如何使您的程序实现下列目标：

- 放置所有必要的锁以确保数据完整性。
- 锁定与前面的目标可能相一致的最少、最小的数据块。

11.4 并发问题

要理解并发的危险性，您必须从多个程序的方面考虑，每一程序以它自己的速度执行。 假设您的程序通过下列游标正在访存行：

```
EXEC SQL DECLARE sto_curse CURSOR FOR
        SELECT * FROM stock
        WHERE manu_code = 'ANZ';
```

将每一行从数据库服务器传送到程序都花费时间。在传送期间和传送之间，其他程序可执行其他数据块操作。大约在您的程序访存由那个查询产生的行的同一时刻，另一用户的程序可能执行下列更新：

```
EXEC SQL UPDATE stock
        SET unit_price = 1.15 * unit_price
        WHERE manu_code = 'ANZ';
```

换句话说，两个程序都在通读同一个表，一个正在访存某些行，而另一个正在更改相同的行。可能出现下列情况：

1. 在您的程序访存它的第一行之前，其他程序完成了它的更新。

   您的程序仅向您显示更新了的行。

2. 在其他程序有机会更新它之前，您的程序访存了每行。

   您的程序仅向您显示原始的行。

3. 在您的程序访存某些原始的行之后，其他程序赶上并继续更新您的程序还要读取的一些行；然后，它执行 COMMIT WORK 语句。

   您的程序可能返回的既有原始的行，也有更新了的行。

4. 与第 3 种情况一样，除了在更新表之后，另一程序发出 ROLLBACK WORK 语句之外。

   您的程序可向您显示的既有原始的行，也有在数据库中不再存在的更新了的行。

前两种可能性是无害的。在第 1 种可能的情况中，在您的查询开始之前完成了更新。更新是在一毫秒之前结束的，还是一周前结束的，并无差异。

在第 2 种可能的情况中，实际上，您的查询在更新开始之前完成。其他程序可能紧跟在您的程序之后只处理了一行，或它可能直到明晚才开始；这没有关系。

然而，后两种可能的情况对于某些应用程序的设计可至关重要。在第 3 种可能的情况下，查询返回的既有更新了的数据，也有原始数据。在某些应用程序中，那种结果可能是有害的。在其他程序中，诸如计算所有价格的平均值，根本无关紧要。

由于取消了它们的事务，如果程序返回一些在表中不可再找到的数据行，则第 4 种可能的情况是灾难性的。

当您的程序使用游标来更新或删除最后访存的数据时，需要引起另外的关注。下列的事件序列产生错误的结果：

- 您的程序访存该行。
- 另一程序更新或删除该行。
- 您的程序更新或删除 WHERE CURRENT OF cursor_name。

要控制诸如此类的事件，请使用数据库服务器的锁定和隔离级别特性。

11.5 锁定如何工作

GBase 8s 数据库服务器支持复杂的、灵活的锁定特性的集合，本部分中描述这些主题。

11.5.1 锁的种类

下表展示 GBase 8s 数据库服务器对不同情况支持的锁的类型。

锁类型	用途
共享的	共享锁为了只读保留它的对象。它防止在该锁保留期间更改该对象。多个程序可在同一对象上放置共享锁。在以共享的模式锁定记录时，多个对象可读取该记录。
排他的	排他锁为了单个程序的使用保留它的对象。当程序打算更改该对象时，使用此锁。 您不可在存在任何其他种类锁的地方放置排他锁。在您放置排他锁之后，您不可在同一对象上放置另一锁。
可提升的（或更新）	可提升的（或更新）锁有更新的意图。您仅可在不存在其他可提升锁或排他锁的地方放置它。您可在已有共享锁的记录上放置可提升锁。当程序要更改该锁定了的对象时，您可将该可提升锁提升为排他锁，但仅当该锁可能从可提升的更改为排他的时刻，该记录上没有包括共享锁在内的其他锁时。如果当设置了可提升锁时，共享锁在该记录上，则在将可提升锁提升为排他锁之前，您必须删除共享锁。

11.5.2 锁作用域

您可将锁应用于整个数据库、整个表、磁盘页、单个行或索引键值。正被锁定的对象的大小称之为锁的作用域（也称为锁颗粒度）。通常，锁的作用域越大，并发降得越低，但编程越简单。

数据库锁

您可锁定整个数据库。打开数据库的操作在数据库的名称上放置一共享锁。使用 CONNECT、DATABASE 或 CREATE DATABASE 语句打开数据库。只要程序有一数据库是打开的，则该名称上的共享锁防止任何其他程序删除该数据库或在其上放置排他锁。

下列语句展示您可能如何排他地锁定整个数据库：

```
DATABASE database_one EXCLUSIVE
```

如果没有其他程序已打开了那个数据库，则此语句成功。放置该锁之后，其他程序不可打开该数据库，即使是读取也不可，因为它在该数据库名称上放置共享锁的尝试会失败。

仅当关闭该数据库时，才释放数据库锁。可使用 DISCONNECT 或 CLOSE DATABASE 语句来显式地执行那个操作，或通过执行另一 DATABASE 语句来隐式地执行。

由于锁定数据库导致那个数据库中的并发降低为零，因此，它使得编程非常简单；不可发生并发效果。然而，仅当其他程序不需要访问时，才应锁定数据库。在非高峰期间，对数据进行大量更改之前，通常使用数据库锁定。

表锁

您可锁定整个表。在某些情况下，数据库服务器自动地执行此操作。您还可使用 LOCK TABLE 语句来显式地锁定整个表。

LOCK TABLE 语句或数据库服务器可放置下列类型的表锁：

共享锁

任何用户都不可写表。在共享模式下，数据库服务器在表上放置一个共享锁，其通知其他用户不可执行更新。此外，数据库服务器为每个更新了的、删除了的或插入了的行添加锁。

排他锁

任何其他用户不可从该表读取或写该表。在排他模式下，数据库服务器仅在该表上放置一个排他锁，不论它更新多少行。排他的表锁防止该表的任何并发使用，因此，如果其他程序正在争夺对该表的使用，则可严重影响性能。然而，当您需要更新表中的大部分行时，请在表上放置排他锁。

使用 LOCK TABLE 语句来锁定表

事务告诉数据库服务器通过 LOCK TABLE 语句来使用表级别锁定。下列示例展示如何在表上放置排他锁：

```
LOCK TABLE tab1 IN EXCLUSIVE MODE
```

下列示例展示如何在表上放置共享锁：

```
LOCK TABLE tab2 IN SHARE MODE
```

提示： 在提供更大的并发时，您可为数据库服务器设置隔离级别来获得与共享的表锁相同的保护程度。

数据库服务器何时自动地锁定表

在数据库服务器对任何下列语句执行操作时，它总是锁定整个表：

ALTER FRAGMENT

ALTER INDEX

ALTER TABLE

CREATE INDEX

DROP INDEX

RENAME COLUMN

RENAME TABLE

完成该语句（或事务结束）会释放该锁。在某些查询期间也可自动地锁定整个表。

使用 ONLINE 关键字来避免表锁定

对于未以 IN TABLE 关键字选项定义的索引，当您使用 ONLINE 关键字来 CREATE 或 DROP 索引时，您可最小化索引了的表上的排他锁的持续时间。

在联机创建或删除索引时，不支持表上的 DDL 操作，但当发出了 CREATE INDEX 或 DROP INDEX 语句时，可完成并发了的操作。直到没有其他进程正在并发地访问该表时，才创建或删除指定的索引。然后，短暂地保持锁来写与该索引相关联的系统目录数据。这提高了系统的可用性，因为通过正在进行的会话或新会话仍可读该表。下列语句展示如何使用 ONLINE 关键字来避免在使用 CREATE INDEX 语句时的自动表锁定：

```
CREATE INDEX idx_1 ON customer (lname) ONLINE;
```

然而，对于使用 IN TABLE 关键字选项定义的“表中”索引，在包括 ONLINE 关键字的 CREATE INDEX 或 DROP INDEX 操作期间，该索引了的表保持锁定。其他会话尝试访问该锁定了的表时，会失败并发出这些错误之一：

```
107: ISAM error:    record is locked.
        211: Cannot read system catalog (systables).
        710: Table (table.tix) has been dropped, altered or renamed.
```

行和键锁

您可锁定表的一行。在其他程序继续处理同一表的其他行时，程序可锁定一行或选择的多行。

行和键锁定不是缺省的行为。当您创建表时，您必须指定行级别锁定。下列示例创建带有行级别锁定的表：

```
CREATE TABLE tab1
        (
        col1...
        ) LOCK MODE ROW;
```

当您创建表时，如果指定 LOCK MODE 子句，则您可稍后使用 ALTER TABLE 语句来更改锁定模式。下列语句将保留表上的锁定模式更改为页级别锁定：

```
ALTER TABLE tab1 LOCK MODE PAGE
```

在某些情况下，数据库服务器必须锁定不存在的行。要这样做，数据库服务器在索引键值上放置一个锁。使用键锁与使用行锁一样。当表使用行锁定时，作为假想行上的锁来实现键锁。当表使用页锁定时，在包含该键或如果存在则包含该键的索引页上放置一键锁。

当您插入、更新或删除键（当您插入、更新或删除行时，自动地执行）时，数据库服务器在该索引的键上创建锁。

当您更新较少行数时，由于它们提高并发性，因此行和键锁通常提供最佳性能。然而，数据库服务器在获取锁时会造成一些开销。

当通过排他锁来锁定表中的一行或多行时，对其他用户的影响部分地依赖于它们的事务隔离级别。其隔离级别不是 Dirty Read 的其他用户可能遇到事务失败，这是由于在指定的时限内，未释放了排他锁。

对于 Committed Read 或 Dirty Read 隔离级别操作，这些操作尝试访问那些并发的会话已在其上设置了排他的行级别锁的表，为了降低锁定冲突的风险，可启用事务来读取锁定了的行中数据的最近提交的版本，而不是等待提交或回滚设置该锁的那个事务。启用对排他地锁定了的行的最后提交的版本的访问，可以采用几种方式来实现：

- 对于个别的会话，发出此 SQL 语句

  ```
  SET ISOLATION TO COMMITTED READ LAST COMMITTED;
  ```

- 对于使用 Committed Read 或 Read Committed 隔离级别的所有会话，DBA 可将 USELASTCOMMITTED 配置参数设置为 'ALL' 或设置为 'COMMITTED READ'。
- 对于使用 Committed Read 或 Read Committed 隔离级别的个别的会话，任何用户都可发出带有 'ALL' 或 'COMMITTED READ' 的 SET ENVIRONMENT USELASTCOMMITTED 语句作为此会话环境选项的值。
- 对于使用 Dirty Read 或 Read Uncommitted 隔离级别的所有会话，DBA 可将 USELASTCOMMITTED 配置参数设置为 'ALL' 或设置为'DIRTY READ'。
- 对于使用 Dirty Read 或 Read Uncommitted 隔离级别的个别的会话，任何用户都可发出带有 'ALL' 或 'DIRTY READ' 的 SET ENVIRONMENT USELASTCOMMITTED 语句作为此会话环境选项的值。

仅当行级别锁定生效时，而不是当另一会话持有对整个表的排他锁时，此 LAST COMMITTED 才有用。对于任何 LOCK TABLE 语句在其上应用表级别锁的任何表，禁用此特性。请参阅《GBase 8s SQL 指南：语法》 中的 SET ENVIRONMENT 语句的描述，要获取关于对并发访问通过排他锁锁定其中一些行的表的此特性的更多信息，以及对于可支持此特性的表的种类限制的更多信息，请参阅《GBase 8s 管理员参考手册》中的 USELASTCOMMITTED 配置参数的描述。

页锁

数据库服务器以称为磁盘页的单位存储数据。一个磁盘页包含一行或多行。在一些情况下，最好锁定一个磁盘页，而不是锁定它上面的个别行。例如，对于要求更改大量行的操作，您可能选择页级别锁定，因为行级别锁定（每行一锁）的性价比可能不高。

当您创建表时，如果您未指定 LOCK MODE 子句，则对于该数据库服务器的缺省行为为页级别锁定。使用页锁定，数据库服务器锁定包含该行的整个页。如果您更新存储在同一页上的若干行时，数据库服务器对该页仅使用一个锁。

为所有 CREATE TABLE 语句设置行锁或页锁模式

对于所有新创建的表，GBase 8s 允许您为单个用户（每会话）或为多个用户（每服务器）将锁模式设置为页级别锁定或行级别锁定。您不再需要在每次使用 CREATE TABLE 语句创建新表时指定锁模式。

如果您想要以特定的锁模式创建您的会话内的每个新表，则您必须设置 IFX_DEF_TABLE_LOCKMODE 环境变量。例如，对于要以锁模式行创建的您的会话内的每个新表，请将 IFX_DEF_TABLE_LOCKMODE 设置为 ROW。要覆盖此行为，请使用 CREATE TABLE 或 ALTER TABLE 语句并重新定义 LOCK MODE 子句。

单用户锁模式

如果在您的会话中创建的所有新表都要求相同的锁模式，则设置单用户锁模式。请使用 IFX_DEF_TABLE_LOCKMODE 环境变量来设置单用户锁模式。例如，对于在您的会话内创建的每一个新表都以行级别锁定来创建，请将 IFX_DEF_TABLE_LOCKMODE 设置为 ROW。要覆盖此行为，请使用 CREATE TABLE 或 ALTER TABLE 语句并重新定义 LOCK MODE 子句。要获取关于设置环境变量的更多信息，请参阅《GBase 8s SQL 参考指南》。

多用户锁模式

通过为同一服务器上的所有用户指定锁模式，数据库管理员可使用多用户锁模式来创建更大的并发。然后，任何用户在那台服务器上创建的所有表都会有相同的锁模式。要启用多用户锁模式，请在启动数据库服务器之前设置 IFX_DEF_TABLE_LOCKMODE 环境变量，或设置 DEF_TABLES_LOCKMODE 配置参数。

优先顺序的规则

CREATE TABLE 或 ALTER TABLE 的锁定模式有下列优先顺序的规则，按从最高优先顺序到最低的顺序罗列：

1. 使用 LOCK MODE 子句的 CREATE TABLE 或 ALTER TABLE SQL 语句
2. 单用户环境变量设置
3. 在该服务器环境中的多用户环境变量设置
4. 配置文件中的配置参数
5. 缺省行为（页级别锁定）

稀疏索引锁

当您将索引的锁模式从常规的更改为稀疏的锁模式时，在该索引上要求索引级别的锁，而不是项级别锁或页级别锁，这些是常规的锁。此模式减少索引上的锁调用的数目。

当您知道不会更改索引时，请使用稀疏索引模式；即，当在该索引上执行只读操作时。

请使用常规的锁模式来使数据库服务器在必要时在索引上放置项级别锁或页级别锁。当频繁地更新索引时，请使用此模式。

当数据库服务器执行命令来将锁模式更改为稀疏的时，在命令期间它需要在表上的排他锁。在数据库服务器切换到稀疏锁模式之前，当前正在使用更细颗粒度的锁的任何事务都必须完成。

智能大对象锁

CLOB 或 BLOB 列上的锁与行上的锁是分开的。仅当访问智能大对象时，才锁定它们。当您锁定包含 CLOB 或 BLOB 列的表时，不锁定智能大对象。如果访问是为了写，则以更新模式锁定智能大对象，且当实际的写发生时，将该锁提升为排他的。如果访问是为了读，则以共享的模式锁定智能大对象。数据库服务器识别事务隔离模式，因此，如果设置 Repeatable Read 隔离级别，则在事务结束之前，数据库服务器不释放智能大对象读锁。

当数据库服务器检索行并更新该行指向的智能大对象时，在更新它期间，仅排他地锁定智能大对象。

字节范围锁

您可锁定智能大对象的字节范围。字节范围锁允许事务有选择地仅锁定访问的那些字节，因此，写用户和读用户可同时访问同一智能大对象中不同的字节范围。

要获取关于如何使用字节范围锁的信息，请参阅您的《GBase 8s 性能指南》。

字节范围锁支持死锁检测。要获取关于死锁检测的信息，请参阅 处理死锁。

11.5.3 锁的持续时间

程序控制数据库锁的持续时间。当数据库关闭时，释放数据库锁。

依赖于数据库是否使用事务，表锁的持续时间有所不同。如果数据库不使用事务（即，如果不存在事务日志且您不使用 COMMIT WORK 语句），则保留表锁，直到通过执行 UNLOCK TABLE 语句移除它为止。

表锁、行锁和索引锁的持续时间依赖于您使用的 SQL 语句，并依赖于是否使用事务。

当您使用事务时，在事务结束时释放所有表锁、行锁、页锁和索引锁。当事务结束时，释放所有锁。

11.5.4 在修改时锁定

当数据库服务器通过更新游标访存行时，它在访存的行上放置可提升锁。如果此操作成功，则数据库服务器知道没有其他程序可改变那一行。由于可提升锁不是排他的，其他程序可继续读取该行。由于访存该行的程序可在它发出 UPDATE 或 DELETE 语句之前花费一些时间，或它仅可访存下一行，因此，可提升锁可提升性能。当到了修改行时，数据库服务器获取该行上的排他锁。如果已有可提升锁，则它将那个锁更改为排他的状态。

排他的行锁的持续时间依赖于是否在使用事务。如果未使用事务，则将修改了的行一写到磁盘就释放该锁。当在使用事务时，保留所有这些锁，直到事务结束为止。此操作防止其他程序使用那些可能回滚到它们的原始状态的行。

当在使用事务时，每当删除行时，就使用键锁。使用键锁防止发生下列错误：

- 程序 A 删除一行。
- 程序 B 插入有相同的键的一行。
- 程序 A 回滚它的事务，强制数据库服务器恢复它的删除了的行。

如何处理由程序 B 插入的行？

通过锁定索引，数据库服务器防止第二个程序插入行，直到第一个程序提交它的事务为止。

当前的隔离级别控制在数据库服务器读取不同的行时放置的锁，如下一部分中讨论的那样。

11.6 使用 SELECT 语句来锁定

数据库服务器放置的锁的类型和持续时间依赖于应用程序中的隔离设置，以及该 SELECT 语句是否在更新游标之内。

本部分描述不同的隔离级别和更新游标。

11.6.1 设置隔离级别

隔离级别是您的程序与其他程序的并发操作的隔离程度。数据库服务器提供隔离级别的选择，反映当程序读数据时，程序如何使用锁的不同的规则集。

要设置隔离级别，请使用 SET ISOLATION 或 SET TRANSACTION 语句。SET TRANSACTION 语句还允许您设置访问模式。要获取关于访问模式的更多信息，请参阅 使用访问模式来控制数据修改。

对比 SET TRANSACTION 与 SET ISOLATION

SET TRANSACTION 语句符合 ANSI SQL-92。此语句类似于 GBase 8s SET ISOLATION 语句；然而，SET ISOLATION 语句不符合 ANSI，且不提供访问模式。

下表展示您使用 SET TRANSACTION 与 SET ISOLATION 语句设置的隔离级别之间的关系。

SET TRANSACTION 相关联的	SET ISOLATION
Read Uncommitted	Dirty Read
Read Committed	Committed Read
不支持	Cursor Stability
(ANSI) Repeatable Read Serializable	(GBase 8s) Repeatable Read (GBase 8s) Repeatable Read

SET TRANSACTION 与 SET ISOLATION 语句之间的主要差异是，在事务内隔离级别的行为。对于一个事务，仅可发出 SET TRANSACTION 语句一次。请保证在那个事务期间打开的任何游标都有那个隔离级别（或访问模式，如果您正在定义访问模式的话）。在启动事务之后，您可使用 SET ISOLATION 语句在该事务内多次更改隔离级别。下列示例说明使用 SET ISOLATION 与使用 SET TRANSACTION 之间的差异。

SET ISOLATION

```
EXEC SQL BEGIN WORK;
EXEC SQL SET ISOLATION TO DIRTY READ;
EXEC SQL SELECT ... ;
EXEC SQL SET ISOLATION TO REPEATABLE READ;
EXEC SQL INSERT ... ;
EXEC SQL COMMIT WORK;
    -- Executes without error
```

SET TRANSACTION

```
EXEC SQL BEGIN WORK;
EXEC SQL SET TRANSACTION ISOLATION LEVEL TO SERIALIZABLE;
EXEC SQL SELECT ... ;
EXEC SQL SET TRANSACTION ISOLATION LEVEL TO READ COMMITTED;
Error -876: Cannot issue SET TRANSACTION once a transaction has started.
```

ANSI Read Uncommitted 与 GBase 8s Dirty Read 隔离

最简单的隔离级别 ANSI Read Uncommitted 和 GBase 8s Dirty Read 实际上相当于没有隔离。当程序访存行时，它不放置锁，且不考虑任何东西；它只是从数据库复制行，而不考虑其他程序正在做什么。

程序总是收到完整的数据行。即使在 ANSI Read Uncommitted 或 GBase 8s Dirty Read 隔离之下，程序也从不看到行中是否更新了某些列而有些没更新。然而，使用 ANSI Read Uncommitted 或 GBase 8s Dirty Read 隔离的程序有时会在更新程序结束它的事务之前读取更新了的行。如果更新程序后来回滚它的事务，则读取程序处理从未真正存在的数据（在并发问题列表中的可能性编号 4。）

ANSI Read Uncommitted 或 GBase 8s Dirty Read 是最有效率的隔离级别。 读取程序从不等待，且从不使另一程序等待。 它是任何下列情况下首选的级别：

- 所有表都是静态的；即，并发程序仅读取数据，且从不修改数据。
- 在排他锁中保持表。
- 仅一个程序正在使用该表。

ANSI Read Committed 与 GBase 8s Committed Read 隔离

当程序请求 ANSI Read Committed 或 GBase 8s Committed Read 隔离级别时，数据库服务器保证它从不返回未提交到数据库的行。此操作防止读取未提交的或后来回滚的数据。

ANSI Read Committed 或 GBase 8s Committed Read 实现简单。在它访存行之前，数据库服务器测试来确定更新进程是否在该行上放置了锁；如果没有，则它返回该行。由于已被更新了的（但还未提交的）行在它们上面有锁，因此，此测试确保程序不读取未提交的数据。

ANSI Read Committed 或 GBase 8s Committed Read 实际上不在访存了的行上放置锁，因此，此隔离级别几乎与 ANSI Read Uncommitted 或 GBase 8s Dirty Read 一样有效率。当将每一行数据作为独立的单元处理，未引用同一表或其他表中的其他行时，适于使用此隔离级别。

然而，如果由于并发会话持有行上的共享锁，导致放置该测试锁的尝试不成功，则在 ANSI Read Committed 或 GBase 8s Committed Read 会话中可发生锁定冲突。要避免等待并发进程（通过提交或回滚来）释放共享锁， GBase 8s 支持 Committed Read 隔离级别的 Last Committed 选项。当此 Last Committed 选项生效时，由另一会话的共享锁导致该查询返回该行的最近提交了的版本。

还可通过将 USELASTCOMMITTED 配置参数设置为 'COMMITTED READ' 或设置为 'ALL'，或通过当用户连接到数据库时设置 sysdbopen( ) 过程中的 SET ENVIRONMENT 语句中的 USELASTCOMMITTED 会话环境选项，来激活 Last Committed 特性。要获取关于 ANSI Read Committed 的 Last Committed 选项，或 GBase 8s Committed Read 隔离级别的更多信息，请参阅《GBase 8s SQL 指南：语法》中的 SET ISOLATION 语句的描述。要获取关于 USELASTCOMMITTED 配置参数的信息，请参阅《GBase 8s 管理员参考手册》。

GBase 8s Cursor Stability 隔离

仅可随同 GBase 8s SQL 语句 SET ISOLATION 使用下一级别 Cursor Stability。

当 Cursor Stability 生效时，GBase 8s 在访存的最新的行上放置锁。它为普通的游标放置共享锁，或为更新游标放置可提升锁。一次仅锁定一行；即，每一次访存一行，释放前一行上的锁（除非更新那一行，在此情况下，保持该锁直到事务结束为止。）由于 Cursor Stability 一次仅锁定一行，因此，它对并发的限制低于表锁或数据库锁。

Cursor Stability 确保在程序检测行时，不更改它。当程序更新基于从该行读取的数据的某个其他的表时，这样的行稳定性非常重要。由于 Cursor Stability，程序保证更新是基于当前的信息的。它防止使用陈旧数据。

下列示例说明 Cursor Stability 隔离的有效使用。根据演示数据库，程序 A 想要为制造商 Hero (HRO) 插入新的库存商品。与此同时，程序 B 想要删除制造商 HRO 以及所有与它相关联的库存。可发生下列事件的序列：

1. 在 Cursor Stability 之下操作的程序 A 从 manufact 表访存 HRO 行来获取制造商代码。此操作在该行上放置共享锁。
2. 对于那一行，程序 B 发出 DELETE 语句。由于该锁，数据库服务器让该程序等待。
3. 程序 A 使用它从 manufact 表获得了的制造商代码在 stock 表中插入新行。
4. 程序 A 在 manufact 表上关闭它的游标，或读取不同的行，释放它的锁。
5. 程序 B 从等待中被释放，完成该行的删除，并继续删除那些使用制造商代码 HRO 的 stock 的行，包括程序 A 刚刚插入的行。

如果程序 A 使用较低级别的隔离，则会发生下列序列：

1. 程序 A 读取 manufact 表的 HRO 行来获取制造商代码。未放置锁。
2. 对于那一行，程序 B 发出 DELETE 语句。操作成功。
3. 程序 B 删除使用制造商代码 HRO 的所有 stock 行。
4. 程序 B 结束。
5. 程序 A 不知道该 HRO 行的它的副本现在是无效的，使用制造商代码 HRO 插入新的 stock 行。
6. 程序 A 结束。

最后，在 stock 中出现一行，在 manufact 中没有与之相匹配的制造商代码。而且，程序 B 显然有问题；它未删除本应删除的行。使用 Cursor Stability 隔离级别可防止这些后果。

即使使用 Cursor Stability，重新安排前面的场景也可能失败。只需要让程序 B 以与程序 A 相反的序列对表操作。如果在程序 B 移除 manufact 的行之前，从 stock 删除它，则任何程度的隔离都不可防止出错。每当可能发生这种错误时，所有涉及到的程序都必须使用相同的访问顺序。

ANSI Serializable、ANSI Repeatable Read 和 GBase 8s Repeatable Read 隔离

如果需要 ANSI Serializable 或 ANSI Repeatable Read，则提供单个称为 GBase 8s Repeatable Read 的隔离级别。这在逻辑上等同于 ANSI Serializable。由于 ANSI Serializable 比 ANSI Repeatable Read 更有约束性，因此，当需要 ANSI Repeatable Read 时，可使用 GBase 8s Repeatable Read（虽然 GBase 8s Repeatable Read 在此上下文中比所需的具有更强的约束性）。

Repeatable Read 隔离级别要求数据库服务器在程序检测和访存的每行上都放置锁。对于普通游标放置共享的，对于更新游标放置可提升的。在检测每一行时单独地放置锁。直到该游标关闭或事务结束，才释放它们。

Repeatable Read 允许使用滚动游标的程序多次读取选择了的行，并确保在读取之间不修改或删除它们。（SQL 编程 描述滚动游标。）没有更低的隔离级别保证行依然存在且在第二次读取它们时保持不变。

Repeatable Read 隔离放置最多的锁且保持它们的时间最长。因此，它是降低并发最多的级别。如果您的程序使用此隔离的级别，则请仔细考虑它放置多少锁，保持它们多长时间，以及对其他程序可产生哪些影响。

除了对并发的影响，大量的锁还可是个问题。数据库服务器在锁定表中，按每一程序记录锁的数目。如果超出锁的最大数目，则锁表填满，且数据库服务器不可放置锁。返回一错误代码。管理 GBase 8s 数据库服务器系统的人员可检测该锁定表，并当过度使用它时通知您。

在缺省情况下，在符合 ANSI 的数据库中，将隔离界别设置为 Serializable。需要 Serializable 隔离级别来确保根据 SQL 的 ANSI 标准执行操作。

11.6.2 更新游标

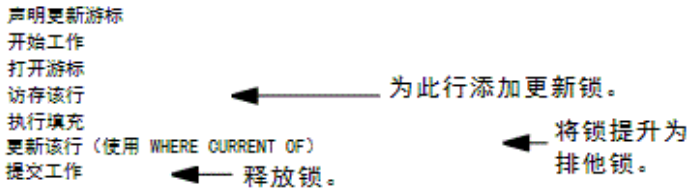
更新游标是一种特殊的游标，当可能潜在地更新行时，应用程序可使用它。要使用更新游标，请在您的应用程序中执行 SELECT FOR UPDATE。更新游标使用可提升锁；即，当应用程序访存行时，数据库服务器放置更新锁（意味着其他用户仍可查看该行），但当应用程序使用更新游标和 UPDATE...WHERE CURRENT OF 更新该行时，将锁更改为排他锁。

使用更新游标的优势在于，您可查看该行，并确信在您查看它时以及您更新它之前，其他用户不可使用更新游标来更改它或查看它。

提示： 在符合 ANSI 的数据库中，由于任何选择游标的行为都与更新游标一样，因此更新游标不是必需的。

下图中的伪代码展现数据库服务器何时使用游标来放置和释放锁。

图: 为更新放置的锁



11.7 保留更新锁

如果用户有低于 Repeatable Read 的隔离级别，则一旦从游标访存下一行，数据库服务器就释放放置在行上的更新锁。当您设置任何下列隔离级别时，使用此特性，您可使用 RETAIN UPDATE LOCKS 子句来保留更新锁，直到事务的结束为止：

- Dirty Read
- Committed Read
- Cursor Stability

此特性允许您避免 Repeatable Read 隔离级别的开销或暂时避开诸如行上的假更新。当打开 RETAIN UPDATE LOCKS 特性，且在 SELECT...FOR UPDATE 语句的访存期间在行上隐式地放置更新锁时，直到事务的结束，才释放更新锁。使用 RETAIN UPDATE LOCKS 特性，仅保持更新锁，直到事务的结束为止，而 Repeatable Read 隔离级别同时保持更新锁和共享锁，直到会话的结束为止。

下列示例展示当您将隔离级别设置为 Committed Read 时，如何使用 RETAIN UPDATE LOCKS 子句。

```
SET ISOLATION TO COMMITTED READ RETAIN UPDATE LOCKS
```

要关闭 RETAIN UPDATE LOCKS 特性，请不要使用 RETAIN UPDATE LOCKS 子句来设置隔离级别。当您关闭该特性时，未直接地释放更新锁。然而，从此时起，后续的访存释放紧前的访存的更新锁，而不是更早的访存操作的更新锁。关闭的游标释放当前行上的更新锁。

要获取关于当您指定隔离级别时，如何使用 RETAIN UPDATE LOCKS 特性的更多信息，请参阅《GBase 8s SQL 指南：语法》。

11.8 使用某些 SQL 语句发生的排他锁

当您执行 INSERT、UPDATE 或 DELETE 语句时，数据库服务器使用排他锁。排他锁意味着任何其他用户都不可更新或删除该项，直到数据库服务器移除该锁为止。此外，任何其他用户都不可查看该行，除非他们正在使用 Dirty Read 隔离级别。

数据库服务器移除排他锁的时刻，依赖于该数据库是否支持事务日志记录。

要获取关于这些排他锁的更多信息，请参阅使用 INSERT、UPDATE 和 DELETE 语句加上的锁定。

11.9 锁类型的行为

GBase 8s 数据库服务器在内部的锁表中存储锁。当数据库服务器读行时，它检查该行或它的相关联的页、表或数据库是否罗列在该锁表中。如果它在锁表中，则数据库服务器还必须检查锁类型。锁表可包含下列类型的锁。

锁名称	描述	通常放置该锁的语句
S	共享锁	SELECT
X	排他锁	INSERT、UPDATE、DELETE
U	更新锁	在更新游标中的 SELECT
B	字节锁	更新 VARCHAR 列的任何语句

此外，锁表可能存储意向锁。意向锁可为意向共享的（IS）、意向排他的（IX）或意向共享排他的（SIX）。意向锁是当需要锁定较低颗粒度对象时，数据库服务器（锁管理器）放置在较高颗粒度对象上的锁。例如，当用户以“共享的”锁定模式锁定行或页时，数据库服务器在该表上放置 IS（意向共享的）锁来提供立即的检查，检查没有其他用户持有该表上的 X 锁。在此情况下，仅在该表上放置意向锁，而不是放置在行或页上。仅可以行、页或表级别放置意向锁。

用户不对意向锁进行直接的控制；锁管理器在内部管理所有的意向锁。

下表展示如果另一用户（或数据库服务器）持有某类型的锁，则用户（或数据库服务器）可放置哪些锁。例如，如果一个用户持有对某项的排他锁，另一用户请求任何种类的锁（排他的、更新或共享的）都会收到错误。此外，如果用户持有对一个项的排他锁，则数据库服务器不能在该项上放置任何意向锁。

	持有 X 锁	持有 U 锁	持有 S 锁	持有 IS 锁	持有 SIX 锁	持有 IX 锁
请求 X 锁	否	否	否	否	否	否
请求 U 锁	否	否	是	是	否	否

请求 S 锁	否	是	是	是	否	否
请求 IS 锁	否	是	是	是	是	是
请求 SIX 锁	否	否	否	是	是	否
请求 IX 锁	否	否	否	是	否	是

要获取关于锁定如何影响性能的信息，请参阅您的《GBase 8s 性能指南》。

11.10 使用访问模式来控制数据修改

GBase 8s 数据库服务器支持访问模式。对于事务内的行，访问模式影响读和写的并发，并使用 SET TRANSACTION 语句设置访问模式。您可使用访问模式来控制共享的文件之中的数据修改。

在缺省情况下，事务是读写的。如果您指定事务为只读，则那个事务不可执行下列任务：

- 插入、删除或更新表行
- 创建、改变或删除任何数据库对象，诸如模式、表、临时表、索引或存储例程。
- 授予或撤销权限
- 更新统计信息
- 重命名列或表

只读访问模式禁止更新。

您可在只读事务中执行存储例程，只要该例程不试图执行任何受限制的操作。

要获取关于如何使用 SET TRANSACTION 语句来指定访问模式的信息，请参阅《GBase 8s SQL 指南：语法》。

11.11 设置锁模式

当您的程序遇到锁定了的数据时，锁模式决定会发生什么情况。当程序尝试访存或修改锁定了的行时，会发生下列情况之一：

- 数据库服务器立即将 SQLCODE 或 SQLSTATE 中的错误代码返回到程序。
- 数据库服务器暂挂程序，直到放置了该锁的程序移除锁为止。
- 数据库服务器暂挂程序一段时间，然后，如果移除该锁，则数据库服务器将错误返回代码发送至该程序。

使用 SET LOCK MODE 语句来在这些结果中选择。

11.11.1 等待锁

当用户遇到锁时，数据库服务器的缺省行为是将错误返回到应用程序。如果您愿意无限期地等待锁（对许多应用程序来说，这是最好的选择），则可执行下列 SQL 语句：

```
SET LOCK MODE TO WAIT
```

当设置此锁模式时，您的程序通常忽略其他并发程序的存在。当您的程序需要访问另一程序已锁定了的行时，它等待，直到移除该锁位置，然后再处理。在大多数情况下，觉察不到该延迟。

您还可等待指定的秒数，如下例所示：

```
SET LOCK MODE TO WAIT 20
```

11.11.2 不等待锁

等待锁的缺点是等待时间可能变长（虽然正确设计的应用程序应短暂地保持它们的锁）。当不可接受长时间延迟的可能性时，程序可执行下列语句：

```
SET LOCK MODE TO NOT WAIT
```

当程序请求锁定了的行时，它立即收到错误代码（例如，错误 -107 Record is locked），并终止当前的 SQL 语句。该程序必须回滚它的当前事务并重试。

当程序启动时，初始的设置为不等待。如果您正在交互地使用 SQL 并看到与锁定相关的错误，则请将锁模式设置为等待。如果您正在编写程序，请考虑使其成为程序首先执行的嵌入式 SQL 语句之一。

11.11.3 等待有限的时间

您可使用下列语句要求数据库服务器设置等待的上限：

```
SET LOCK MODE TO WAIT 17
```

此语句对任何等待的长度放置 17 秒的上限。如果在那个时间内未移除锁，则返回错误代码。

11.11.4 处理死锁

死锁是一对程序阻塞彼此的进度的情况。每一程序对其他程序想要访问的一些对象有锁。仅当所有相关的程序将它们的锁模式都设置为等待锁时，才发生死锁。

当仅涉及单个网络服务器上的数据时，GBase 8s 数据库服务器立即检测到死锁。通过将错误（错误 -143 ISAM error: deadlock detected）返回给要请求锁的第二个程序，它防止发生

死锁。如果程序将它的锁模式设置为不等待锁，则程序收到该错误代码。如果即使程序将锁模式设置为等待之后，它还收到与锁相关的错误代码，则您知道是由于即将发生的死锁。

11.11.5 处理外部的死锁

在不同数据库服务器上的程序之间也可发生死锁。在此情况下，数据库服务器不可立即检测到死锁。（完善的死锁检测需要在网络中的所有数据库服务器之间大量的通信。）而是每一数据库服务器对于程序可等待的时间量设置一个上限，来在不同的数据库服务器上获得对数据的锁。如果到时间了，则数据库服务器假定发生死锁并返回一个与锁相关的错误代码。

也就是说，当涉及外部的数据库时，每个程序都运行最大锁等待时间。DBA 可为数据库服务器设置或修改该最大值。

11.12 简单的并发

如果您不确定如何在锁定与并发之间做出选择，则可使用下列指导方针：如果您的应用程序访问非静态表，且不存在死锁的风险，则当您成程序启动时（就在第一个 CONNECT 或 DATABASE 语句之后），让它执行下列语句：

```
SET LOCK MODE TO WAIT
        SET ISOLATION TO REPEATABLE READ
```

请忽略来自两个语句的返回代码。请执行，就如同没有其他程序存在一样。如果未出现性能问题，则您不需再阅读这部分。

11.13 保持游标

当使用事务日志记录时，GBase 8s 保证在事务结束时，可回滚在事务内所作的一切。要可靠地处理事务，数据库服务器通常应用下列规则：

- 当事务结束时，关闭所有游标。
- 当事务结束时，释放所有锁。

对于支持事务的大多数数据库系统，用于可靠地处理事务的规则都是正常的。然而，存在一些情况，随同游标使用标准事务是不可能的。例如，在没有事务的情况下，下列代码正常工作。然而，当添加事务时，关闭游标与同时使用两个游标发生冲突。

```
EXEC SQL DECLARE master CURSOR FOR
        EXEC SQL DECLARE detail CURSOR FOR   FOR UPDATE
```

```
EXEC SQL OPEN master;
while(SQLCODE == 0)
{
EXEC SQL FETCH master INTO
if(SQLCODE == 0)
{
EXEC SQL BEGIN WORK;
EXEC SQL OPEN detail USING
EXEC SQL FETCH detail
EXEC SQL UPDATE    WHERE CURRENT OF detail
EXEC SQL COMMIT WORK;
}
}
EXEC SQL CLOSE master;
```

在此设计中，使用一个游标来扫描表。选择了的记录用作更新不同的表的基础。问题在于，当将每一更新当做分开的事务处理时（如前一示例中伪代码所示），跟在 UPDATE 之后的 COMMIT WORK 语句关闭所有游标，包括主游标。

最简单的替代方案是将 COMMIT WORK 语句和 BEGIN WORK 语句分别移到最后一个语句和第一个语句， 这样，对整个主表的扫描就是一个大事务。将主表的扫描作为一个大事务来处理，有时是可能的，但如果需要更新许多行，它可变得不现实。锁的数目可能太大，并且在程序期间持有它们。

GBase 8s 数据库服务器支持的解决方案是将关键字 WITH HOLD 添加到主游标的声明。引用这样的游标作为持有游标，在不在事务结束时关闭。数据库服务器仍然关闭所有其他游标，且它仍然释放所有锁，但持有游标保持打开，直到显式地关闭它为止。

在您尝试使用持有游标之前，您必须确保了解此处描述的锁定机制，且您还必须了解正在并发地运行的程序。每当执行 COMMIT WORK 时，释放所有锁，包括放置在任何通过该持有游标访存的行上的任何锁。

对于对表的单向扫描，如果如您所愿地使用游标，则锁的移除无关紧要。然而，您可为任何游标指定 WITH HOLD，包括更新游标和滚动游标。在您这么做之前，您必须了解该事实的含义，即，在事务结束时，释放所有锁（包括对整个表的锁）。

11.14 SQL 语句高速缓存

SQL 语句高速缓存是一个特性，允许您将反复地执行的同一 SQL 语句存储在缓冲区中，以便在不同的用户中可重用这些语句，而不需要每个会话都分配内存。对于包含大量准备

好的语句的应用程序，语句高速缓存可显著地提高性能。然而，当使用语句高速缓存来高速缓存那些一次准备多次执行的语句时，性能的提升就不太显著。

当为数据库服务器启用语句高速缓存时，请使用 SQL 为个别的数据库服务器打开或关闭语句高速缓存。下列语句展示如何使用 SQL 为当前的数据库会话打开高速缓存：

```
SET STATEMENT CACHE ON
```

下列语句展示如何使用 SQL 为当前的数据库会话关闭高速缓存：

```
SET STATEMENT CACHE OFF
```

当禁用高速缓存时，如果您尝试关闭或打开语句高速缓存，则数据库服务器返回错误。

要了解关于 SET STATEMENT CACHE 语句的语法的信息，请参阅《GBase 8s SQL 指南：语法》。要了解关于 STMT_CACHE 和 STMT_CACHE_SIZE 配置参数的信息，请参阅《GBase 8s 管理员参考手册》和您的《GBase 8s 性能指南》。要获取关于 STMT_CACHE 环境变量的信息，请参阅《GBase 8s SQL 参考指南》。

11.15 总结

每当多个程序并发地访问一个数据库（且当其中至少有一个可修改数据时，）所有程序必须允许在它们读数据时，另一程序可更改该数据的可能性。数据库服务器提供锁和隔离级别的机制，其通常允许程序运行，如同它们独占数据一样。

SET STATEMENT CACHE 语句允许您将反复地使用的相同的 SQL 语句存储在缓冲区中。当打开语句高速缓存时，数据库服务器存储相同的语句，因此可在不同的用户会话之中重用它们，而无需为每个会话都分配内存。

12 创建和使用 SPL 例程

本部分描述如何创建和使用 SPL 例程。SPL 例程是以 GBase 8s“存储过程语言”（SPL）编写的用户定义的例程。GBase 8s SPL 是提供流控制的 SQL 的扩展，诸如循环和分支。在数据库上有 Resource 权限的任何人都可创建 SPL 例程。

尽可能地解析和优化以 SQL 编写的例程，然后以可执行的格式存储在系统目录表中。对于 SQL 密集的任务，SQL 例程可能是一个好的选择。SPL 例程可执行以 C 或其他外部语言编写的例程，且外部的例程可执行 SPL 例程。

您可使用 SPL 例程来执行您可以 SQL 执行的任何任务，且可扩展您可单独使用 SQL 完成的任务。由于 SQPL 是数据库的本地语言，且当创建 SPL 例程而不是在运行时时，解析和优化 SPL 例程，对于某些任务，SPL 例程可提升性能。SPL 例程还可减少客户机应用程序与数据库服务器之间的流量并降低程序复杂度。

在 GBase 8s SQL 指南：语法 中描述每一 SPL 语句的语法。每一语句都配有示例。

12.1 SPL 例程介绍

SPL 例程 是包括 SPL 过程 和 SPL 函数 的一个广义术语。SPL 过程是以 SPL 和 SQL 编写的不返回值的例程。SPL 函数是以 SPL 和 SQL 编写的返回单个值、复合数据类型的值或多个值的例程。通常，以 SPL 编写的返回一个值的例程是 SPL 函数。

使用 SQL 和 SPL 语句来编写 SPL 例程。仅可在 CREATE PROCEDURE、CREATE PROCEDURE FROM、CREATE FUNCTION 和 CREATE FUNCTION FROM 语句内使用 SPL 语句。使用诸如 GBase 8s ESQL/C 这样的 SQL API 都可用所有这些语句。使用 DB-Access 可用 CREATE PROCEDURE 和 CREATE FUNCTION 语句。

要在数据库中罗列所有 SPL 例程，请运行此命令，该命令创建和显示数据库的模式：

```
dbschema -d database_name -f all
```

12.1.1 使用 SPL 例程可做什么

使用 SPL 例程，您可实现广泛的目标，包括提升数据库性能，简化应用程序编写，以及限制或监视对数据的访问。

由于以可执行的格式存储 SPL 例程，您可使用它来频繁地执行反复的任务以提升性能。当您执行 SPL 例程而不是直接的 SQL 代码时，您可绕过反复的解析、有效性检查以及查询优化。

您可在数据操纵 SQL 语句中使用 SPL 例程来为那个语句提供值。例如，您可使用例程来执行下列操作：

- 提供要插入到表内的值
- 提供一个值，该值是组成 SELEC、DELETE 或 UPDATE 语句中条件子句的一部分

这些操作是在数据操作语句中例程的两种可能的使用，但也存在其他的。实际上，数据操纵 SQL 语句中的任何表达式都可由例程调用构成。

您还可在 SPL 例程中发出 SQL 语句来对数据库用户隐藏那些 SQL 语句。不是让所有用户都了解如何使用 SQL，一位有经验的 SQL 用户可编写 SPL 例程来封装 SQL 活动，并让其他人了解在该数据库中存储着该例程，以便他们可以执行它。

您可编写 SPL 例程，由不具有 DBA 权限的用户使用 DBA 权限来运行它。此特征允许您限制和控制对数据库中数据的访问。另外，SPL 例程可监视访问某些表或数据的用户。

12.2 SPL 例程格式

SPL 例程由开始语句、语句块和结束语句组成。在语句块内，您可使用 SQL 或 SPL 语句。

12.2.1 CREATE PROCEDURE 或 CREATE FUNCTION 语句

您必须首先决定您正在创建的例程是否返回值。如果例程不返回值，则使用 CREATE PROCEDURE 语句来创建一个 SPL 过程。如果例程返回一个值，则使用 CREATE FUNCTION 语句来创建一个 SPL 函数。

要创建 SPL 例程，请使用一个 CREATE PROCEDURE 或 CREATE FUNCTION 语句来编写该例程体，并注册它。

开始和结束例程

要创建不返回值的 SPL 例程，请使用 CREATE PROCEDURE 语句开始，并以 END PROCEDURE 关键字结束。下图展示如何开始和结束 SPL 过程。

图: 开始和结束 SPL 例程。

```
CREATE PROCEDURE new_price( per_cent REAL )
        . . .
        END PROCEDURE;
```

要获取关于命名约定的更多信息，请参阅《GBase 8s SQL 指南：语法》中的“标识符”段。

要创建返回一个或多个值的 SPL 函数，请使用 CREATE FUNCTION 语句开始，并以 END FUNCTION 关键字结束。 下图展示如何开始和结束 SPL 函数。

图: 开始和结束 SPL 函数。

```
CREATE FUNCTION discount_price( per_cent REAL)
          RETURNING MONEY;
          . . .
          END FUNCTION;
```

在 SPL 例程中，END PROCEDURE 或 END FUNCTION 关键字是必需的。

重要： 为了与较早的 GBase 8s 产品相兼容，您可使用带有 RETURNING 子句的 CREATE PROCEDURE 来创建返回值的用户定义的例程。然而，如果您对于不返回值的 SPL 例程（SPL 过程）使用 CREATE PROCEDURE，而对于返回一个或多个值的 SPL 例程（SPL 函数）使用 CREATE FUNCTION，则您的代码会更易于阅读和维护，

指定例程名称

紧跟在 CREATE PROCEDURE 或 CREATE FUNCTION 语句之后，且在参数列表之前为 SPL 例程指定名称，如图所示。

图: 为 SPL 例程指定名称。

```
CREATE PROCEDURE add_price (arg INT )
```

GBase 8s 允许您以相同的名称但以不同的参数创建多个 SPL 例程。此特性称为例程重载。例如，您可能在您的数据库中创建下列每一 SPL 例程：

```
CREATE PROCEDURE multiply (a INT, b FLOAT)
        CREATE PROCEDURE multiply (a INT, b SMALLINT)
        CREATE PROCEDURE multiply (a REAL, b REAL)
```

如果您以名称 multiply() 调用例程，则数据库服务器评估该例程的名称和它的参数来确定执行哪个例程。

例程解析是数据库服务器在其中搜索它可使用的例程签名，给定例程的名称和参数列表的过程。每个例程都有一个基于下列信息唯一地标识该例程的签名：

- 例程的类型（过程或函数）
- 例程名称
- 参数的数目
- 参数的数据类型
- 参数的顺序

如果您输入该例程的完整参数列表，则在 CREATE、DROP 或 EXECUTE 语句中使用该例程签名。例如，下图中的每一语句都使用例程签名。

图: 例程签名。

```
CREATE FUNCTION multiply(a INT, b INT);

        DROP PROCEDURE end_of_list(n SET, row_id INT);

        EXECUTE FUNCTION compare_point(m point, n point);
```

添加特定的名称

由于 GBase 8s 支持例程重载，因此，不可能单独通过 SPL 例程的名称来唯一地标识它。然而，可通过特定的名称来唯一地标识例程。除了例程名称之外，特定的名称是您在 CREATE PROCEDURE 或 CREATE FUNCTION 语句中定义的唯一的标识符。使用 SPECIFIC 关键字来定义特定的名称，且该名称在数据库中是唯一的。在同一数据库中，不可有两个相同的特定的名称，即使它们有不同的所有者也不行。

特定的名称最长可达 128 字节。下图展示如何在创建 calculate() 函数的 CREATE FUNCTION 语句中定义特定的名称 calc1。

图: 定义特定的名称。

```
CREATE FUNCTION calculate(a INT, b INT, c INT)
        RETURNING INT
        SPECIFIC calc1;
        . . .
        END FUNCTION;
```

由于所有者 bsmith 已给定了 SPL 函数特定的名称 calc1，因此任何其他用户都不可使用特定的名称 calc1 来定义例程——SPL 或外部的。现在，您可引用该例程为 bsmith.calculate，或在任何需要 SPECIFIC 关键字的语句中使用 SPECIFIC 关键字 calc1。

添加参数列表

当您创建 SPL 例程时，您可定义参数列表，以便当调用例程时，它接受一个或多个参数。参数列表是可选的。

SPL 例程的参数必须有名称，且可使用缺省值来定义。下列是参数可指定的数据类型的种类：

- 内建的数据类型
- Opaque 数据类型
- Distinct 数据类型
- Row 类型
- 集合类型
- 智能大对象（CLOB 和 BLOB）

参数列表不可直接地指定任一下列数据类型：

- SERIAL
- SERIAL8
- BIGSERIAL
- TEXT
- BYTE

然而，对于序列数据类型，例程可返回在数值上对等的值，将它们强制转型为对应的整数类型（INT、INT8 或 BIGINT）。类似地，对于支持简单大对象数据类型的例程，参数列表可包括 REFERENCES 关键字来返回指向 TEXT 或 BYTE 对象的存储位置的描述符。

下图展示参数列表的示例。

图: 不同的参数列表的示例。

```
CREATE PROCEDURE raise_price(per_cent INT);

                CREATE FUNCTION    raise_price(per_cent INT DEFAULT 5);


                CREATE PROCEDURE update_emp(n employee_t);
                CREATE FUNCTION    update_nums( list1 LIST(ROW (a
VARCHAR(10),
                b VARCHAR(10),
                c INT) NOT NULL ));
```

当您定义参数时，同时完成两个任务：

- 当执行例程时，您请求用户提供值。
- 您隐式地定义您可在例程体中用作本地变量的变量（带有与参数名称相同的名称）。

如果您以缺省值定义参数，则用户可使用或不用对应的参数来执行该 SPL 例程。如果用户执行不带参数的 SPL 例程，则数据库服务器指定参数的缺省值作为参数。

当您调用 SPL 例程时，您可对参数给定 NULL 值。在缺省情况下，SPL 例程处理 NULL 值。然而，如果参数为集合元素，则您不可对该参数给定 NULL 值。

简单大对象作为参数

虽然您不可使用简单大对象（包含 TEXT 或 BYTE 数据类型的大对象）来定义参数，但您可使用 REFERENCES 关键词来定义指向简单大对象的参数，如下图所示。

图: REFERENCES 关键字的使用。

```
CREATE PROCEDURE proc1(lo_text REFERENCES TEXT)

                CREATE FUNCTION proc2(lo_byte REFERENCES BYTE DEFAULT
NULL)
```

REFERENCES 关键字意味着将包含指向简单大对象的指针的描述符传给 SPL 例程，而不是传对象本身。

未定义的参数

当您调用 SPL 例程时，您可指定全部或部分已定义的参数，或不指定参数。如果您未指定参数，且如果它的对应的参数没有缺省值，则给定在 SPL 例程内作为变量使用的该参数的状态为未定义的。

未定义是为没有值的 SPL 变量使用的特殊的状态。只要您在该例程体中不尝试使用状态为未定义的变量，该 SPL 例程就不会出错。

未定义的状态与 NULL 值不一样。（NULL 值意味着该值为未知的，或不存在，或不适用。）

添加返回子句

如果您使用 CREATE FUNCTION 来创建 SPL 例程，您必须指定返回一个或多个值的返回子句。

提示： 如果您使用 CREATE PROCEDURE 语句来创建 SPL 例程，则您有指定返回子句的选项。然而，如果您使用 CREATE FUNCTION 语句来创建返回值的例程，则您的代码会比较易读且易于维护。

要指定返回子句，请使用带有该例程将返回的数据类型的列表的 RETURNING 或 RETURNS 关键字。数据类型可为除了 SERIAL、SERIAL8、TEXT 或 BYTE 之外的任何 SQL 数据类型。

下图中的返回子句指定，该 SPL 例程将返回 INT 值和 REAL 值。

图: 指定返回子句。

```
FUNCTION find_group(id INT)
        RETURNING INT, REAL;
        . . .
        END FUNCTION;
```

在您指定返回子句之后，您还必须在例程体中指定 RETURN 语句，显式地返回调用例程的值。要获取关于编写 RETURN 语句的更多信息，请参阅从 SPL 函数返回值。

要指定应返回简单大对象（TEXT 或 BYTE 值）的函数，您必须使用 REFERENCES 子句，如下图所示，这是因为 SPL 例程仅返回指向该对象的指针，而不是该对象本身。

图: 使用 REFERENCES 子句。

```
CREATE FUNCTION find_obj(id INT)
        RETURNING REFERENCES BYTE;
```

添加显示标签

您可使用 CREATE FUNCTION 来创建例程，其为返回的值指定显示标签的名称。如果您未为显示标签指定名称，则该标签会显示为 expression。

此外，虽然对于返回值的例程推荐使用 CREATE FUNCTION，但您可使用 CREATE PROCEDURE 来创建返回值的例程，并指定返回的值的显示标签。

如果您选择为一个返回值指定显示标签，则您必须为每个返回值指定显示标签。此外，每一返回值必须有唯一的显示标签。

要添加显示标签，您必须指定返回子句，请使用 RETURNING 关键字。下图中的返回子句指定该例程将返回一个带有 serial_num 显示标签的 INT 值，一个带有 name 显示标签的

CHARE 值，以及一个带有 points 显示标签的 INT 值。您可使用下图中的 CREATE FUNCTION 或 CREATE PROCEDURE。

图: 指定返回子句。

```
CREATE FUNCTION p(inval INT DEFAULT 0)
        RETURNING INT AS serial_num, CHAR (10) AS name, INT AS points;
        RETURN (inval + 1002), "Newton", 100;
        END FUNCTION;
```

在下图中展示返回的值和它们的显示标签。

图: 返回的值和它们的显示标签。

```
serial_num      name        points

           1002             Newton      100
```

提示：由于您可在 SELECT 语句中直接地为返回值指定显示标签，因此，当在 SELECT 语句中使用 SPL 例程时，该标签会显示为 expression。要获取关于在 SELECT 语句中为返回值指定显示标签的更多内容，请参阅 编写 SELECT 语句。

指定 SPL 函数是否为变体

当您创建 SPL 时，在缺省情况下，该函数为变体。当使用相同的参数调用函数时，如果它返回不同的结果，或如果它修改数据库或变量状态，则该函数为变体。例如，返回当前的日期或时间的函数是变体函数。

虽然在缺省情况下，SPL 函数为变体，但如果当您创建函数时指定 WITH NOT VARIANT，则该函数不可包含任何 SQL 语句。您仅可在非变体函数上定义函数索引。

添加修饰符

当您编写 SPL 函数时，您可使用 WITH 子句来将修饰符添加到 CREATE FUNCTION 语句。在 WITH 子句中，您可指定 COMMUTATOR 或 NEGATOR 函数。其他修饰符是用于外部例程的。

限制： 您仅可以 SPL 函数使用 COMMUTATOR 或 NEGATOR 修饰符。您不可以 SPL 过程使用任何修饰符。

COMMUTATOR 修饰符

COMMUTATOR 修饰符允许您指定 SPL 函数为您正在创建的 SPL 函数的转换函数。转换函数接受相同的参数作为您正在创建的 SPL 函数，但以相反的顺序，并返回相同值。对于 SQL 优化器的执行，转换函数的成本效益比更高。

例如，如果 a 小于 b，则函数 lessthan(a,b) 返回 TRUE， 而如果 b 大于或等于 a，则 greaterthan(b,a) 返回 TRUE， 二者是转换函数。 下图使用 WITH 子句来定义转换函数。

图: 定义转换函数。

```
CREATE FUNCTION lessthan( a dtype1, b dtype2 )
        RETURNING BOOLEAN
        WITH ( COMMUTATOR = greaterthan );
        . . .
        END FUNCTION;
```

如果 greaterthan(b,a) 的执行成本低于 lessthan(a,b)，则优化器可能使用 greaterthan(b,a)。要指定转换函数，您必须同时拥有该转换函数和您正在编写的 SPL 函数。您还必须将两个函数的 Execute 权限授予您的 SPL 函数的用户。

要了解授予权限的详细描述，请参阅《GBase 8s SQL 指南：语法》 中的 GRANT 语句的描述。

NEGATOR 修饰符

NEGATOR 修饰符是用于布尔函数的变量。如果两个布尔函数的参数相同，顺序相同，且返回互补的布尔值，则它们是否定函数。

例如，如果 a 等于 b，则函数 equal(a,b) 返回 TRUE，而如果 a 等于 b，则 notequal(a,b) 返回 FALSE，二者是否定函数。如果您指定的否定函数的执行成本低于原始的函数，则优化器可能选择执行该否定函数。

下图展示如何使用 CREATE FUNCTION 语句的 WITH 子句来指定否定函数。

图: 指定否定函数。

```
CREATE FUNCTION equal( a dtype1, b dtype2 )
        RETURNING BOOLEAN
        WITH ( NEGATOR = notequal );
        . . .
        END FUNCTION;
```

提示：在缺省情况下，任何 SPL 例程都可处理在参数列表中传给其的 NULL 值。换言之，对于 SPL 例程，将 HANDLESNULLS 修饰符设置为 YES，且您不可更改它的值。

要获取关于 COMMUTATOR 和 NEGATOR 修饰符的更多信息，请参阅《GBase 8s SQL 指南：语法》 中的 Routine Modifier 段。

指定 DOCUMENT 子句

DOCUMENT 和 WITH LISTING IN 子句跟在 END PROCEDURE 或 END FUNCTION 语句之后。

DOCUMENT 子句允许您将注释添加到您的 SPL 例程，另一例程可从系统目录表选择它，如果需要的话。下图中的 DOCUMENT 子句包含展示用户如何运行该 SPL 例程的用法语句。

图: 展示用户如何运行 SPL 例程的用法语句。

```
CREATE FUNCTION raise_prices(per_cent INT)
        . . .
        END FUNCTION
        DOCUMENT "USAGE: EXECUTE FUNCTION raise_prices (xxx)",
        "xxx = percentage from 1 - 100";
```

请记住在文字的子句两头放置单引号或双引号。如果文字的子句跨过一行，则在每一行的两头放置引号。

指定清单文件

WITH LISTING IN 选项允许您将任何可能发生的编译时警告定向到文件。

当您在 UNIX™ 上工作时，下图展示如何将编译时警告记录在 /tmp/warn_file 中。

图: 在 UNIX 上记录编译时警告。

```
CREATE FUNCTION raise_prices(per_cent INT)
        . . .
        END FUNCTION
        WITH LISTING IN '/tmp/warn_file'
```

当您在 Windows™ 上工作时，下图展示如何将编译时警告记录在 \tmp\listfile 中。

图: 在 Windows 上记录编译时警告。

```
CREATE FUNCTION raise_prices(per_cent INT)
        . . .
        END FUNCTION
        WITH LISTING IN 'C:\tmp\listfile'
```

请始终记住在文件名称或路径名称两头放置单引号或双引号。

添加注释

您可将注释添加到 SPL 例程的任何行，即使是空行也行。

要添加注释，请使用任一下列注释说明类型：

- 在注释的左边放置双连字符（--）。
- 将注释文本括在一对大括号之间（{ . . .}）。
- 在 C 类型的“斜杠和星号”注释指示符之间定界注释（/* . . .*/）。

要添加多行注释，请执行下列操作之一：

- 在每一行注释前放置双连字符
- 将全部注释括在一对大括号之内。
- 在注释的第一行的左边放置 /*，并在注释的最后一行的末尾放置 */。

以大括号作为注释指示符是 GBase 8s 对 SQL 语言的 ANSI/ISO 标准的扩展。在 SPL 例程中，全部三种注释类型也是有效的。

如果您使用大括号或 C 类型注释指示符来定界注释的文本，则开头的指示符必须与结尾的指示符是同一类型。

下图中的所有实例都是有效的注释。

图: 有效的注释实例。

```
SELECT * FROM customer -- Selects all columns and rows

        SELECT * FROM customer
        -- Selects all columns and rows
        -- from the customer table

        SELECT * FROM customer
        { Selects all columns and rows
        from the customer table }

        SELECT * FROM customer
        /* 从 customer 表选择所有的列和行 */
```

重要： 大括号（{ }）可用于定界注释，也用于定界集合中元素的列表。要确保解析器正确地识别注释的结束或集合中元素列表的结束，请在处理集合数据类型的 SPL 例程中为注释使用连字符（--）。

12.2.2 完整例程的示例

下列 CREATE FUNCTION 语句创建读取客户地址的例程：

```
CREATE FUNCTION read_address          (lastname CHAR(15)) -- one argument
        RETURNING CHAR(15), CHAR(15), CHAR(20), CHAR(15),CHAR(2)
        CHAR(5); -- 6 items

        DEFINE p_lname,p_fname, p_city CHAR(15);
        --define each routine variable
        DEFINE p_add CHAR(20);
```

```
        DEFINE p_state CHAR(2);
        DEFINE p_zip CHAR(5);

        SELECT fname, address1, city, state, zipcode
        INTO p_fname, p_add, p_city, p_state, p_zip
        FROM customer
        WHERE lname = lastname;

        RETURN p_fname, lastname, p_add, p_city, p_state, p_zip;
        --6 items
        END FUNCTION;

        DOCUMENT 'This routine takes the last name of a customer as',
        --brief description
        'its only argument. It returns the full name and address',
        'of the customer.'

        WITH LISTING IN 'pathname' -- modify this pathname according
        -- to the conventions that your operating system requires

        -- compile-time warnings go here
        ; -- end of the routine read_address
```

12.2.3 在程序中创建 SPL 例程

要使用 SQL API 来创建 SPL 例程，请将 CREATE PROCEDURE 或 CREATE FUNCTION 语句的文本放在文件中。请使用 CREATE PROCEDURE FROM 或 CREATE FUNCTION FROM 语句并引用那个文件来编译该例程。例如，要创建读取客户姓名的例程，您可使用诸如在前面的示例中的一个语句，并将它存储在文件中。如果将该文件命名为 read_add_source，则下列语句编译 read_address 例程：

```
CREATE PROCEDURE FROM 'read_add_source';
```

下列示例展示在 GBase 8s ESQL/C 程序中，前面的 SQL 语句是怎样的：

```
/* This program creates whatever routine is in *
        * the file 'read_add_source'.
        */
        #include <stdio.h>
        EXEC SQL include sqlca;
```

```
EXEC SQL include sqlda;
EXEC SQL include datetime;
/* Program to create a routine from the pwd */

main()
{
EXEC SQL database play;
EXEC SQL create procedure from 'read_add_source';
}
```

12.2.4 在本地的或远程的数据库中删除例程

在您创建 SPL 例程之后，您不可更改该例程体。相反，您需要删除该例程并重新创建它。然而，在您删除例程之前，请确保您在数据库之外的某个地方有它的文本的副本。

通常，请使用带有 SPL 过程名称的 DROP PROCEDURE 和带有 SPL 函数名称的 DROP FUNCTION，如下图所示。

图: DROP PROCEDURE 和 DROP FUNCTION。

```
DROP PROCEDURE raise_prices;
        DROP FUNCTION read_address;
```

提示： 您还可使用带有函数名称的 DROP PROCEDURE 来删除 SPL 函数。然而，推荐您使用仅带有过程名称的 DROP PROCEDURE，以及仅带有函数名称的 DROP FUNCTION。

如果数据库有同名称的其他例程（重载的例程），则您不可只通过它的例程名称来删除 SPL 例程。要删除已重载了的例程，您必须指定它的签名或它的特定的名称。下图展示您可能删除重载了的例程的两种方式。

图: 删除重载了的例程。

```
DROP FUNCTION calculate( a INT, b INT, c INT);
        -- this is a signature

        DROP SPECIFIC FUNCTION calc1;
        -- this is a specific name
```

如果您不知道例程的类型（是函数还是过程），则您可使用 DROP ROUTINE 语句来删除它。DROP ROUTINE 对函数或过程都有效。DROP ROUTINE 还有 SPECIFIC 关键字，如下图所示。

图: DROP ROUTINE 语句。

```
DROP ROUTINE calculate;
        DROP SPECIFIC ROUTINE calc1;
```

在您删除存储在远程数据库服务器上的 SPL 例程之前，请注意下列限制。仅当只用例程名称而不需它的参数，就足以标识该例程时，您才可以 database@dbservername:owner.routinename 的形式使用完全限定的例程来删除 SPL 例程。

分布式操作中对数据类型的限制

如果 SPL 例程访问非本地的数据库服务器中的表，或调用 SPL 例程作为另一数据库服务器的数据库的 UDR，则该例程仅可有非 opaque 内建的数据类型作为它们的参数或返回的值。

然而，如果表或 UDR 驻留在同一 GBase 8s 示例的另一数据库上，则使用 SPL（或 GBase 8s 支持的外部语言）编写的例程的参数和返回的值可为内建的 opaque 数据类型 BLOB、BOOLEAN、CLOB 和 LVARCHAR。如果下列条件为真，则它们还可为 UDT 或 DISTINCT 数据类型：

- 远程数据库与当前的数据库有相同的服务器。
- 将 UDT 参数显式地强制转型为内建的数据类型。
- DISTINCT 类型是基于内建的类型的，且被显式地强制转型为内建的类型。
- 在所有参与的数据库中定义 SPL 例程和所有强制转型。

12.3 定义和使用变量

您必须在例程体中定义在 SPL 例程中使用的任何变量，而不是在例程的参数列表中隐式地定义的变量。

在内存中保持变量的值；该变量不是数据库对象。因此，回滚事务不恢复 SPL 变量的值。

要在 SPL 例程中定义变量，请使用 DEFINE 语句。DEFINE 不是可执行语句。DEFINE 必须出现在 CREATE PROCEDURE 语句之后且任何其他语句之前。下图中的示例是所有合法的变量定义。

图: 变量定义。

```
DEFINE a INT;
        DEFINE person person_t;
        DEFINE GLOBAL gl_out INT DEFAULT 13;
```

要获取关于 DEFINE 的更多信息，请参阅《GBase 8s SQL 指南：语法》 中的描述。

SPL 变量有名称和数据类型。变量名称必须是有效的标识符，如 GBase 8s SQL 指南：语法 中“标识符”段中描述的那样。

12.3.1 声明本地变量

您可定义变量为作用域中的本地的或全局的。本部分描述本地变量。在 SPL 例程中，本地变量：

- 仅对于该 SPL 例程的持续时间是有效的
- 每一次执行例程时，重置为它们的初始值或为用户传给该例程的值
- 不可有缺省值

您可在任一下列数据类型上定义本地变量：

- 内建的数据类型（除了 SERIAL、SERIAL8、BIGSERIAL、TEXT 或 BYTE 之外）
- 在执行该 SPL 例程之前，在数据库中定义的任何扩展的数据类型（row 类型、opaque、distinct 或集合类型）

本地变量的作用域时在其中声明它的那个语句块。您可以不同的定义在该语句块之外使用相同的变量名称。

要获取关于定义全局变量的更多信息，请参阅 声明全局变量。

本地变量的作用域

在定义本地变量的那个语句块内以及任何嵌套的语句块内，它是有效的，除非您在该语句块中重新定义该变量。

在系统中的 SPL 过程的开头，定义并初始化整数变量 x、y 和 z。

图: 定义和初始化变量。

```
CREATE PROCEDURE scope()
                DEFINE x,y,z INT;
                LET x = 5;
                LET y = 10;
                LET z = x + y; --z is 15
                BEGIN
                DEFINE x, q INT;
                DEFINE z CHAR(5);
                LET x = 100;
                LET q = x + y;     -- q = 110
                LET z = 'silly'; -- z receives a character value
                END
                LET y = x; -- y is now 5
                LET x = z; -- z is now 15, not 'silly'
                END PROCEDURE;
```

BEGIN 与 END 语句标记在其中定义整数变量 x 和 q 以及 CHAR 变量 z 的嵌套的语句块。在嵌套的块内，重新定义的变量 x 掩盖原始的变量 x。在标记该嵌套的块结束的 END 语句之后，可再次访问 x 的原始值。

声明内建的数据类型的变量

声明为内建的 SQL 数据类型的变量可持有从那个内建的类型的列检索的值。您可将 SPL 变量声明为大部分内建的类型，除了 BIGSERIAL、SERIAL 和 SERIAL8 之外，如下图所示。

图: 内建的类型变量。

```
DEFINE x INT;
        DEFINE y INT8;
        DEFINE name CHAR(15);
        DEFINE this_day DATETIME YEAR TO DAY;
```

您可声明适当的整数数据类型（诸如 BIGINT、INT 或 INT8）的 SPL 变量，来存储序列列或序列对象的值。

声明智能大对象的变量

BLOB 或 CLOB 对象（或包含智能大对象的数据类型）的变量不包含该对象本身，而是指向该对象的指针。下图展示如何为 BLOB 和 CLOB 对象定义变量。

图: BLOB 或 CLOB 对象的变量。

```
DEFINE a_blob BLOB;
        DEFINE b_clob CLOB;
```

声明简单大对象的变量

简单大对象（TEXT 或 BYTE 对象）的变量不包含该对象本身，而是执行该对象的指针。当您对 TEXT 或 BYTE 数据类型定义变量时，您必须在数据类型之前使用关键字 REFERENCES，如下图所示。

图: 在数据类型之前使用 REFERENCES 关键字。

```
DEFINE t REFERENCES TEXT;
        DEFINE b REFERENCES BYTE;
```

声明集合变量

为了保持从数据库访存的集合，变量必须为类型 SET、MULTISET 或 LIST。

重要： 必须将集合变量定义为本地变量。您不可将集合变量定义为全局变量。

SET、MULTISET 或 LIST 类型的变量是保存在 DEFINE 语句中命名的类型的集合的集合变量。下图展示如何定义 typed 集合变量。

图: 定义 typed 集合变量。

```
DEFINE a SET ( INT NOT NULL );

        DEFINE b MULTISET ( ROW (   b1 INT,
        b2 CHAR(50),
        )   NOT NULL );

        DEFINE c LIST ( SET (DECIMAL NOT NULL) NOT NULL);
```

您必须始终将集合变量的元素定义为 NOT NULL。在此示例中，定义变量 a 来保存非 NULL 整数的 SET；变量 b 保存非 NULL row 类型的 MULTISET；变量 c 保存非 NULL 十进制值的非 NULL 集合的 LIST。

在变量定义中，您可在任何组合或深度中嵌套复合的类型，来与存储在您的数据库中的数据类型相匹配。

您不可将一种类型的集合变量分配给另一类型的集合变量。例如，如果您将集合变量定义为 SET，则您不可将另一 MULTISET 或 LIST 类型的集合变量分配给它。

声明 row 类型变量

Row 类型变量保存命名的和未命名的 row 类型的数据。您可定义命名的 row 变量或未命名的 row 变量。假设您定义如下图所示的命名的 row 类型。

图: 命名的和未命名的 row 变量。

```
CREATE ROW TYPE zip_t
        (
        z_code          CHAR(5),
        z_suffix        CHAR(4)
        );

        CREATE ROW TYPE address_t
        (
        street          VARCHAR(20),
        city            VARCHAR(20),
        state           CHAR(2),
        zip             zip_t
```

```
);

CREATE ROW TYPE employee_t
(
name            VARCHAR(30),
address         address_t
salary          INTEGER
);

CREATE TABLE employee OF TYPE employee_t;
```

如果您以命名的 row 类型的名称定义变量，则该变量仅可保存那种 row 类型的数据。在下图中，person 变量仅可保存 employee_t 类型的数据。

图: 定义 person 变量。

```
DEFINE person employee_t;
```

要定义保存在未命名的 row 类型中的数据的变量，请跟在 row 类型的字段之后使用 ROW 关键字，如下图所示。

图: 使用后跟 row 类型的字段的 ROW 关键字。

```
DEFINE manager ROW (name            VARCHAR(30),
                    department    VARCHAR(30),
                    salary          INTEGER );
```

由于仅对未命名的 row 类型的结构等价进行类型检查，因此，以未命名的 row 类型定义的变量可保存任何未命名的 row 类型的数据，其有相同的字段数和相同的类型定义。因此，变量 manager 可保存下图中任何 row 类型的数据。

图: 未命名的 row 类型。

```
ROW ( name              VARCHAR(30),
                    department     VARCHAR(30),
                    salary           INTEGER );

                    ROW ( french          VARCHAR(30),
                    spanish         VARCHAR(30),
                    number          INTEGER );

                    ROW ( title           VARCHAR(30),
                    musician        VARCHAR(30),
                    price           INTEGER );
```

重要：在您使用 row 类型变量之前，您必须使用 LET 语句或 SELECTINTO 语句来初始化该 row 变量。

声明 opaque 类型和 distinct 类型变量

Opaque 类型变量保存从 opaque 数据类型检索的数据。Distinct 类型变量保存从 distinct 数据类型检索的数据。如果您以 opaque 数据类型或 distinct 数据类型定义变量，则该变量仅可保存那种类型的数据。

如果您定义名为 point 的 opaque 数据类型，和名为 centerpoint 的 distinct 数据类型，则您可定义 SPL 变量来保存这两类数据，如下图所示。

图: 定义 SPL 变量来保存 opaque 和 distinct 数据类型。

```
DEFINE a point;
        DEFINE b centerpoint;
```

变量 a 仅可保存类型 point 的数据，b 仅可保存类型 centerpoint 的数据。

使用 LIKE 子句来声明列数据的变量

如果您使用 LIKE 子句，则数据库服务器定义有相同数据类型的变量作为表或视图中的列。

如果该列包含集合、row 类型或嵌套的复合类型，则该变量具有在该列中定义的复合的或嵌套的复合类型。

在下图中，变量 loc1 定义 image 表中 locations 列的数据类型。

图: 为 image 表中的 locations 列定义 loc1 数据类型。

```
DEFINE loc1 LIKE image.locations;
```

声明 PROCEDURE 类型变量

在 SPL 例程中，您可定义类型 PROCEDURE 的变量，并将现有的 SPL 例程或外部例程的名称分配给该变量。定义 PROCEDURE 类型的变量指示该变量是对用户定义的例程的调用，而不是对同一名称的内建例程的调用。

例如，下图中的语句定义 length 为一个 SPL 过程或 SPL 函数，不作为内建的 LENGTH 函数。

图: 定义 length 作为 SPL 过程。

```
DEFINE length PROCEDURE;
        LET x = length( a,b,c );
```

此定义在该语句块的作用域内禁用内建的 LENGTH 函数。如果您已以名称 LENGTH 创建了 SPL 或外部例程，则您可使用这样的定义。

由于 GBase 8s 支持例程重载，因此，您可以相同的名称定义多个 SPL 例程或外部例程。如果您从 SPL 例程调用任何例程，则 GBase 8s 基于指定的参数和例程确定规则，确定使用哪个例程。要获取关于例程重载和例程确定的信息，请参阅《GBase 8s 用户定义的例程和数据类型开发者指南》。

提示：如果您以相同的名称创建 SPL 例程作为聚集函数（SUM、MAX、MIN、AVG、COUNT）或使用名称 extend，则您必须以所有者名称来限定该例程。

带有变量的下标

您可随同 CHAR、VARCHAR、NCHAR、NVARCHAR、BYTE 或 TEXT 数据类型的变量使用下标。下标指示您想要在变量内使用的起始的和终止的字符位置。

下标必须始终为常量。您不可使用变量作为下标。下图展示如何随同 CHAR(15) 变量使用下标。

图: 带有 CHAR(15) 变量的下标。

```
DEFINE name CHAR(15);
        LET name[4,7] = 'Ream';
        SELECT fname[1,3] INTO name[1,3] FROM customer
        WHERE lname = 'Ream';
```

在此示例中，将客户的姓置于 name 的位置 4 与 7 之间。将客户的名的前三个字符检索到 name 的位置 1 至 3 内。由两个下标定界的变量的该部分称为子字符串。

变量与关键字歧义

如果您声明的变量的名字是 SQL 关键字，则可发生歧义。下列标识符的规则帮助您避免 SPL 变量、SPL 例程名称和内建的函数名称的歧义：

- 定义了的变量优先级最高。
- 以 DEFINE 语句中的 PROCEDURE 关键字定义的例程优先于 SQL 函数。
- SQL 函数优先于那些存在但未以 DEFINE 语句中的 PROCEDURE 关键字标识的 SPL 例程。

通常，请避免为变量的名称使用 ANSI 保留字。例如，您不可以名称 count 或 max 定义变量，因为它们是聚集函数的名称。要了解您应避免用作变量名称的保留的关键字列表，请参阅《GBase 8s SQL 指南：语法》中的“标识符”段。

要获取关于 SPL 例程名称与 SQL 函数名称之间的歧义的信息，请参阅《GBase 8s SQL 指南：语法》。

变量和列名称

如果您为 SPL 变量使用一个您为列名称使用的同样的标识符，则数据库服务器假定该标识符的每一实例都是变量。请以表名称限定列名称，使用点符号表示法，以便将标识符用作列名称。

在下图中的 SELECT 语句中，customer.lname 是列名称，lname 是变量名称。

图: SELECT 语句中的列名称和变量名称。

```
CREATE PROCEDURE table_test()

        DEFINE lname CHAR(15);
        LET lname = 'Miller';

        SELECT customer.lname INTO lname FROM customer
        WHERE customer_num = 502;
        . . .
        END PROCEDURE;
```

变量和 SQL 函数

如果您为 SPL 变量使用与为 SQL 函数一样的标识符，则数据库服务器假定该表达式的每一实例都是变量，并不允许使用该 SQL 函数。在定义该变量的代码块内，您不可使用该 SQL 函数。下图中的示例展示在其中定义名为 user 的变量的 SPL 过程内的块。此定义不允许在 BEGIN END 块中使用 USER 函数。

图: 不允许在 BEGIN END 块中使用 USER 函数的过程。

```
CREATE PROCEDURE user_test()
        DEFINE name CHAR(10);
        DEFINE name2 CHAR(10);
        LET name = user; -- the SQL function

        BEGIN
        DEFINE user CHAR(15); -- disables user function
        LET user = 'Miller';
        LET name = user; -- assigns 'Miller' to variable name
        END
        . . .
        LET name2 = user; -- SQL function again
```

12.3.2 声明全局变量

全局变量将它的值存储在内存中，其他 SPL 例程可用，由相同的用户会话运行在同一数据库上。全局变量有下列特征：

- 它需要缺省值。
- 可在任何 SPL 例程中使用它，虽然必须在使用它的每一例程中定义它。
- 它将它的值从一个 SPL 例程带到另一个，直到会话结束为止。

限制： 您不可将集合变量定义为全局变量。

下图展示分享一个全局变量的两个 SPL 函数。

图: 分享一个全局变量的两个 SPL 函数。

```
CREATE FUNCTION func1()  RETURNING INT;
                DEFINE GLOBAL gvar INT DEFAULT 2;
                LET gvar = gvar + 1;
                RETURN gvar;
                END FUNCTION;

                CREATE FUNCTION func2()  RETURNING INT;
                DEFINE GLOBAL gvar INT DEFAULT 5;
                LET gvar = gvar + 1;
                RETURN gvar;
                END FUNCTION;
```

虽然您必须定义带有缺省值的全局变量，但仅在您首次使用它时，将变量设置为缺省值。如果您以给定的顺序在下图中执行这两个函数，则 gvar的值将为 4。

图: 全局变量缺省值。

```
EXECUTE FUNCTION func1();
                EXECUTE FUNCTION func2();
```

但是，如果您以相反的顺序执行函数，如下图所示，则 gvar 的值将为 7。

图: 全局变量缺省值。

```
EXECUTE FUNCTION func2();
                EXECUTE FUNCTION func1();
```

要了解更多信息，请参阅 执行例程。

12.3.3 赋值给变量

在 SPL 例程内，请使用 LET 语句将值分配给您已定义的变量。

如果您未赋值给变量，或通过传递给例程的参数，或通过 LET 语句，则该变量有未定义的值。

未定义的值与 NULL 值不同。如果您尝试以 SPL 例程内未定义的值使用变量，则会收到错误。

您可以下列任一方式赋值给例程变量：

- 使用 LET 语句。
- 使用 SELECT INTO 语句。
- 将 CALL 语句与带有 RETURNING 子句的过程一起使用。
- 使用 EXECUTE PROCEDURE INTO 或 EXECUTE FUNCTION INTO 语句。

LET 语句

以 LET 语句，您可以等号（=）和有效的表达式或函数名称来使用一个或多个变量名称。下图中的每一示例都是有效的 LET 语句。

图: 有效的 LET 语句。

```
LET a = 5;
        LET b = 6; LET c = 10;
        LET a,b = 10,c+d;
        LET a,b = (SELECT cola,colb
        FROM tab1 WHERE cola=10);
        LET d = func1(x,y);
```

GBase 8s 允许您将值分配给 opaque 类型变量、row 类型变量，或 row 类型的字段。您还可将外部函数或另一 SPL 函数的值返回到 SPL 变量。

假设您定义命名的 row 类型 zip_t 和 address_t，如 图 1 所示。每当您定义 row 类型变量时，您必须在可使用它之前初始化该变量。下图展示您可能如何定义和初始化 row 类型变量。您可使用任何 row 类型值来初始化该变量。

图: 定义和初始化 row 类型变量。

```
DEFINE a address_t;
        LET a = ROW ('A Street', 'Nowhere', 'AA',
        ROW(NULL, NULL))::address_t
```

在您定义并初始化 row 类型变量之后，您可编写下图所示的 LET 语句。

图: 编写 LET 语句。

```
LET a.zip.z_code = 32601;
        LET a.zip.z_suffix = 4555;
        -- Assign values to the fields of address_t
```

提示： 请以 variable.field or variable.field.field 的形式使用点符号表示法来访问 row 类型的字段，如 处理 row 类型数据 描述的那样。

假设您定义 opaque-type point，其包含定义二维点的两个值，且该值的文本表示为 '(x,y)'。您还可能有计算圆的周长的函数 circum()，给定的点 '(x,y)' 和半径 r。

如果您定义以一点为圆心的 opaque 类型 center，以及计算圆的周长的函数 circum()，基于点和半径，您可为每一变量编写变量声明。在下图中，c 是一个 opaque 类型变量，d 保存外部函数 circum() 返回的值。

图: 编写变量声明。

```
DEFINE c point;
        DEFINE r REAL;
        DEFINE d REAL;

        LET c = '(29.9,1.0)' ;
        -- Assign a value to an opaque type variable

        LET d = circum( c, r );
        -- Assign a value returned from circum()
```

GBase 8s SQL 指南：语法 详细地描述 LET 语句的语法。

赋值给变量的其他方式

您可使用 SELECT 语句来从数据库访存一个值，并直接地将它分配给变量，如下图所示。

图: 从数据库访存一个值，并直接地将它分配给变量。

```
SELECT fname, lname INTO a, b FROM customer
        WHERE customer_num = 101
```

请使用 CALL 或 EXECUTE PROCEDURE 语句来将由 SPL 函数或外部函数返回的值分配给一个或多个 SPL 变量。您可能使用下图中的一个语句来将来自 SPL 函数 read_address 的全名和地址返回到指定的 SPL 变量内。

图: 返回来自 SPL 函数的全名和地址。

```
EXECUTE FUNCTION read_address('Smith')
        INTO p_fname, p_lname, p_add, p_city, p_state,
        p_zip;

        CALL read_address('Smith')
        RETURNING p_fname, p_lname, p_add, p_city,
        p_state, p_zip;
```

12.4 SPL 例程中的表达式

您可在 SPL 例程中使用任何 SQL 表达式，除了聚集表达式以外。GBase 8s SQL 指南：语法 提供 SQL 表达式的完整语法与描述。

下列示例包含 SQL 表达式：

```
var1
        var1 + var2 + 5
        read_address('Miller')
        read_address(lastname = 'Miller')
        get_duedate(acct_num) + 10 UNITS DAY
        fname[1,5] || ''|| lname '(415)' || get_phonenum(cust_name)
```

12.5 编写语句块

每个 SPL 例程至少有一个语句块，它是在 CREATE 语句与 END 语句之间的一组 SQL 和 SPL 语句。您可在语句块内使用任何 SPL 语句或任何允许的 SQL 语句。要了解在 SPL 语句块内不允许使用的 SQL 语句的列表，请参阅《GBase 8s SQL 指南：语法》 中语句块段的描述。

12.5.1 隐式的和显式的语句块

在 SPL 例程中，隐式的语句块从 CREATE 语句的结尾扩展到 END 语句的开头。您还可定义显式的语句块，它以 BEGIN 语句开头并以 END 语句结尾，如下图所示。

图: 显式的语句块。

```
BEGIN
        DEFINE distance INT;
        LET distance = 2;
        END
```

显式的语句块允许您定义仅在语句块内有效的变量或处理。例如，您可定义或重新定义变量，或以不同的方式处理异常，仅对于显式的语句块的作用域。

下图中的 SPL 函数有一个显式的语句块，它重新定义在隐式的块中定义的变量。

图: 重新定义在隐式的块中定义的变量的显式的语句块。

```
CREATE FUNCTION block_demo()
            RETURNING INT;
```

```
        DEFINE distance INT;
        LET distance = 37;
        BEGIN
        DEFINE distance INT;
        LET distance = 2;
        END
        RETURN distance;

        END FUNCTION;
```

在此示例中，隐式的语句块定义变量 distance 并赋值 37。显式的语句块定义名为 distance 的不同的变量，并赋值 2。然而，RETURN 语句返回存储在第一个 distance 变量中的值，即 37。

12.5.2 FOREACH 循环

FOREACH 循环定义游标，指向一组中的一项的特定的=标识符，或为一组行，或为集合中的元素。

FOREACH 循环声明并打开游标，从数据库访存行，处理该组中的每一项，然后关闭游标。如果 SELECT、EXECUTE PROCEDURE 或 EXECUTE FUNCTION 语句可能返回多行，则您必须声明游标。在您声明游标之后，将 SELECT、EXECUTE PROCEDURE 或 EXECUTE FUNCTION 语句放置在其内。

返回一组行的 SPL 例程称为游标例程，因为您必须使用游标来访问它返回的数据。不返回值、返回单个值或任何其他值的 SPL 例程不需要游标，称为无游标例程。FOREACH 循环声明并打开游标，从数据库访存行或集合，处理该组中的每一项，然后关闭游标。如果 SELECT、EXECUTE PROCEDURE 或 EXECUTE FUNCTION 语句可能返回多个行或集合，则您必须声明游标。在您声明游标之后，请将 SELECT、EXECUTE PROCEDURE 或 EXECUTE FUNCTION 语句放置其内。

在 FOREACH 循环中，您可使用 EXECUTE FUNCTION 或 SELECT INTO 语句来执行为迭代函数的外部函数。

12.5.3 FOREACH 循环定义游标

FOREACH 循环以 FOREACH 关键字开始，并以 END FOREACH 结束。在 FOREACH 与 END FOREACH 之间，您可声明游标或使用 EXECUTE PROCEDURE 或 EXECUTE FUNCTION。下图中的两个示例展示 FOREACH 循环的结构。

图: FOREACH 循环的结构。

```
FOREACH cursor FOR
        SELECT column INTO variable FROM table
```

```
        . . .
        END FOREACH;

        FOREACH
        EXECUTE FUNCTION name() INTO variable;
        END FOREACH;
```

下图创建使用 FOREACH 循环的例程来在 employee 表上操作。

图: 对 employee 表操作的 FROEACH 循环。

```
CREATE_PROCEDURE increase_by_pct( pct INTEGER )
        DEFINE s INTEGER;

        FOREACH sal_cursor FOR
        SELECT salary INTO s FROM employee
        WHERE salary > 35000
        LET s = s + s * ( pct/100 );
        UPDATE employee SET salary = s
        WHERE CURRENT OF sal_cursor;
        END FOREACH;

        END PROCEDURE;
```

前图中的例程执行 FOREACH 内的这些任务：

- 声明游标
- 一次从 employee 表选择一个 salary 值
- 按百分率提高 salary
- 以新的 salary 更新 employee
- 访存下一个 salary 值

将 SELECT 语句放置在游标内，因为它返回表中所有大于 35000 的薪酬。

UPDATE 语句中的 WHERE CURRENT OF 子句仅更新该游标当前定位在其上的行，并在当前行上设置更新游标。更新游标在该行上放置更新锁，以便于其他用户不可更新该行，直到您的更新发生为止。

如果 FOREACH 循环内的 UPDATE 或 DELETE 语句使用 WHERE CURRENT OF 子句，则 SPL 例程将自动地设置更新游标。如果您使用 WHERE CURRENT OF，则必须显式地引用 FOREACH 语句内的游标。如果您正在使用更新游标，则可在 FOREACH 语句之前添加 BEGIN WORK 语句，并在 END FOREACH 之后添加 COMMIT WORK 语句，如下图所示。

图: 自动地设置更新游标。

```
BEGIN WORK;
        FOREACH sal_cursor FOR
        SELECT salary INTO s FROM employee WHERE salary > 35000;
        LET s = s + s * ( pct/100 );
        UPDATE employee SET salary = s WHERE CURRENT OF sal_cursor
        END FOREACH;
        COMMIT WORK;
```

对于前图中 FOREACH 循环的每一迭代，需要新锁（如果您使用行级别锁定的话）。在 FOREACH 循环的最后迭代之后，COMMIT WORK 语句释放所有的锁（并将所有更新了的行作为单个事务提交）。

要在循环的每一迭代之后提交更新了的行，您必须打开游标 WITH HOLD，并在 FOREACH 循环内包括 BEGIN WORK 和 COMMIT WORK 语句，如下列 SPL 例程那样。

图: 在循环的每一迭代之后提交更新了的行。

```
CREATE PROCEDURE serial_update();
        DEFINE p_col2 INT;
        DEFINE i INT;
        LET i = 1;
        FOREACH cur_su WITH HOLD FOR
        SELECT col2 INTO p_col2 FROM customer WHERE 1=1
        BEGIN WORK;
        UPDATE customer SET customer_num = p_col2 WHERE CURRENT
OF cur_su;
        COMMIT WORK;
        LET i = i + 1;
        END FOREACH;
        END PROCEDURE;
```

SPL 例程 serial_update() 提交每一行作为分开的事务。

对 FOREACH 循环的限制

在 FOREACH 循环内，SELECT 查询必须在更改该 SELECT 游标的数据集的任何 DELETE、INSERT 或 UPDATE 操作之前执行完成。确保 SELECT 查询完成的一种方式是，在 SELECT 语句中使用 ORDER BY 子句。ORDER BY 子句在该列上创建索引，并通过在同一 FOREACH 循环中更改 SELECT 语句的查询结果的 UPDATE、INSERT、DELETE 语句来防止导致的错误。

12.5.4 IF - ELIF - ELSE 结构

下列 SPL 例程使用 IF - ELIF - ELSE 结构来比较该例程接受的两个参数。

图: 比较两个参数的 IF - ELIF - ELSE 结构。

```
CREATE FUNCTION str_compare( str1 CHAR(20), str2 CHAR(20))
        RETURNING INTEGER;

        DEFINE result INTEGER;

        IF str1 > str2 THEN
        LET result = 1;
        ELIF str2 > str1 THEN
        LET result = -1;
        ELSE
        LET result = 0;
        END IF
        RETURN result;
        END FUNCTION;
```

假设您以下图所示的列定义名为 manager 的表。

图: 定义 manager 表。

```
CREATE TABLE manager
        (
        mgr_name     VARCHAR(30),
        department VARCHAR(12),
        dept_no      SMALLINT,
        direct_reports SET( VARCHAR(30) NOT NULL ),
        projects LIST( ROW ( pro_name VARCHAR(15),
        pro_members SET( VARCHAR(20) NOT NULL ) )
        NOT NULL),
        salary       INTEGER,
        );
```

下列 SPL 例程使用 IF - ELIF - ELSE 结构来检查 direct_reports 列中 SET 中元素的数目，并基于该结果来调用不同的外部例程。

图: 检查 SET 中元素数的 IF - ELIF - ELSE 结构。

```
CREATE FUNCTION checklist( d SMALLINT )
        RETURNING VARCHAR(30), VARCHAR(12), INTEGER;
```

```
DEFINE name VARCHAR(30);
DEFINE dept VARCHAR(12);
DEFINE num INTEGER;

SELECT mgr_name, department,
CARDINALITY(direct_reports)
FROM manager INTO name, dept, num
WHERE dept_no = d;
IF num > 20 THEN
EXECUTE FUNCTION add_mgr(dept);
ELIF num = 0 THEN
EXECUTE FUNCTION del_mgr(dept);
ELSE
RETURN name, dept, num;
END IF;

END FUNCTION;
```

cardinality() 函数计数集合包含的元素数。要获取更多信息，请参阅 基数函数。

SPL 例程中的 IF - ELIF - ELSE 结构有至多下列四个部分：

- IF THEN 条件

  如果跟在 IF 语句之后的该条件为 TRUE，则例程执行 IF 块中的语句。如果该条件为假，则例程对 ELIF 条件求值。

  IF 语句中的表达式可为任何有效的条件，如 GBase 8s SQL 指南：语法 的 Condition 段描述的那样。要了解 IF 语句的完整语法和详细的讨论，请参阅《GBase 8s SQL 指南：语法》。

- 一个或多个 ELIF 条件（可选的）

  仅当 IF 条件为假时，例程才对 ELIF 条件求值。如果 ELIF 条件为真，则例程执行 ELIF 块中的语句。如果 ELIF 条件为假，则例程或对下一个 ELIF 块求值，或执行 ELSE 语句。

- ELSE 条件（可选的）

  如果 IF 条件和所有 ELIF 条件都为假，则例程执行 ELSE 块中的语句。

- END IF 语句

  END IF 语句结束该语句块。

12.5.5 添加 WHILE 和 FOR 循环

WHILE 与 FOR 语句都可在 SPL 例程中创建执行循环。WHILE 循环以 WHILE condition 开始，只要条件为真就执行语句块，并以 END WHILE 结束。

下图展示有效的 WHILE 条件。只要在 WHILE 语句中指定的条件为真，例程就执行 WHILE 循环。

图: 只要在 WHILE 语句中指定的条件为真，例程就执行 WHILE 循环。

```
CREATE PROCEDURE test_rows( num INT )

        DEFINE i INTEGER;
        LET i = 1;

        WHILE i < num
        INSERT INTO table1 (numbers) VALUES (i);
        LET i = i + 1;
        END WHILE;

        END PROCEDURE;
```

前图中的 SPL 例程接受整数作为参数，然后在它每一次执行 WHILE 循环时，就将整数值插入到 table1 的 numbers 列内。插入的值从 1 开始，且增大到 num - 1。

请当心，不要创建无限的循环，如下图所示。

图: 接受整数为参数，然后将整数值插入到 numbers 列的例程。

```
CREATE PROCEDURE endless_loop()

        DEFINE i INTEGER;
        LET i = 1;
        WHILE ( 1 = 1 )                 -- don't do this!
        LET i = i + 1;
        INSERT INTO table1 VALUES (i);
        END WHILE;

        END PROCEDURE;
```

FOR 循环从 FOR 语句扩展到 END FOR 语句，并执行在 FOR 语句中定义的指定次数的迭代。下图展示在 FOR 循环中定义迭代的几种方式。

对于 FOR 循环的每一迭代，重置迭代变量（在后面的示例中声明为 i），并以该变量的新值执行该循环内的语句。

图: 定义 FOR 循环中的迭代。

```
FOR i = 1 TO 10
        . . .
        END FOR;

        FOR i = 1 TO 10 STEP 2
        . . .
        END FOR;

        FOR i IN (2,4,8,14,22,32)
        . . .
        END FOR;

        FOR i IN (1 TO 20 STEP 5, 20 to 1 STEP -5, 1,2,3,4,5)
        . . .
        END FOR:
```

在第一个示例中，只要 i 介于 1 之间 10，包括 1 与 10，该 SPL 过程就执行 FOR 循环。在第二个示例中，i 从 1 到 3、5、7，等等递进，但从不超过 10。第三个示例检查 i 是否在定义了的值集之内。在第四个示例中，当 i 为 1、6、11、16、20、15、10、5、1、2、3、4 或 5 时，该 SPL 过程执行循环——换言之，执行循环 13 次。

提示： WHILE 循环与 FOR 循环之间的主要差异是，FOR 循环保证会结束，但 WHILE 循环不然。FOR 语句指定循环执行的确切次数，除非语句导致例程退出该循环。使用 WHILE，可能创建无限的循环。

12.5.6 退出循环

在没有标签的 FOR、FOREACH、LOOP 或 WHILE 循环中，您可使用 CONTINUE 或 EXIT 语句来控制循环的执行。

- CONTINUE 导致例程跳过该循环的剩余语句，并移至 FOR、LOOP 或 WHILE 语句的下一迭代。
- EXIT 终止该循环，并导致例程继续执行跟在 END FOR、END LOOP 或 END WHILE 关键字之后的第一个语句。

请记住，当 EXIT 出现在为嵌套循环语句的最内层循环的 FOREACH 语句内时，它必须后跟 FOREACH 关键字。当 EXIT 出现在在 FOR、LOOP 或 WHILE 语句内时，它可不紧跟关键字出现，但如果您指定一个关键字，该关键字与从其发出了 EXIT 语句的循环语句不相匹配，则发出错误。如果 EXIT 出现在循环语句的上下文之外，则也发出错误。

要获取关于 SPL 例程中的循环的更多信息，包括带标签的循环，请参阅《GBase 8s SQL 指南：语法》。

下图展示在 FOR 循环内的 CONTINUE 和 EXIT 的示例。

图: FOR 循环内的 CONTINUE 和 EXIT 的示例。

```
FOR i = 1 TO 10
        IF i = 5 THEN
        CONTINUE FOR;
        . . .
        ELIF i = 8 THEN
        EXIT FOR;
        END IF;

        END FOR;
```

提示： 您可使用 CONTINUE 和 EXIT 来提升 SPL 例程的性能，以免执行不必要的循环。

12.6 从 SPL 函数返回值

SPL 函数可返回一个或多个值。要是您的 SPL 函数返回值，需要包括下列两个部分：

1. 在指定要返回的值的数目及其数据类型的 CREATE PROCEDURE 或 CREATE FUNCTION 语句中编写 RETURNING 子句。
2. 在函数体内，输入显式地返回值的 RETURN 语句。

提示： 您可以返回值的 CREATE PROCEDURE 语句来定义例程，但在那种情况下，该例程实际上是函数。但例程返回值时，推荐您使用 CREATE FUNCTION 语句。

在您（以 RETURNING 语句）定义返回子句之后，SPL 函数可返回那些与指定的数目和数据类型相匹配的值，或根本不返回值。如果您指定返回子句，且 SPL 例程未返回实际的值，则仍将它视为函数。在那种情况下，例程为在返回子句中定义的每一值都返回一个 NULL 值。

SPL 函数可返回变量、表达式，或另一函数调用的结果。如果 SPL 函数返回变量，则该函数必须首先通过下列方法之一赋值给该变量：

- LET 语句
- 缺省值
- SELECT 语句
- 将值传至该变量内的另一函数

SPL 函数返回的每一值最长可为 32 KB。

重要： SPL 函数的返回值必须为特定的数据类型。您不可指定类属行或类属集合数据类型作为返回类型。

12.6.1 返回单个值

下图展示 SPL 函数可如何返回单个值。

图: 返回单个值的 SPL 函数。

```
CREATE FUNCTION increase_by_pct(amt DECIMAL, pct DECIMAL)
        RETURNING DECIMAL;

        DEFINE result DECIMAL;

        LET result = amt + amt * (pct/100);

        RETURN result;

        END FUNCTION;
```

increase_by_pct 函数收到两个 DECIMAL 值的参数，一个为要增加的数量，一个为要增加的百分比。指定该函数的返回子句将返回一个 DECIMAL 值。RETURN 语句返回存储在 result 中的 DECIMAL 值。

12.6.2 返回多个值

SPL 函数可从表的单个行返回多个值。下图展示从表的单个行返回两个列值的 SPL 函数。

图: 从表的单个行返回两个列值的 SPL 函数。

```
CREATE FUNCTION birth_date( num INTEGER )
        RETURNING VARCHAR(30), DATE;

        DEFINE n VARCHAR(30);
        DEFINE b DATE;

        SELECT name, bdate INTO n, b FROM emp_tab
        WHERE emp_no = num;
        RETURN n, b;
```

```
        END FUNCTION;
```

该函数从 emp_tab 表的一行将两个值（名和生日）返回给调用的例程。在此情况下，必须准备调用的例程来处理返回的 VARCHAR 和 DATE 值。

下图展示从多行返回多个值的 SPL 函数。

图: 从多行返回多个值的 SPL 函数。

```
CREATE FUNCTION birth_date_2( num INTEGER )
        RETURNING VARCHAR(30), DATE;
        DEFINE n VARCHAR(30);
        DEFINE b DATE;
        FOREACH cursor1 FOR
        SELECT name, bdate INTO n, b FROM emp_tab
        WHERE emp_no > num
        RETURN n, b WITH RESUME;
        END FOREACH;
        END FUNCTION;
```

在前图中，SELECT 语句从其员工编号大于用户输入的编号的行集访存两个值。满足该条件的行集可能包含一行、多行，或零行。由于 SELECT 语句可返回多行，因此将它放置在游标内。

提示： 当 SPL 例程内的语句未返回行时，为对应的 SPL 变量赋值 NULL。

RETURN 语句使用 WITH RESUME 关键字。当执行 RETURN WITH RESUME 时，将控制返回到调用的例程。但在下一次（通过 FETCH 或通过调用的例程中的游标的下一迭代）调用该 SPL 函数时，SPL 函数中的所有变量保持它们的相同的值，并从紧跟在 RETURN WITH RESUME 语句之后的语句继续执行。

如果您的 SPL 例程返回多个值，则调用的例程必须能够通过游标或循环来处理多个值，如下：

- 如果调用的例程为 SPL 例程，则它需要 FOREACH 循环。
- 如果调用的例程为 GBase 8s ESQL/C 程序，则它需要以 DECLARE 语句声明的游标。
- 如果调用的例程为外部的例程，则它需要与编写该例程的语言相适应的游标或循环。

重要： 由 UDR 从本地服务器的外部数据库返回的值必须为内建的数据类型，或 UDT 显式地强制转型为内建的类型，或基于内建的类型的 DISTINCT 类型并显式地强制转型为内建的类型。此外，您必须定义 UDR 和参与的数据库中的所有强制转型。

下列是您可跨数据库执行的 SQL 操作的示例：

```
database db1;
```

```
create table ltab1(lcol1 integer, lcol2 boolean, lcol3 lvarchar);
insert into ltab1 values(1, 't', "test string 1");

database db2;
create table rtab1(r1col1 boolean, r1col2 blob, r1col3 integer)
put r1col2 in (sbsp);
create table rtab2(r2col1 lvarchar, r2col2 clob) put r2col2 in (sbsp);
create table rtab3(r3col1 integer, r3col2 boolean,
r3col3 lvarchar, r3col4 circle);

create view rvw1 as select  * from rtab3;
```

（该示例为跨数据库 Insert。）

```
database db1;
create view lvw1 as select * from db2:rtab2;
insert into db2:rtab1 values('t',
filetoblob('blobfile', 'client', 'db2:rtab1', 'r1col2'), 100);
insert into db2:rtab2 values("inserted directly to rtab2",
filetoclob('clobfile', 'client', 'db2:rtab2', 'r2col2'));
insert into db2:rtab3 (r3col1, r3col2, r3col3)
select lcol1, lcol2, lcol3 from ltab1;
insert into db2:rvw1 values(200, 'f', "inserted via rvw1");
insert into lvw1 values ("inserted via lvw1", NULL);
```

12.7 处理 row 类型数据

在 SPL 例程中，您可使用命名的 ROW 类型和未命名的 ROW 类型作为参数定义、参数、变量定义和返回值。要获取关于如何在 SPL 中声明 ROW 变量的信息，请参阅 声明 row 类型变量。

下图定义 row 类型 salary_t 和 emp_info 表，它们是本部分使用的示例。

图: 定义 row 类型 salary_t 和 emp_info 表

```
CREATE ROW TYPE salary_t(base MONEY(9,2), bonus MONEY(9,2))

CREATE TABLE emp_info (emp_name VARCHAR(30), salary salary_t);
```

emp_info 表有员工姓名和薪酬信息的列。

12.7.1 点符号表示法的优先顺序

以 GBase 8s，SPL 例程中 SQL 语句中使用点符号表示法（如在 proj.name）被解释为有三种含义之一，优先顺序如下：

1. variable.field
2. column.field
3. table.column

换言之，首先将表达式 proj.name 求值为 variable.field。如果例程未找到变量 proj，则它将该表达式求值为 column.field。如果例程未找到列 proj，则它将该表达式求值为 table.column。（如果不可将名称解析为数据库中对象的标识符，或在 SPL 例程中声明了的变量或字段，则返回错误。）

12.7.2 更新 row 类型表达式

从 SPL 例程内，您可使用 ROW 变量来更新 row 类型表达式。下图展示当员工的基本薪酬按某一百分比增长时，用于更新 emp_info 表的 SPL 过程 emp_raise。

图: 用于更新 emp_info 表的 SPL 过程。

```
CREATE PROCEDURE emp_raise( name VARCHAR(30),
                pct DECIMAL(3,2) )

                DEFINE row_var salary_t;

                SELECT salary INTO row_var FROM emp_info
                WHERE emp_name = name;

                LET row_var.base = row_var.base * pct;

                UPDATE emp_info SET salary = row_var
                WHERE emp_name = name;
                END PROCEDURE;
```

SELECT 语句将来自 emp_info 表的 salary 列的行选择到 ROW 变量 row_var 内。

emp_raise 过程使用 SPL 点符号表示法来直接地访问变量 row_var 的 base 字段。在此情况下，点符号表示法意味着 variable.field。emp_raise过程重新计算 row_var.base 的值作为 (row_var.base * pct)。然后，该过程以新的 row_var 值来更新 emp_info 表的 salary 列。

重要：在可设置或引用 row 类型变量字段之前，必须将它初始化为行。您可以 SELECT INTO 语句或 LET 语句初始化 row 类型变量。

12.8 处理集合

集合是同一数据类型的一组元素，诸如 SET、MULTISET 或 LIST。

表可能包含集合，存储集合作为列的内容，或作为列内 ROW 类型的字段。集合可为简单的或嵌套的。简单的集合是内建的、opaque 或 distinct 数据类型的 SET、MULTISET 或 LIST。嵌套的集合是包含其他集合的集合。

12.8.1 集合数据类型

本章节的下列部分凭借几个不同的示例来展示您可如何在 SPL 程序中操纵集合。

在 SPL 程序中处理集合的基本内容是使用 numbers 表来说明的，如下图所示。

图: 在 SPL 程序中处理集合。

```
CREATE TABLE numbers
                (
                id INTEGER PRIMARY KEY,
                primes          SET( INTEGER NOT NULL ),
                evens           LIST( INTEGER NOT NULL ),
                twin_primes   LIST( SET( INTEGER NOT NULL )
                NOT NULL )
```

primes 和 evens 列保存简单的集合。twin_primes 列保存嵌套的集合，SET 的 LIST。（双素数是一对相差 2 的连续素数，诸如 5 和 7，或 11 和 13。）设计 twin_primes 列以允许您输入这样的值对。

本章节中的一些示例使用下图中的 polygons 表，来说明如何操纵集合。polygons 表包含集合来表示两维图形数据。例如，假设您定义名为point 的 opaque 数据类型，其有表示两维点的 x 和 y 坐标的两个双精度值，其坐标可能表示为 '1.0, 3.0'。使用 point 数据类型，您可创建包含一系列定义多边形的点的表。

图: 操纵集合。

```
CREATE OPAQUE TYPE point ( INTERNALLENGTH = 8);


                CREATE TABLE polygons
                (
                id              INTEGER PRIMARY KEY,
                definition   SET( point NOT NULL )
                );
```

polygons 表中的 definition 列包含简单的集合，point 值的 SET。

12.8.2 准备集合数据类型

在您可访问和处理简单的或嵌套的集合的个别元素之前，您必须执行下列任务：

- 声明集合变量来保存该集合。
- 声明元素变量来保存集合的个别元素。
- 将集合从数据库选择至集合变量内。

在您做了这些初始的步骤之后，您可将元素插入到集合内，或选择或处理已在集合中的元素。

在下列部分中，使用 numbers 表为示例，说明每一步骤。

提示： 您可在任何 SPL 例程中处理集合。

声明集合变量

在您可从数据库将集合检索至 SPL 例程内之前，您必须声明集合变量。下图展示如何声明集合变量来从 numbers 表检索 primes 列。

图: 声明集合变量。

```
DEFINE p_coll SET( INTEGER NOT NULL );
```

DEFINE 语句声明集合变量 p_coll，其类型与存储在 primes 列中的集合的数据类型相匹配。

声明元素变量

在您声明集合变量之后，请声明元素变量来保存该集合的个别元素。元素变量的数据类型必须与集合元素的数据类型相匹配。

例如，要保存 primes 列中 SET 的元素，请使用诸如下图所示的一种元素变量声明。

图: 元素变量声明。

```
DEFINE p INTEGER;
```

要声明保存 twin_primes 列的元素的变量，其保存嵌套的集合，请使用诸如下图所示的一种变量声明。

图: 变量声明。

```
DEFINE s SET( INTEGER NOT NULL );
```

变量 s 保存整数的 SET。每一 SET 是存储在 twin_primes 中的 LIST 的一个元素。

将集合选择至集合变量内

在您声明集合变量之后，您可将集合访存至它内。要将集合访存至集合变量内，请输入 SELECT INTO 语句，该语句从数据库将集合列选择至您已命名了的集合变量内。

例如，要选择保存在 numbers 的 primes 列的一行中的集合，请添加 SELECT 语句至您的 SPL 例程，诸如下图展示的一个。

图: 添加 SELECT 语句来选择存储在一行中的集合。

```
SELECT primes INTO p_coll FROM numbers
                WHERE id = 220;
```

SELECT 语句中的 WHERE 子句指定您想要选择只存储在 numbers 的一行中的集合。该语句将集合放置到集合变量 p_coll 内，图 1 声明它。

现在，变量 p_coll 保存来自 primes 列的集合，它会包含值 SET {5,7,31,19,13}。

12.8.3 将元素插入至集合变量内

在您将集合检索至集合变量内之后，您可将值插入至该集合变量。INSERT 语句的语法略有不同，这依赖于您想要条件到的 集合的类型。

插入至 SET 或 MULTISET 内

要插入至存储在集合变量中的 SET 或 MULTISET 内，请使用 INSERT 语句，并跟在带有集合变量的 TABLE 关键字之后，如下图所示。

图: 插入至存储在集合变量中的 SET 或 MULTISET 内。

```
INSERT INTO TABLE(p_coll) VALUES(3);
```

TABLE 关键字使得集合变量成为集合派生的表。在 处理 SELECT 语句中的集合 部分中描述集合派生的表。前图派生的集合是一列的虚拟表，集合的每一元素表示表的一行。在插入之前，请将 p_coll 考虑作为包含下图展示的行（元素）的虚拟表。

图: 虚拟表元素。

```
5
        7
        31
        19
        13
```

在插入之后，p_coll 可能看上去像下图所示的虚拟表一样。

图: 虚拟表元素。

```
5
    7
```

```
31
19
13
3
```

由于该集合为 SET，因此，将新的值添加到该集合，但未定义新元素的位置。对于 MULTISET ，适用同样的原理。

提示： 您一次仅可将一个值插入至简单的集合内。

插入至 LIST 内

如果集合为 LIST，则您可将新的元素添加在 LIST 中的特定点，或添加在 LIST 的末尾。如同 SET 或 MULTISET 一样，您必须首先定义集合变量，并从数据库将集合选择至该集合变量内。

下图展示您需要定义集合变量并从 numbers 表选择 LIST 至该集合变量内的语句。

图: 定义集合变量并选择 LIST。

```
DEFINE e_coll LIST(INTEGER NOT NULL);

        SELECT evens INTO e_coll FROM numbers
        WHERE id = 99;
```

此时，e_coll 的值可能为 LIST {2,4,6,8,10}。由于 e_coll 保存 LIST，因此，每一元素有在该列表中的编号的位置。要将元素添加在 LIST 中特定的位置，请将 AT position 子句添加到 INSERT 语句，如下图所示。

图: 将元素添加在 LIST 中特定的点。

```
INSERT AT 3 INTO TABLE(e_coll) VALUES(12);
```

现在，e_coll 中的 LIST 有元素 {2,4,12,6,8,10}，依此顺序。

您在 AT 子句中为 position 输入的值可为数值或变量，但它必须有 INTEGER 或 SMALLINT 数据类型。您不可使用字母、浮点数值、十进制值或表达式。

检查 LIST 集合的基数

有时，您可能想要将元素添加在 LIST 的末尾。在此情况下，您可使用 cardinality() 函数来找到 LIST 中的元素的编号，然后输入一个大于cardinality() 返回的值的位置。

GBase 8s 允许您随同存储在列中的集合来使用 cardinality() 函数，但不允许随同存储在集合变量中的集合来使用。在 SPL 例程中，您可以 SELECT 语句检查列中的集合的基数，并将该值返回给变量。

假设在 numbers 表中，其 id 列为 99 的那一行的 evens 列仍然包含集合 LIST {2,4,6,8,10}。这一次，您想要将元素 12 添加在该 LIST 的末尾。您可使用 SPL 过程 end_of_list 实现，如下图所示。

图: end_of_list SPL 过程。

```
CREATE PROCEDURE end_of_list()

        DEFINE n SMALLINT;
        DEFINE list_var LIST(INTEGER NOT NULL);

        SELECT CARDINALITY(evens) FROM numbers INTO n
        WHERE id = 100;

        LET n = n + 1;

        SELECT evens INTO list_var FROM numbers
        WHERE id = 100;

        INSERT AT n INTO TABLE(list_var) VALUES(12);

        END PROCEDURE;
```

在 end_of_list 中，变量 n 保存 cardinality() 返回的值，即，LIST 中的项数。LET 语句使 n 递增，以便于 INSERT 语句可在 LIST 的最后的位置插入值。SELECT 语句将来自该表的一行的集合选择至集合变量 list_var 内。INSERT 语句将元素 12 插入在该列表的结尾。

VALUES 子句的语法

当您插入至 SPL 集合变量内时，VALUES 子句的语法与当您插入至集合列内时是不一样的。将文字插入至集合变量内的语法规则如下：

在 VALUES 关键字之后使用圆括号来括起值的完整列表。

如果您正在插入至简单的集合内，则无需使用类型构造函数或方括号。

如果您正在插入至嵌套的集合，则您需要指定文字的集合。

12.8.4 从集合选择元素

假设您想要您的 SPL 例程从存储在集合变量内的集合选择元素，则请一次选择一个，以便于您可处理这些元素。

要在集合的元素间移动，您首先需要使用 FOREACH 语句来声明游标，就如同您会声明游标来在一组行间一同一样。下图展示 FOREACH 和 END FOREACH 语句，在它们之间还没有语句。

图: FOREACH 和 END FOREACH 语句。

```
FOREACH cursor1 FOR
        . . .
        END FOREACH
```

在 FOREACH 循环 和 GBase 8s SQL 指南：语法 中描述 FOREACH 语句。

下一主题，集合查询，描述在 FOREACH 与 END FOREACH 语句之间省略的那些语句。

下列部分中的示例是基于 图 2 的 polygons 表的。

集合查询

在您在 FOREACH 与 END FOREACH 语句之间声明游标之后，请您输入称为集合查询的特殊的、受限形式的 SELECT 语句。

集合查询是使用后跟集合变量的名称的 FROM TABLE 关键字的 SELECT 语句。下图展示此结构，称其为集合派生的表。

图: 集合派生的表。

```
FOREACH cursor1 FOR

        SELECT * INTO pnt FROM TABLE(vertexes)
        . . .
        END FOREACH
```

该 SELECT 语句使用集合变量 vertexes 作为集合派生的表。您可将集合派生的表视为一列的表，该集合的每一元素都是表的一行。例如，您可将存储在 vertexes 中的四个点的 SET 可视化作为带有四行的表，诸如下图展示的一个。

图: 带有四行的表。

```
'(3.0,1.0)'
        '(8.0,1.0)'
        '(3.0,4.0)'
        '(8.0,4.0)'
```

在前图中的 FOREACH 语句的第一个迭代之后，该集合查询选择 vertexes 中的第一个元素，并将其存储在 pnt 中，因此，pnt 包含值 '(3.0,1.0)'。

提示： 由于集合变量 vertexes 包含 SET，而不是 LIST，因此，vertexes 中的元素没有定义了的顺序。在真实的数据库中，值 '(3.0,1.0)' 可能不是 SET 中的第一个元素。

将集合查询添加至 SPL 例程

现在，您可将以 FOREACH 定义的游标和集合查询添加至 SPL 例程，如下例所示。

图: 以 FOREACH 定义的游标和集合查询。

```
CREATE PROCEDURE shapes()

        DEFINE vertexes SET( point NOT NULL );
        DEFINE pnt point;

        SELECT definition INTO vertexes FROM polygons
        WHERE id = 207;

        FOREACH cursor1 FOR
        SELECT * INTO pnt FROM TABLE(vertexes)
        . . .
        END FOREACH
        . . .
        END PROCEDURE;
```

以上展示的语句形成处理集合变量的元素的 SPL 例程的框架。要将集合分解为它的元素，请使用集合派生的表。在将集合分解为它的元素之后，该例程可单独地访问元素作为集合派生的表的行。既然您已选择了 pnt 中的一个元素，您就可更新或删除那个元素，如 更新集合元素 和 删除集合元素 描述的那样。

要了解集合查询的完整语法，请参阅《GBase 8s SQL 指南：语法》 中的 SELECT 语句。要了解集合派生的表的语法，请参阅《GBase 8s SQL 指南：语法》 中的“集合派生的表”段。

提示：如果您正在从不包含元素或包含零元素的集合选择，则您可使用未声明游标的集合查询。然而，如果该集合包含多个元素，且您未使用游标，则您会收到错误消息。

注意： 在上述程序段中，如果 FOREACH 游标定义内的查询（

```
SELECT * INTO pnt FROM TABLE(vertexes)
```

）已以分号（;）作为语句终止符终止了，则数据库服务器可能已发出了语法错误。在此，END FOREACH 关键字是逻辑的语句终止符。

12.8.5 删除集合元素

在您将个别的元素从集合变量选择至元素变量内之后，您可从集合删除该元素。例如，在您以集合查询从集合变量 vertexes 选择一个点之后，您可将该点从集合移除。

删除集合元素涉及的步骤包括：

1. 声明集合变量和元素变量。
2. 将集合从数据库选择至集合变量内。
3. 声明游标，以便于您可从集合变量一次选择一个元素。
4. 编写定位您想要删除的元素的循环或分支。
5. 使用 DELETE WHERE CURRENT OF 语句来从集合删除元素，该语句使用集合变量作为集合派生的表。

下图展示删除 vertexes 中四个点之一的例程，以便于多边形成为三角形，而不是矩形。

图: 删除四个点之一的例程。

```
CREATE PROCEDURE shapes()

        DEFINE vertexes SET( point NOT NULL );
        DEFINE pnt point;

        SELECT definition INTO vertexes FROM polygons
        WHERE id = 207;

        FOREACH cursor1 FOR
        SELECT * INTO pnt FROM TABLE(vertexes)
        IF pnt = '(3,4)' THEN
        -- calls the equals function that
        -- compares two values of point type
        DELETE FROM TABLE(vertexes)
        WHERE CURRENT OF cursor1;
        EXIT FOREACH;
        ELSE
        CONTINUE FOREACH;
        END IF;
        END FOREACH
        . . .
        END PROCEDURE;
```

在前图中，FOREACH 语句声明游标。SELECT 语句是集合派生的查询，从集合变量 vertexes 一次将选择一个元素至元素变量 pnt。

IF THEN ELSE 结构测试当前在 pnt 中的值，看它是否为点 '(3,4)'。请注意，表达式 pnt = '(3,4)' 调用在点数据类型上定义的 equal() 函数的实例。如果 pnt 中的当前值为 '(3,4)'，则 DELETE 语句删除它，且 EXIT FOREACH 语句退出该游标。

提示： 从存储在集合变量中的集合删除元素，未将它从存储在数据库中的集合删除。在您从集合变量删除元素之后，您必须以新的集合更新存储在数据库中的集合。要了解展示如何更新集合列的示例，请参阅 更新数据库中的集合。

在《GBase 8s SQL 指南：语法》中描述 DELETE 语句的语法。

更新数据库中的集合

在您（通过删除、更新或插入元素）更改 SPL 例程中集合变量的内容之后，您必须以新的集合更新数据库。

要更新数据库中的集合，请添加一个设置表中的集合列的 UPDATE 语句，设置为更新了的集合变量的内容。例如，下图中的 UPDATE 语句展示如何更新 polygons 表，来将 definition 列设置为存储在集合变量 vertexes 中的新集合。

图: 更新数据库中的集合。

```
CREATE PROCEDURE shapes()

        DEFINE vertexes SET(point NOT NULL);
        DEFINE pnt point;

        SELECT definition INTO vertexes FROM polygons
        WHERE id = 207;

        FOREACH cursor1 FOR
        SELECT * INTO pnt FROM TABLE(vertexes)
        IF pnt = '(3,4)' THEN
        -- calls the equals function that
        -- compares two values of point type
        DELETE FROM TABLE(vertexes)
        WHERE CURRENT OF cursor1;
        EXIT FOREACH;
        ELSE
        CONTINUE FOREACH;
        END IF;
        END FOREACH

        UPDATE polygons SET definition = vertexes
        WHERE id = 207;
```

```
        END PROCEDURE;
```

现在，shapes() 例程完成。在您运行 shapes() 之后，更新存储在其 ID 列为 207 的行中的集合，以便于它包含三个值，而不是四个。

您可使用 shapes() 例程作为框架，来编写操纵集合的其他 SPL 例程。

出现在存储在 polygons 表的 207 的 definition 列中的集合的元素罗列如下：

```
'(3,1)'
        '(8,1)'
        '(8,4)'
```

删除整个集合

如果您想要删除集合的所有元素，您可使用单个 SQL 语句。您无需声明游标。要删除整个集合，您必须执行下列任务：

- 定义集合变量。
- 将该集合从数据库选择至集合变量内。
- 输入使用集合变量作为集合派生的表的 DELETE 语句。
- 更新来自数据库的集合。

下图展示您可能在 SPL 例程中用于删除整个集合的语句。

图: 删除整个集合的 SPL 例程。

```
DEFINE vertexes SET( INTEGER NOT NULL );

        SELECT definition INTO vertexes FROM polygons
        WHERE id = 207;

        DELETE FROM TABLE(vertexes);

        UPDATE polygons SET definition = vertexes
        WHERE id = 207;
```

此种形式的 DELETE 语句删除集合变量 vertexes 中的整个集合。您不可使用一个使用集合派生的表的 DELETE 语句中的 WHERE 子句。

在 UPDATE 语句之后，polygons 表包含空的集合，其中 id 列等于 207。

在 GBase 8s SQL 指南：语法 中描述 DELETE 语句的语法。

12.8.6 更新集合元素

您可通过访问游标内的集合来更新集合元素，就如同您选择或删除个别的元素一样。

如果您想要更新集合 SET{100, 200, 300, 500} 来将值 500 更改为 400，请从数据库将该 SET 检索至集合变量内，然后声明游标来在 SET 中的元素间移动，如下图所示。

图: 更新集合元素。

```
DEFINE s SET(INTEGER NOT NULL);
        DEFINE n INTEGER;

        SELECT numbers INTO s FROM orders
        WHERE order_num = 10;

        FOREACH cursor1 FOR
        SELECT * INTO n FROM TABLE(s)
        IF ( n == 500 ) THEN
        UPDATE TABLE(s)(x)
        SET x = 400 WHERE CURRENT OF cursor1;
        EXIT FOREACH;
        ELSE
        CONTINUE FOREACH;
        END IF;
        END FOREACH
```

UPDATE 语句使用集合变量 s 作为集合派生的表。要指定集合派生的表，请使用 TABLE 关键字。在 UPDATE 语句中跟在 (s) 之后的值 (x) 是您提供的列名称 derived column，因为 SET 子句需要它，即使集合派生的表没有列也需要。

请将集合派生的表视作有一行，且看起来与下列示例有些相似：

```
100        200        300        500
```

在此示例中，x 是包含值 500 的“列”的虚构的列名称。如果您正在更新内建的、opaque、distinct 或集合类型元素的集合，则仅指定派生的列。如果您正在更新 row 类型的集合，请使用字段名，而不是派生的列，如 更新 row 类型的集合 描述的那样。

使用变量更新集合

您还可使用存储在变量中的值，而不是文字值，来更新集合。

下图中的 SPL 过程使用类似于 图 1 展示的语句，除了此过程是使用变量，而不是文字值，来更新 manager 表的 direct_reports 列中的 SET。图 2 定义该 manager 表。

图: 使用变量更新集合。

```
CREATE PROCEDURE new_report(mgr VARCHAR(30),
        old VARCHAR(30), new VARCHAR(30) )

        DEFINE s SET (VARCHAR(30) NOT NULL);
        DEFINE n VARCHAR(30);

        SELECT direct_reports INTO s FROM manager
        WHERE mgr_name = mgr;

        FOREACH cursor1 FOR
        SELECT * INTO n FROM TABLE(s)
        IF ( n == old ) THEN
        UPDATE TABLE(s)(x)
        SET x = new WHERE CURRENT OF cursor1;
        EXIT FOREACH;
        ELSE
        CONTINUE FOREACH;
        END IF;
        END FOREACH

        UPDATE manager SET mgr_name = s
        WHERE mgr_name = mgr;

        END PROCEDURE;
```

嵌套在 FOREACH 循环中的 UPDATE 语句使用集合派生的表 s 和派生的列 x。如果 n 的当前值与 old 相同，则 UPDATE 语句将它更改为 new 的值。第二个 UPDATE 语句在 manager 表中存储新集合。

12.8.7 更新整个集合

如果您想要将集合中的所有元素更新为相同的值，或如果该集合仅包含一个值，则您无需使用游标。下图中的语句展示你可如何将集合检索至集合变量内，然后使用一个语句来更新它。

图: 检索并更新集合。

```
DEFINE s SET (INTEGER NOT NULL);
```

```
SELECT numbers INTO s FROM orders
WHERE order_num = 10;

UPDATE TABLE(s)(x) SET x = 0;

UPDATE orders SET numbers = s
WHERE order_num = 10;
```

此示例中的第一个 UPDATE 语句随同集合派生的表 s 使用名为 x 的派生的列，并将值 0 赋给集合中的所有元素。第二个 UPDATE 语句在数据库中存储新的集合。

更新 row 类型的集合

要更新 ROW 类型的集合，您可采用这些步骤：

1. 声明集合变量，其字段数据类型与该集合中的那些 ROW 类型相匹配。
2. 将集合变量的个别字段设置为 ROW 类型的正确数据值。
3. 对于每一 ROW 类型，请使用集合变量更新集合派生的表的全部行。

图 2 中的 manager 表有一名为 projects 的列，包含在下图展示其定义的 ROW 类型的 LIST。

图: ROW 类型的 LIST 定义。

```
projects    LIST( ROW( pro_name VARCHAR(15),
            pro_members SET(VARCHAR(20) NOT NULL) ) NOT NULL)
```

要访问 LIST 中的 ROW 类型，请声明游标，并将该 LIST 选择至集合变量内。然而，在您检索 projects 列中每一 ROW 类型值之后，您不可个别地更新 pro_name 或 pro_members 字段。相反，对于需要在集合中更新的每一 ROW 值，您必须将整个 ROW 替换为来自包括新字段值的集合变量的值，如下图所示。

图: 访问 LIST 中的 ROW 类型。

```
CREATE PROCEDURE update_pro( mgr VARCHAR(30),
            pro VARCHAR(15) )

            DEFINE p LIST(ROW(a VARCHAR(15), b SET(VARCHAR(20)
            NOT NULL) ) NOT NULL);
            DEFINE r ROW(p_name VARCHAR(15), p_member SET(VARCHAR(20)
NOT NULL) );
            LET r = ROW("project", "SET{'member'}");
```

```
        SELECT projects INTO p FROM manager
        WHERE mgr_name = mgr;

        FOREACH cursor1 FOR
        SELECT * INTO r FROM TABLE(p)
        IF (r.p_name == 'Zephyr') THEN
        LET r.p_name = pro;
        UPDATE TABLE(p)(x) SET x = r
        WHERE CURRENT OF cursor1;
        EXIT FOREACH;
        END IF;
        END FOREACH

        UPDATE manager SET projects = p
        WHERE mgr_name = mgr;

        END PROCEDURE;
```

在您可在 SPL 程序中使用 row 类型变量之前，您必须使用 LSET 语句或 SELECT INTO 语句来初始化该行变量。前图的 FOREACH 循环中嵌套的 UPDATE 语句将 row 类型的 pro_name 字段设置为变量 pro 中提供的值。

提示： 要更新 ROW 类型的 pro_members 字段中 SET 中的值，请声明游标，并使用带有派生的列的 UPDATE 语句，如 更新集合元素 说明的那样。

更新嵌套的集合

如果您想要更新集合的集合，则必须声明游标来访问外层的集合，然后声明嵌套的游标来访问内层的集合。

例如，假设 manager 表有一附加的列 scores，它包含一其元素类型为整数的 MULTISET 的 LIST，如下图所示。

图: 更新集合的集合。

```
scores          LIST(MULTISET(INT NOT NULL) NOT NULL);
```

要更新 MULTISET 中的值，请声明在 LIST 中每一值间移动的游标，以及在 MULTISET 中 每一值间移动的嵌套的游标，如下图所示。

图: 更新 MULTISET 中的值。

```
CREATE FUNCTION check_scores ( mgr VARCHAR(30) )
        SPECIFIC NAME nested;
```

```
RETURNING INT;

DEFINE l LIST( MULTISET( INT NOT NULL ) NOT NULL );
DEFINE m MULTISET( INT NOT NULL );
DEFINE n INT;
DEFINE c INT;

SELECT scores INTO l FROM manager
WHERE mgr_name = mgr;

FOREACH list_cursor FOR
SELECT * FROM TABLE(l) INTO m;

FOREACH set_cursor FOR
SELECT * FROM TABLE(m) INTO n;
IF (n == 0) THEN
DELETE FROM TABLE(m)
WHERE CURRENT OF set_cursor;
ENDIF;
END FOREACH;
LET c = CARDINALITY(m);
RETURN c WITH RESUME;
END FOREACH

END FUNCTION
WITH LISTING IN '/tmp/nested.out';
```

该 SPL 函数将 scores 列中的每一 MULTISET 选择至 l 内，然后将 MULTISET 中的每一值选择至 m 内。如果 m 中的值为 0，则函数从 MULTISET 删除它。在删除 0 的值之后，该函数统计每一 MULTISET 中剩余的元素数，并返回一整数。

提示： 由于此函数为 LIST 中每一 MULTISET 返回一值，因此，当您执行该函数时，您必须使用游标来括起 EXECUTE FUNCTION 语句。

12.8.8 插入至集合内

您可将值插入至集合内，而不声明游标。如果该集合为 SET 或 MULTISET，则将该值添加到集合，但不定义新元素的位置，因为该集合没有特定的顺序。如果该值为 LIST，则您可将新元素添加在 LIST 中特定的位置，或添加在 LIST 的末尾。

在 manager 表中，direct_reports 列包含 SET 类型的集合，且 projects 列包含 LIST。要将名称添加到 direct_reports 列中的 SET，请使用带有集合派生的表的 INSERT 语句，如下图所示。

图: 将值插入至集合。

```
CREATE PROCEDURE new_emp( emp VARCHAR(30), mgr VARCHAR(30) )

        DEFINE r SET(VARCHAR(30) NOT NULL);

        SELECT direct_reports INTO r FROM manager
        WHERE mgr_name = mgr;

        INSERT INTO TABLE (r) VALUES(emp);

        UPDATE manager SET direct_reports = r
        WHERE mgr_name = mgr;

        END PROCEDURE;
```

此 SPL 过程将员工姓名和管理者姓名作为参数。然后，该过程为用户已输入了的管理者选择 direct_reports 列中的集合，添加用户已输入了的员工名称，并以新集合更新 manager 表。

前图中的 INSERT 语句将用户提供的新的员工姓名插入至在集合变量 r 中包含的 SET 内。然后，UPDATE 语句将新集合存储在 manager 表中。

请注意 VALUES 子句的语法。将文字的数据和变量插入至集合变量内的语法规则如下：

- 使用 VALUES 关键字之后的圆括号来括起值的完整列表。
- 如果该集合为 SET、MULTISET 或 LIST，则请使用后跟方括号的类型构造函数来将要插入的值的列表括起来。此外，必须将集合值括在引号中。

  ```
  VALUES( "SET{ 1,4,8,9 }" )
  ```

- 如果该集合包含 row 类型，请使用后跟圆括号的 ROW 来将要插入的值的列表括起来：

  ```
  VALUES( ROW( 'Waters', 'voyager_project' ) )
  ```

- 如果该集合为嵌套的集合，则根据定义数据类型的方式，嵌套关键字、圆括号和方括号：

  ```
  VALUES( "SET{ ROW('Waters', 'voyager_project'),
                        ROW('Adams', 'horizon_project') }")
  ```

要获取关于将值插入至集合的信息，请参阅 修改数据。

插入至嵌套的集合内

如果您想要插入至嵌套的集合内，则 VALUES 子句的语法有变化。例如，假设您想要将值插入至 图 1 展示的 numbers 表的 twin_primes 列内。

对于 twin_primes 列，您可能想要将 SET 插入至 LIST，或将元素插入至内层的 SET。下列部分描述每一任务。

将集合插入至外层的集合内

将 SET 插入至 LIST 内，类似于将单个值插入至简单的集合内。

要将 SET 插入至 LIST 内，请声明集合变量来保存该 LIST，并将整个集合选择至它内。当您使用集合变量作为集合派生的表时，该 LIST 中的每一 SET 成为该表的一行。然后，您可将另一 SET 插入在该 LIST 的末尾或插入在指定的点。

例如，一个数值行的 twin_primes 列可能包含下列 LIST，如下图所示。

图: 样例 LIST。

```
LIST( SET{3,5}, SET{5,7}, SET{11,13} )
```

如果您将 LIST 视为集合派生的表，则它看上去可能像这样。

图: 将 LIST 视为集合派生的表。

```
{3,5}
                {5,7}
                {11,13}
```

您可能想要插入值 "SET{17,19}" 作为 LIST 中的第二项。下图中的语句展示如何执行。

图: 将值插入到 LIST 中。

```
CREATE PROCEDURE add_set()

        DEFINE l_var LIST( SET( INTEGER NOT NULL ) NOT NULL );

        SELECT twin_primes INTO l_var FROM numbers
        WHERE id = 100;

        INSERT AT 2 INTO TABLE (l_var) VALUES( "SET{17,19}" );

        UPDATE numbers SET twin_primes = l
        WHERE id = 100;

        END PROCEDURE;
```

在 INSERT 语句中，VALUES 子句将值 SET {17,19} 插入在 LIST 的第二个位置。现在，该 LIST 看上去像下图这样。

图: LIST 项。

```
{3,5}
        {17,19}
        {5,7}
        {11,13}
```

通过将 SET 作为参数传到 SPL 例程，您可执行相同的插入，如下图所示。

图: 将 SET 作为参数传到 SPL 例程。

```
CREATE PROCEDURE add_set( set_var SET(INTEGER NOT NULL),
        row_id INTEGER );

        DEFINE list_var LIST( SET(INTEGER NOT NULL) NOT NULL );
        DEFINE n SMALLINT;

        SELECT CARDINALITY(twin_primes) INTO n FROM numbers
        WHERE id = row_id;

        LET n = n + 1;

        SELECT twin_primes INTO list_var FROM numbers
        WHERE id = row_id;

        INSERT AT n INTO TABLE( list_var ) VALUES( set_var );

        UPDATE numbers SET twin_primes = list_var
        WHERE id = row_id;

        END PROCEDURE;
```

在 add_set() 中，用户提供 SET 来添加到 LIST，以及标识将 SET 插入其中的那行的 id 的 INTEGER 值。

将值插入至内层的集合

在 SPL 例程中，您还可将值插入至嵌套的集合的内层集合。通常，要访问嵌套的集合的内层集合并将值插入到它，请执行下列步骤：

1. 声明集合变量来在表的一行中保存整个集合。

2. 声明元素变量来保存该外层的集合的一个元素。元素变量本身是集合变量。
3. 将整个集合从表的一行选择至集合变量。
4. 声明游标，以便您可在外层的集合的元素间移动。
5. 一次将一个元素选择至元素变量内。
6. 请使用分支或循环来定位您想要更新的内层集合。
7. 将新值插入至内层的集合内。
8. 关闭游标。
9. 以新的集合更新数据库表。

作为示例，您可在 numbers 的 twin_primes 列上使用此过程。 例如，假设 twin_primes 包含下图所示的值，且您想要将值 18 插入至 LIST 的最后的 SET 中。

图: twin_primes 列表。

```
LIST( SET( {3,5}, {5,7}, {11,13}, {17,19} ) )
```

下图展示插入该值的过程的开始。

图: 插入值的过程。

```
CREATE PROCEDURE add_int()

        DEFINE list_var LIST( SET( INTEGER NOT NULL ) NOT NULL );
        DEFINE set_var SET( INTEGER NOT NULL );

        SELECT twin_primes INTO list_var FROM numbers
        WHERE id = 100;
```

至此，attaint 过程已执行了步骤 1、2 和 3。第一个 DEFINE 语句声明保存在一个数值行中的整个集合的集合变量。

第二个 DEFINE 语句声明保存该集合的元素的元素变量。在此情况下，元素变量本身是集合变量，因为它保存 SET。SELECT 语句将整个集合从一行选择至集合变量 list_var 内。

下图展示如何声明游标，以便于您可在外层的集合的元素间移动。

图: 声明游标来在外层的集合的元素间移动。

```
FOREACH list_cursor FOR
            SELECT * INTO set_var FROM TABLE( list_var);

            FOREACH element_cursor FOR
```

12.9 执行例程

您可以下列任一方式执行 SPL 例程或外部例程：

- 使用从 DB-Access 执行的单独的 EXECUTE PROCEDURE 或 EXECUTE FUNCTION 语句
- 从另一 SPL 例程或外部例程显式地调用例程
- 在 SQL 语句中使用带有表达式的例程名称

执行例程的附加的机制仅支持 sysdbopen 和 sysdbclose 过程，DBA 可定义这些过程。当用户通过 CONNECT 或 DATABASE 语句连接到数据库时，如果 sysdbopen 过程的所有者与数据库中存在的用户的登录标识符相匹配，则自动地执行那个例程。如果没有 sysdbopen 例程的所有者与该用户的登录标识符相匹配，但存在 PUBLIC.sysdbopen 例程，则执行那个例程。这种自动的调用使得 DBA 能够在连接时刻为用户定制会话环境。当用户从数据库断开连接时，类似地调用 sysdbclose 例程。（要获取关于这些会话配置例程的更多信息，请参阅《GBase 8s SQL 指南：语法》和 GBase 8s 管理员指南。）

外部例程是以 C 或某种其他外部语言编写的例程。

12.9.1 EXECUTE 语句

您可使用 EXECUTE PROCEDURE 或 EXECUTE FUNCTION 来执行 SPL 例程或外部例程。通常，最好将 EXECUTE PROCEDURE 用于过程，将 EXECUTE FUNCTION 用于函数。

提示： 为了向后兼容， EXECUTE PROCEDURE 语句允许您使用 SPL 函数名称和 INTO 子句来返回值。然而，推荐您仅将 EXECUTE PROCEDURE 用于过程，仅将 EXECUTE FUNCTION 用于函数。

您可从 DB-Access 或从 SPL 例程或外部例程内，发出 EXECUTE PROCEDURE 和 EXECUTE FUNCTION 语句作为独立的语句。如果在数据库内该例程名称是唯一的，且如果它不需要参数，则您可通过在 EXECUTE PROCEDURE 之后只输入它的名称和圆括号来执行它，如下图所示。

图: 执行过程。

```
EXECUTE PROCEDURE update_orders();
```

由于过程不返回任何值，因此，当您以 EXECUTE 语句调用过程时，从不出现 INTO 子句。

如果例程期望参数，则您必须在圆括号内输入参数值，如下图所示。

图: 执行带有参数的过程。

```
EXECUTE FUNCTION scale_rectangles(107, 1.9)
```

```
        INTO new;
```

该语句执行函数。由于函数返回值，因此，EXECUTE FUNCTION 使用 INTO 子句，指定存储返回值的变量。当您使用 EXECUTE 语句来执行函数时，始终出现 INTO 子句。

如果数据库有多个同名的过程或函数，则 GBase 8s 基于参数的数据类型来定位到正确的函数。例如，前图中的语句提供 INTEGER 和 REAL 值作为参数，因此，如果您的数据库包含名为 scale_rectangles() 的多个例程，在数据库服务器仅执行接受 INTEGER 和 REAL 数据类型的scale_rectangles() 函数。

SPL 例程的参数列表始终有参数名称及数据类型。当您执行例程时，参数名称是可选的。然而，如果您通过名称（而不是只通过值）来将参数传到 EXECUTE PROCEDURE 或 EXECUTE FUNCTION，如下图所示，则 GBase 8s 仅逐个例程地解析名称和参数，该过程称为部分的例程解析。

图: 执行通过名称传递参数的例程。

```
EXECUTE FUNCTION scale_rectangles( rectid = 107,
        scale = 1.9 ) INTO new_rectangle;
```

您还可通过将限定的例程名称添加到语句来执行存储在另一数据库服务器上的 SPL 例程；即，database@dbserver:owner_name.routine_name 形式的名称，如下图所示。

图: 执行存储在另一数据库服务器上的 SPL 例程。

```
EXECUTE PROCEDURE gbasedbt@davinci:bsmith.update_orders();
```

当您远程地执行例程时，限定的例程名称中的 owner_name 是可选的。

12.9.2 CALL 语句

您可使用 CALL 语句，从 SPL 例程调用 SPL 例程或外部例程。CALL 可执行过程，也可执行函数。如果您使用 CALL 来执行函数，则请添加 RETURNING 子句和将要接收该函数返回的值的 SPL 变量的名称。

例如，假设您想要 scale_rectangles 函数调用计算矩形面积的外部函数，然后返回带有矩形描述的面积，如下图所示。

图: 调用外部函数。

```
CREATE FUNCTION scale_rectangles( rectid INTEGER,
        scale REAL )
        RETURNING rectangle_t, REAL;

        DEFINE rectv rectangle_t;
        DEFINE a REAL;
        SELECT rect INTO rectv
        FROM rectangles WHERE id = rectid;
```

```
        IF ( rectv IS NULL ) THEN
        LET rectv.start = (0.0,0.0);
        LET rectv.length = 1.0;
        LET rectv.width = 1.0;
        LET a = 1.0;
        RETURN rectv, a;
        ELSE
        LET rectv.length = scale * rectv.length;
        LET rectv.width = scale * rectv.width;
        CALL area(rectv.length, rectv.width) RETURNING a;
        RETURN rectv, a;
        END IF;

        END FUNCTION;
```

该 SPL 函数使用执行外部函数 area() 的 CALL 语句。返回的值 area() 保存在 a 中，并通过 RETURN 语句返回到调用例程。

在此示例中，area() 是外部函数，但您可以同样的方式将 CALL 用于 SPL 函数。

12.9.3 执行表达式中的例程

正如内建的函数那样，您可通过在 SQL 和 SPL 语句中的表达式，使用 SPL 例程来执行 SPL 例程（以及来自 SPL 例程的外部例程）。表达式中使用的例程通常为函数，因为它将值返回至语句的剩余部分。

例如，您可能通过将返回值分配给变量的 LET 语句来执行函数。下图中的语句执行相同的任务。它们执行 SPL 例程内的外部函数，并将返回值分配给变量 a。

图: 执行 SPL 例程内的外部函数。

```
LET a = area( rectv.length, rectv.width );


        CALL area( rectv.length, rectv.width ) RETURNING a;
        -- 这些语句是等同的
```

您还可从 SQL 语句执行 SPL 例程，如下图所示。假设您编写 SPL 函数 increase_by_pct，对给定的价格增加给定的百分比。在您编写 SPL 例程之后，在任何其他 SPL 例程中都可使用它。

图: 从 SQL 语句执行 SPL 例程。

```
CREATE FUNCTION raise_price ( num INT )
        RETURNING DECIMAL;
```

```
        DEFINE p DECIMAL;

        SELECT increase_by_pct(price, 20) INTO p
        FROM inventory WHERE prod_num = num;

        RETURN p;

        END FUNCTION;
```

该示例选择 inventory 的指定的行的 price 列，并使用该值作为 SPL 函数 increase_by_pct 的参数。然后，该函数返回新的 price 值，在变量中增加 20%。

12.9.4 使用 RETURN 语句执行外部函数

您可使用 RETURN 语句来从 SPL 例程内执行任何外部函数。下图展示在 SPL 程序的 RETURN 语句中使用的外部函数。

图: 从 SPL 例程内执行外部函数的 RETURN 语句。

```
CREATE FUNCTION c_func() RETURNS int
        LANGUAGE C;

        CREATE FUNCTION spl_func() RETURNS INT;
        RETURN(c_func());
        END FUNCTION;

        EXECUTE FUNCTION spl_func();
```

当您执行 spl_func() 函数时，调用 c_func() 函数，且 SPL 函数返回外部函数返回的值。

12.9.5 从 SPL 例程执行游标函数

游标函数是返回一行或多行数据的用户定义的函数，因此需要游标来执行。游标函数可为系列函数之一：

- 其 RETURN 语句包括 WITH RESUME 的 SPL 函数
- 定义作为迭代函数的外部函数

游标函数的行为与 SPL 函数或外部函数都一样。然而，SPL 游标函数每迭代可返回多个值，而外部游标函数（迭代函数）每迭代仅可返回一个值。

要从 SPL 例程执行游标函数，您必须在 SPL 例程的 FOREACH 循环中包括该函数。下列示例展示在 FOREACH 循环中执行游标函数的不同方式：

```
FOREACH SELECT cur_func1(col_name) INTO spl_var FROM tab1
        INSERT INTO tab2 VALUES (spl_var);
        END FOREACH

        FOREACH EXECUTE FUNCTION cur_func2() INTO spl_var
        INSERT INTO tab2 VALUES (spl_var);
        END FOREACH
```

12.9.6 动态的例程名称规范

通过在调用例程内构建被调用的例程的名称，动态的例程名称规范允许您从另一 SPL 例程执行 SPL 例程。动态的例程名称规范简化您编写调用另一 SPL 例程的 SPL 例程的方式，直到运行时才能知道另一例程的名称。数据库服务器允许您在 EXECUTE PROCEDURE 或 EXECUTE FUNCTION 语句中指定 SPL 变量，而不是 SPL 例程的显式的名称。

在下图中，SPL 过程 company_proc 更新大型的公司销售表，然后，分配名为 salesperson_proc 的 SPL 变量来保存更新另一较小表的动态地创建的 SPL 过程的名称，这个较小的表包含个别销售人员的每月销售情况。

图: 动态的例程名称规范。

```
CREATE PROCEDURE company_proc ( no_of_items INT,
        itm_quantity SMALLINT, sale_amount MONEY,
        customer VARCHAR(50), sales_person VARCHAR(30) )

        DEFINE salesperson_proc VARCHAR(60);

        -- 更新公司表
        INSERT INTO company_tbl VALUES (no_of_items, itm_quantity,
        sale_amount, customer, sales_person);

        -- 生成变量 salesperson_proc 的过程名称
        LET salesperson_proc = sales_person || "." || "tbl" ||
        current_month || "_" || current_year || "_proc" ;

        -- 执行 salesperson_proc 变量指定的
        -- SPL 过程
```

```
        EXECUTE PROCEDURE salesperson_proc (no_of_items,
        itm_quantity, sale_amount, customer)
        END PROCEDURE;
```

在示例中，过程 company_proc 接受五个参数，并将它们插入至 company_tbl 内。然后，LET 语句使用不同的值和连接运算符 || 来产生要执行的另一 SPL 过程的名称。在 LET 语句中：

sales_person

传给 company_proc 过程的参数。

current_month

系统日期中的当前月份。

current_year

系统日期中的当前年份。

因此，如果名为 Bill 的销售人员在 1988 年 7 月完成一笔销售，
则 company_proc 在 company_tbl 中插入一记录，并执行 SPL 过程bill.tbl07_1998_proc，更新包含个别销售人员的每月销售情况的较小的表。

动态例程名称规范的规则

您必须定义保存动态地执行的 SPL 例程的名称的 SPL 变量为 CHAR、VARCHAR、NCHAR 或 NVARCHAR 类型。您还必须为 SPL 变量提供一个有效的且非 NULL 的名称。

在可执行动态的例程名称规范标识的 SPL 例程之前，该例程必须存在。如果你将有效的 SPL 例程的名称分配给该 SPL 变量，则 EXECUTE PROCEDURE 或 EXECUTE FUNCTION 语句执行在该变量中包含其名称的例程，即使存在同名的内建的函数。

在 EXECUTE PROCEDURE 或 EXECUTE FUNCTION 语句中，您不可使用两个 SPL 变量来创建形式为 owner.routine_name 的变量名称。然而，您可使用包含完全限定的例程名称的 SPL 变量，例如，bill.proc1。下图同时展示这两种情况。

图: 包含完全限定的例程名称的 SPL 变量。

```
EXECUTE PROCEDURE owner_variable.proc_variable;
        -- 不允许这样


        LET proc1 = bill.proc1;
        EXECUTE PROCEDURE proc1; -- 允许这样
```

12.10 对例程的权限

权限将可创建例程的用户与可执行例程的用户区分开来。有些权限表现为其他权限的一部分。例如，DBA 权限包括创建例程、执行例程的权限，以及将这些权限授予其他用户的权限。

12.10.1 注册例程的权限

要在数据库中注册例程，被授权的用户将 SPL 命令包含在 CREATE FUNCTION 或 CREATE PROCEDURE 语句中。数据库服务器存储内部注册了的 SPL 例程。下列用户具有在数据库中注册新的例程的资格：

- 有 DBA 权限的任何用户可在 CREATE 语句中，使用或不使用 DBA 关键字来注册例程。

  要了解 DBA 关键字的说明，请参阅 执行例程的 DBA 权限。

- 没有 DBA 权限的用户需要 Resource 权限来注册 SPL 例程。该例程的创建者是所有者。

  没有 DBA 权限的用户不可使用 DBA 关键字来注册例程。

  DBA 必须给其他需要创建例程的用户授予 Resource 权限。DBA 还可撤销 Resource 权限，防止用户创建更多的例程。

- 除了对在其中注册 UDR 的数据库保存持有 DBA 权限或 Resource 权限之外，创建 UDR 的用户还必须持有对以其编写 UDR 的编程语言的 Usage 权限。这些 SQL 可为特定的编程语言授予语言级别的 Usage 权限：

  - GRANT USAGE ON LANGUAGE C
  - GRANT USAGE ON LANGUAGE JAVA
  - GRANT USAGE ON LANGUAGE SPL

  除了个别的用户之外，这些权限的被授予者还可为用户定义的角色，或 PUBLIC 组。在将语言级别 Usage 权限授予角色之后，持有那个角色的任何用户都可通过使用 SQL 的 SET ROLE 语句使得该角色的所有访问权限能够指定那个角色作为当前的角色。

对于以 C 语言或 Java™ 语言编写的外部例程，如果启用 IFX_EXTEND_ROLE 配置参数，则仅 DBSA 已授予其 EXTERNAL 角色的用户可注册、删除或修改外部的 UDR 或 DataBlade 模块。在缺省情况下，启用此参数。通过将 IFX_EXTEND_ROLE 配置参数设置为 OFF 或设置为 0，DBSA 可禁用对持有 DataBlade 模块或外部 UDR 的 DDL 操作 EXTEND 角色的要求。然而，此安全特性对 SPL 例程不起作用。

总之，持有以上标识的数据库级别和语言级别自主访问控制权限（且还持有 EXTEND 角色，如果启用 IFX_EXTEND_ROLE 且该 UDR 为外部例程的话）的用户，可在下列 SQL 语句中引用 UDR：

- DBA 或用户可以 CREATE FUNCTION、CREATE FUNCTION FROM、CREATE PROCEDURE、CREATE PROCEDURE FROM、CREATE ROUTINE 或 CREATE ROUTINE FROM 语句来注册新的 UDR。
- DBA 或现有 UDR 的所有者可以 DROP FUNCTION、DROP PROCEDURE 或 DROP ROUTINE 语句来取消那个 UDR 的注册。
- DBA 或现有 UDR 的所有者可以 ALTER FUNCTION、ALTER PROCEDURE 或 ALTER ROUTINE 语句来修改那个 UDR 的定义。

12.10.2 执行例程的权限

Execute 权限使得用户能够调用例程。通过 EXECUTE 或 CALL 语句，或通过使用表达式中的函数可能调用例程。下列用户拥有缺省的 Execute 权限，这使得他们能够调用例程：

- 在缺省情况下，任何具有 DBA 权限的用户都可执行数据库中的任何例程。
- 如果以限定的 CREATE DBA FUNCTION 或 CREATE DBA PROCEDURE 语句注册该例程，则仅拥有 DBA 权限的用户对那个例程有缺省的 Execute 权限。
- 如果数据库不符合 ANSI，则用户 public（任何拥有 Connect 数据库权限的用户）自动地拥有对例程的 Execute 权限，未以 DBA 关键字注册该例程。
- 在符合 ANSI 的数据库中，过程所有者和任何拥有 DBA 权限的用户都可执行该例程，而无需收到附加的权限。

授予和撤销 Execute 权限

例程有下列 GRANT 和 REVOKE 要求：

- DBA 可将 Execute 权限授予数据库中的任何例程，也可撤销它。
- 例程的创建者可授予或取消对那个特定的例程的 Execute 权限。通过包括带有 GRANT EXECUTE ON 语句的 AS grantor 子句，创建者丧失授予或撤销的能力。
- 如果所有者在 GRANT EXECUTE ON 语句中应用了 WITH GRANT 关键字，则另一用户可授予 Execute 权限。

对于下列条件，DBA 或例程所有者必须显式地将 Execute 权限授予非 DBA 用户：

- 以 DBA 关键字子句注册了的例程
- 在符合 ANSI 的数据库中的例程
- 不符合 ANSI 的数据库中的例程，但将 NODEFDAC 环境变量设置为 yes。

即使数据库服务器缺省地将权限授予 public，所有者也可限制对例程的 Execute 权限。为此，请发出 REVOKE EXECUTE ON PUBLIC 语句。DBA 和所有者仍可执行该例程，且如果使用的话，则可将 Execute 权限授予特定的用户。

使用 COMMUTATOR 和 NEGATOR 函数的 Execute 权限

重要： 如果您显式地授予对 SPL 函数的 Execute 权限，其为 UDR 的换向函数或否定函数，则在被授予者可使用任意函数之前，您还必须授予对换向函数或否定函数的那种权限。您不可随同 SPL 过程指定 COMMUTATOR 或 NEGATOR 修饰符。

下列示例演示对于函数的限制授权，以及将它的否定函数限定为一组用户。假设您创建下列否定函数对：

```
CREATE FUNCTION greater(y PERCENT, z PERCENT)
RETURNS BOOLEAN
NEGATOR= less(y PERCENT, z PERCENT);
. . .
CREATE FUNCTION less(y PERCENT, z PERCENT)
RETURNS BOOLEAN
NEGATOR= greater(y PERCENT, z PERCENT);
```

在缺省情况下，任何用户都可执行该函数和否定函数。下列函数仅允许 accounting 执行这些函数：

```
REVOKE EXECUTE ON FUNCTION greater FROM PUBLIC;
REVOKE EXECUTE ON FUNCTION less FROM PUBLIC;
GRANT accounting TO mary, jim, ted;
GRANT EXECUTE ON FUNCTION greater TO accounting;
GRANT EXECUTE ON FUNCTION less TO accounting;
```

用户可能接收附带 WITH GRANT OPTION 授权的 Execute 权限来将 Execute 权限授予其他用户。如果用户失去对例程的 Execute 权限，则还从通过那个用户授予了 Execute 权限的那些用户撤销 Execute 权限。

要获取更多信息，请参阅《GBase 8s SQL 指南：语法》 中的 GRANT 和 REVOKE 语句描述。

12.10.3 对与例程相关联的对象的权限

数据库服务器检查是否存在任何被引用的对象，并验证调用该例程的用户是否拥有访问被引用的对象的必要权限。

由例程引用的对象可包括：

- 表和列
- 序列对象
- 用户定义的数据类型
- 由该例程执行的其他例程

当例程运行时，定义有效的权限为下列的联合：

- 运行该例程的用户的权限，
- 带有 GRANT 选项的所有者的权限。

在缺省情况下，数据库管理员拥有数据库中带有 GRANT 选项的所有权限。因此，执行由数据库管理员拥有的例程的用户可从给定的数据库中所有表进行选择。

GRANT EXECUTE ON 语句提供给被授予者任何表级别权限，授予者从包含 WITH GRANT 关键字的 GRANT 语句收到这些权限。

该例程的所有者，不是运行该例程的用户，拥有在执行该例程过程中创建的未限定的对象。例如，假设用户 howie 注册创建两个表的 SPL 例程，使用下列 SPL 例程：

```
CREATE PROCEDURE promo()
        . . .
        CREATE TABLE newcatalog
        (
        catlog_num INTEGER
        cat_advert VARCHAR(255, 65)
        cat_picture BLOB
        ) ;
        CREATE TABLE dawn.mailers
        (
        cust_num INTEGER
        interested_in SET(catlog_num INTEGER)
        );
        END PROCEDURE;
```

用户 julia 运行该例程，创建表 newcatalog。由于没有所有者名称来限定表名称 newcatalog，因此，例程所有者（howie）拥有newcatalog。相比之下，限定的名称 dawn.maillist 标识 dawn 作为 maillist 的所有者。

12.10.4 执行例程的 DBA 权限

如果 DBA 使用 DBA 关键字创建例程，则数据库服务器自动地仅将 Execute 权限授予有 DBA 权限的其他用户。然而，DBA 可显式地将 DBA 例程上的 Execute 权限授予没有 DBA 权限的用户。

当用户执行以 DBA 关键字注册了的例程时，该用户假设在例程持续期间持有 DBA 权限。如果没有 DBA 权限的用户运行 DBA 例程，则数据库服务器隐式地将临时的 DBA 权限授予调用者。在退出 DBA 例程之前，数据库服务器隐式地撤销该临时的 DBA 权限。

执行 DBA 例程的用户拥有在运行该 DBA 例程期间创建的对象，除非例程中的语句显式地命名其他用户作为所有者。 例如，假设 tony 以 DBA 关键字注册 promo() 例程，如下：

```
CREATE DBA PROCEDURE promo()
                . . .
                CREATE TABLE catalog
                . . .
                CREATE TABLE libby.mailers
                . . .
                END PROCEDURE;
```

虽然 tony 拥有该例程，但如果 marty 运行它，那么 marty 拥有 catalog 表，但由于用户 libby 的名称限定 libby.mailers 表名称，使得她成为该表的所有者，因此它拥有该表。

被调用的例程未继承 DBA 权限。如果 DBA 例程执行未以 DBA 关键字创建了的例程，则 DBA 权限不影响被调用的例程。

如果未以 DBA 关键字注册的例程调用 DBA 例程，则调用者对于被调用的 DBA 例程必须有 Execute 权限。该 DBA 例程内的语句执行如同任何 DBA 例程内的语句一样。

下列示例展示当 DBA 与非 DBA 例程相互作用时发生的情况。假设过程 dbspc_cleanup() 执行另一过程 clust_catalog()。还假设 clust_catalog()创建索引，且 clust_catalog() 的 SPL 源代码包括下列语句：

```
CREATE CLUSTER INDEX c_clust_ix ON catalog (catalog_num);
```

DBA 过程 dbspc_cleanup() 以下列语句调用其他例程：

```
EXECUTE PROCEDURE clust_catalog(catalog);
```

假设 tony 注册了 dbspc_cleanup() 作为 DBA 过程，而未以 DBA 关键字注册 clust_catalog()，如下列语句所示：

```
CREATE DBA PROCEDURE dbspc_cleanup(loc CHAR)
                CREATE PROCEDURE clust_catalog(catalog CHAR)
                GRANT EXECUTE ON dbspc_cleanup(CHAR) to marty;
```

假设用户 marty 运行 dbspc_cleanup()。由于通过非 DBA 例程创建索引 c_clust_ix，因此，同时拥有两个例程的 tony 也拥有 c_clust_ix。相对地，如果 clust_catalog() 为 DBA 过程，则 marty 会拥有索引 c_clust_ix，如下列注册和授权语句所示：

```
CREATE PROCEDURE dbspc_cleanup(loc CHAR);
                CREATE DBA PROCEDURE clust_catalog(catalog CHAR);
                GRANT EXECUTE ON clust_catalog(CHAR) to marty;
```

请注意，dbspc_cleanup() 无需 DBA 过程来调用 DBA 过程。

12.11 在 SPL 例程中查找错误

当您以 DB-Access 使用 CREATE PROCEDURE 或 CREATE FUNCTION 来编写 SPL 例程时，如果在例程体中发生语法错误，则当您从菜单选择 Run时，该语句失败。

如果您正在 DB-Access 中创建例程，当您从菜单选择 Modify 选项时，光标移动至包含语法错误的那一行。您可再次选择 Run 和 Modify 来检查后续的行。

12.11.1 编译时刻警告

如果数据库服务器检测到潜在的问题，但该 SPL 例程的语法是正确的，则数据库服务器生成警告，并将它放置在列表文件中。您可在执行该例程前，检测此文件来检查潜在的问题。

在 CREATE PROCEDURE 或 CREATE FUNCTION 语句的 WITH LISTING IN 子句中指定文件名称和列表文件的路径名称。要获取关于如何指定列表文件的路径名称的信息，请参阅 指定 DOCUMENT 子句。

如果您正在网络上工作，则在该数据库驻留的系统上创建列表文件。如果您为该文件提供绝对的路径名称和文件名称，则在您指定的位置创建该文件。

对于 UNIX™，如果您为列表文件提供相对的路径名称，则在数据库驻留的计算机上您的 home 目录中创建该文件。（如果您没有 home 目录，则在 root 目录中创建文件。）

对于 Windows™，如果您为列表文件提供相对的路径名称，则如果该数据库在本地计算机上，则缺省的目录为您的当前工作目录。否则，缺省的目录为 %GBASEDBTDIR%\bin。

在您创建例程之后，您可查看在 WITH LISTING IN 子句中指定的文件，来查看它包含的警告。

12.11.2 生成例程的文本

在您创建 SPL 例程之后，将它存储在 sysprocbody 系统目录表中。sysprocbody 系统目录表包含可执行的例程，以及它的文本。

要检索例程的文本，请从 sysprocbody 系统目录表选择 data 列。文本条目的 datakey 列有代码 T。

下图中的 SELECT 语句读取 SPL 例程 read_address 的文本。

图: 读取 SPL 例程的文本的 SELECT 语句。

```
SELECT data FROM gbasedbt.sysprocbody
                WHERE datakey = 'T'                    -- 查找文本行
                AND procid =
                ( SELECT procid
                FROM gbasedbt.sysprocedures
                WHERE gbasedbt.sysprocedures.procname =
```

```
            'read_address' )
```

12.12 调试 SPL 例程

在您成功地创建并运行 SPL 例程之后，您可遇到逻辑错误。如果例程有逻辑错误，则请使用 TRACE 语句来帮助查找它们。您可跟踪下列项的值：

- 变量
- 参数
- 返回值
- SQL 错误代码
- ISAM 错误代码

要生成被跟踪的值的列表，首先请使用 SQL 语句 SET DEBUG FILE 来命名包含被跟踪的输出的文件。当您创建 SPL 例程时，请包括 TRACE 语句。

下列方法指定 TRACE 输出的形式。

语句

操作

TRACE ON

跟踪 SQL 语句之外的所有语句。在使用变量之前，打印它们的内容。跟踪例程调用和返回的值。

TRACE PROCEDURE

仅跟踪例程调用和返回的值。

TRACE expression

打印文字或表达式。如果必要，在将表达式的值发送至文件之前，计算它。

下图演示您可如何使用 TRACE 语句来监视 SPL 函数执行的方式。

图: TRACE 语句。

```
CREATE FUNCTION read_many   (lastname CHAR(15))
            RETURNING CHAR(15), CHAR(15), CHAR(20), CHAR(15),
            CHAR(2), CHAR(5);

            DEFINE p_lname,p_fname, p_city CHAR(15);
            DEFINE p_add CHAR(20);
            DEFINE p_state CHAR(2);
            DEFINE p_zip CHAR(5);
```

```
DEFINE lcount, i INT;

LET lcount = 1;

TRACE ON;          -- 从此开始跟踪每个表达式
TRACE 'Foreach starts';   -- 跟踪带有文字的语句

FOREACH
SELECT fname, lname, address1, city, state, zipcode
INTO p_fname, p_lname, p_add, p_city, p_state, p_zip

FROM customer
WHERE lname = lastname
RETURN p_fname, p_lname, p_add, p_city, p_state, p_zip
WITH RESUME;
LET lcount = lcount + 1;     -- 对返回的地址计数
END FOREACH

TRACE 'Loop starts';          -- 另一文字
FOR i IN (1 TO 5)
BEGIN
RETURN i , i+1, i*i, i/i, i-1,i WITH RESUME;
END
END FOR;

END FUNCTION;
```

使用 TRACE ON 语句，在您每次执行被跟踪的例程时，都将条目添加到您在 SET DEBUG FILE 语句中指定了的文件。要查看调试条目，请使用任何文本编辑器来查看该输出文件。

下列列表包含前面的示例中函数生成的一些输出。每一被跟踪的语句之后是对它的内容的解释。

语句

操作

TRACE ON

回送 TRACE ON 语句。

TRACE Foreach starts

在此情况下，跟踪表达式，文字字符串 Foreach 开始。

start select cursor

提供打开游标来处理 FOREACH 循环的通知。

select cursor iteration

提供选择游标的每一迭代的开始的通知。

expression: (+lcount, 1)

对遇到的表达式 (lcount+1) 求值为 2。

let lcount = 2

以该值回送每一 LET 语句。

12.13 异常处理

您可使用 ON EXCEPTION 语句来捕获数据库服务器返回给您的 SPL 例程的任何异常（或错误），或该例程产生的任何异常。RAISE EXCEPTION 语句允许您生成 SPL 例程内的异常。

在 SPL 例程中，您不可使用异常处理来处理下列情况：

- 成功（返回了行）
- 成功（未返回行）

12.13.1 错误捕获与恢复

ON EXCEPTION 语句提供捕获任何错误的机制。

要捕获错误，请将一组语句包含在以 BEGIN 与 END 标记的语句块中，并在该语句块的开头添加 ON EXCEPTION IN 语句。如果在跟在 ON EXCEPTION 语句之后的块中发生错误，则您可采取恢复措施。

下图展示语句块内的 ON EXCEPTION 语句。

图: 捕获错误。

```
BEGIN
        DEFINE c INT;
        ON EXCEPTION IN
        (
        -206, --  表不存在
        -217   --  列不存在
```

```
) SET err_num

IF err_num = -206 THEN
CREATE TABLE t (c INT);
INSERT INTO t VALUES (10);
-- 在插入语句之后继续
ELSE
ALTER TABLE t ADD(d INT);
LET c = (SELECT d FROM t);
-- 在选择语句之后继续
END IF
END EXCEPTION WITH RESUME

INSERT INTO t VALUES (10);   -- 如果 t 不存在，则失败

LET c = (SELECT d FROM t);   -- 如果 d 不存在，则失败
END
```

当发生错误时，SPL 解释器搜索捕获该错误的最内层 ON EXCEPTION 声明。捕获错误之后的第一个操作是重置该错误。当完成错误操作代码的执行时，且如果引起错误的 ON EXCEPTION 声明包括了 WITH RESUME 关键字，则以跟在产生了该错误的语句之后的语句自动地恢复执行。如果 ON EXCEPTION 声明未包括 WITH RESUME 关键字，则执行完全地退出当前的块。

12.13.2 ON EXCEPTION 语句的控制作用域

ON EXCEPTION 语句的作用域从紧跟在 ON EXCEPTION 语句之后的语句扩展，并结束于在其中发出 ON EXCEPTION 语句的语句块的末尾。如果 SPL 例程未包括显式的语句块，则作用域为该例程中所有后续的语句。

对于在 IN 子句中指定的异常（或对于所有异常，如果未指定 IN 子句的话），ON EXCEPTION 语句的作用域包括同一语句块内跟在 ON EXCEPTION 语句之后的所有语句。如果在那个块内嵌套其他语句块，则该作用域还包括跟在 ON EXCEPTION 语句之后的嵌套的语句块中的所有语句，以及在那些嵌套的块内嵌套的语句块中的任何语句。

下列伪代码展示在例程内该例程为有效的位置。 即，如果错误 201 发生在任何指示了的块中，则发生标号为 a201 的操作。

图: ON EXCEPTION 语句的控制作用域。

```
CREATE PROCEDURE scope()
```

```
DEFINE i INT;
. . .
BEGIN                -- 开始语句块 A
. . .
ON EXCEPTION IN (201)
-- 执行操作 a201
END EXCEPTION
BEGIN                -- 嵌套的语句块 aa
-- 执行操作，a201 在此有效
END
BEGIN                -- 嵌套的语句块 bb
-- 执行操作，a201 在此有效
END
WHILE i < 10
-- 执行某操作，a201 在此有效
END WHILE

END      -- 语句块 A 的末尾
BEGIN                -- 开始语句块 B
-- 执行某操作
-- a201 在此 NOT 有效
END
END PROCEDURE;
```

12.13.3 用户生成的异常

您可使用 RAISE EXCEPTION 语句生成您自己的错误，如下图所示。

图: RAISE EXCEPTION 语句。

```
BEGIN
        ON EXCEPTION SET esql, eisam     -- 捕获所有错误
        IF esql = -206 THEN              -- 未找到表
        -- 某种恢复
        ELSE
        RAISE exception esql, eisam;   -- 放过该错误
        END IF
        END EXCEPTION
```

```
        -- 执行某操作
        END
```

在该示例中，ON EXCEPTION 语句使用两个变量 esql 和 eisam，来保存数据库服务器返回的错误编号。如果发生错误且如果 SQL 错误编号为 -206，则执行 IF 子句。如果捕获任何其他 SQL 错误，则将它从此 BEGINEND 块传至前面的示例的最后 BEGINEND 块。

模拟 SQL 错误

您可产生错误来模拟 SQL 错误，如下图所示。如果用户为 pault， 则 SPL 运行如同那个用户没有 update 权限一样，即使该用户实际确实拥有那个权限。

图: 模拟 SQL 错误。

```
BEGIN
        IF user = 'pault' THEN
        RAISE EXCEPTION -273;   -- 拒绝 Paul 的 update 权限
        END IF
        END
```

使用 RAISE EXCEPTION 来退出嵌套的代码

下图展示您可如何使用 RAISE EXCEPTION 语句来退出深度嵌套的块。

图: RAISE EXCEPTION 语句。

```
BEGIN
        ON EXCEPTION IN (1)
        END EXCEPTION WITH RESUME -- do nothing significant (cont)

        BEGIN
        FOR i IN (1 TO 1000)
        FOREACH select ..INTO aa FROM t
        IF aa < 0 THEN
        RAISE EXCEPTION 1;        -- emergency exit
        END IF
        END FOREACH
        END FOR
        RETURN 1;
        END

        --do something;                    -- emergency exit to
```

```
        -- this statement.
        TRACE 'Negative value returned';
        RETURN -10;
        END
```

如果最内层的条件为真（如果 aa 为负），则发生异常，且执行跳至跟在该块的 END 之后的代码。在此情况下，执行跳至 TRACE 语句。

请记住，BEGINEND 块为单个语句。如果在块中的某处发生错误，且在该块之外捕获，则当执行恢复时，跳过该块剩余的部分，并从下一语句开始执行。

除非您在该块的某处为此错误设置捕获，否则，将错误条件传回至包含该调用的块，并传回至包含该块的任何块。如果不存在设置处理该错误的 ON EXCEPTION 语句，则停止该 SPL 例程的执行，为正在执行该 SPL 例程的例程创建一个错误。

12.14 检查 SPL 例程中处理的行数

在 SPL 例程内，您可使用 DBINFO 行数来找出在 SELECT、INSERT、UPDATE、DELETE、EXECUTE PROCEDURE 和 EXECUTE FUNCTION 语句中已处理了的行数。

下图展示一个 SPL 函数，使用带有 'sqlca.sqlerrd2' 选项的 DBINFO 行数，来确定从表删除的行数。

图: 确定从表删除的行数。

```
CREATE FUNCTION del_rows ( pnumb INT )
        RETURNING INT;

        DEFINE nrows INT;

        DELETE FROM sec_tab WHERE part_num = pnumb;
        LET nrows = DBINFO('sqlca.sqlerrd2');

        RETURN nrows;

        END FUNCTION;
```

要确保有效的结果，请在已执行完毕的 SELECT 和 EXECUTE PROCEDURE 或 EXECUTE FUNCTION 语句之后使用此选项。此外，如果您在游标内使用 'sqlca.sqlerrd2' 选择，请确保在关闭游标之前访存所有行，来保证有效的结果。

12.15 总结

SPL 例程为提高您的数据库处理效率提供许多机会，包括提升数据库性能，简化应用程序，以及限制或监视对数据的访问。您还可使用 SPL 例程来处理扩展的数据类型，诸如集合类型、row 类型、opaque 类型和 distinct 类型。要了解 SPL 语句的语法图，请参阅《GBase 8s SQL 指南：语法》。

13 创建和使用触发器

本章描述 CREATE TRIGGER 语句的每个组成部分的用途，说明触发器的一些用法，并描述将 SPL 例程用作触发器的优点。

此外，本章还描述可在视图上定义的 INSTEAD OF 触发器。

SQL 触发器是驻留在数据库中的一种机制。具有使用许可权的任何用户都可以使用它。SQL 触发器指定当数据操纵语言（DML）操作（INSERT 、SELECT 、DELETE 或 UPDATE 语句）时，数据库服务器应自动执行一个或多个附加操作。对于在视图上定义的触发器，视图基本表上的触发操替换触发事件。对于表或视图上的触发器，触发操作可以是 INSERT 、DELETE 、UPDATE 、EXECUTE PROCEDURE 或 EXECUTE FUNCTION 语句。

GBase 8s 还支持用 C 或 Java™ 编写的用户定义的例程作为触发操作。

有关如何撰写 C UDR 以获取有关触发器事件的元数据信息，请参阅《GBase 8s DataBlade API 程序员指南》。

13.1 何时使用触发器

因为触发器驻留在数据库中，且具有必需特权的任何用户都可以使用它，所以触发器允许您编写可供多个应用程序使用的一组 SQL 语句。它可在多个程序需要执行同一数据库操作时避免冗余码。

可使用触发器执行下列操作以及在此列表中找不到的其它操作：

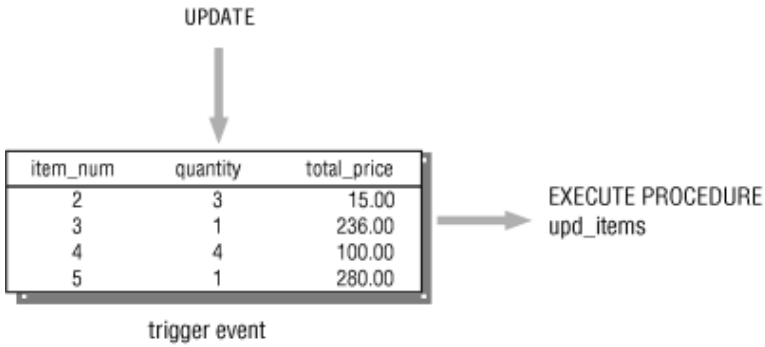
- 在数据库中创建活动的审计跟踪。例如：可通过更新审计表的确认信息来跟踪对订单的更新。
- 实现业务规则。例如：可以确定何时订单超出客户的信用卡限制并对此情况显示一条消息。
- 派生表内或数据内未提供的其它数据。例如：当对 items 表的 quantity 列进行更新时，可以计算对 total_price 列的相应调整。
- 强制执行引用完整性。例如：在删除客户时，可以使用触发器来删除 orders 表中具有相同客户号的相应行。

13.2 如何创建触发器

使用 CREATE TRIGGER 语句定义新触发器。CREATE TRIGGER 语句是数据定义语句，它将称为触发操作的 SQL 语句与表上的诱发事件相关联。当发生诱发操作时，它触发存储在数据库中的关联 SQL 语句。

在本示例中，触发事件是引用 items 表中的 quantity 列的 UPDATE 语句。下图说明激活触发器的 DML 操作（称为触发器事件）与触发操作之间的关系。

图: 触发事件和触发操作



CREATE TRIGGER 语句由执行下列操作的子句组成：

- 声明触发器名称。
- 指定在指定表或试图上作为触发事件的 DML 操作。
- 定义该事件触发器的 SQL 操作。

FOR EACH ROW 触发操作中讨论了称为 REFERENCING 子句的可选子句。

要创建触发器，使用 DB-Access 或某个 SQL API。本节描述当您在 DB-Access 中使用交互查询语言选项输入 CREATE TRIGGER 语句时的用法。在 SQL API 中，您在语句前加上将该语句标识为嵌入式语句的符号或关键字。

13.2.1 声明触发器名称

触发器名称标识触发器，且该名称在数据库的触发器名称中必须唯一。在语句中，触发名称跟在 CREATE TRIGGER 后面。与任何 SQL 标识一样，该名称最长为 128 个字节，以字母开始并由字母、数字和下划线（_）组成。在以下示例中，所示的 CREATE TRIGGER 语句部分声明了触发器的名称upqty：

```
CREATE TRIGGER upqty          -- declare trigger name
```

13.2.2 指定触发器事件

触发器事件是一种激活触发器的 DML 语句。当对表执行此类型的语句时，数据库服务器执行组成触发操作的 SQL 语句。对于表，触发器事件可以是 INSERT 、SELECT 、DELETE 或 UPDATE 语句。对于 UPDATE 或 SELECT 触发事件，可以指定表中一列或多列可以激活触发器。可以在同一表上定义多个 INSERT 、SELECT 、DELETE 和 UPDATE 触发器，也可以在同一视图上定义多个 INSERT 、DELETE 和 UPDATE 触发器。

在当前数据库中的一个表或一个视图上，只能创建一个触发器。触发器不能引用远程表或视图。

在以下 CREATE TRIGGER 语句片段中，触发事件定义为更新 items 表中 quantity 列：

```
CREATE TRIGGER upqty
UPDATE OF quantity ON items        -- an UPDATE trigger event
```

语句的这一部分标识在其上定义触发器的表。如果触发器事件为插入或删除操作，那么仅需要语句的类型和表名，如以下示例所示：

```
CREATE TRIGGER ins_qty
NSERT ON items                     -- an INSERT trigger event
```

13.2.3 定义触发操作

触发操作是当触发事件发生时执行的 SQL 语句。触发操作可以由 INSERT 、DELETE 、UPDATE 、EXECUTE FUNCTION 和 EXECUTE PROCEDURE 语句组成。但是，除了指定要执行什么操作外，还必须就触发语句何时执行这些操作。您有以下选择：

- 在触发语句执行之前
- 在触发语句执行之后
- 针对触发语句作用的每一行

表上的单个触发器可以为上述每一时间定义操作。

要定义触发操作，指定该操作何时发生，然后提供要执行的 SQL 语句，可使用关键字 BEFORE 、AFTER 或 FOR EACH ROW 指定该操作何时发生。然后是括在圆括号中的触发操作。下面的触发操作定义指定在触发语句之前执行 SPL 例程 upd_items_p1 ：

```
BEFORE(EXECUTE PROCEDURE upd_items_p1) -- a BEFORE action
```

13.2.4 完整的 CREATE TRIGGER 语句

要定义完整的 CREATE TRIGGER 语句，将触发器名称子句、触发事件子句和触发操作子句组合起来。下面的 CREATE TRIGGER 语句是将前例中的语句的各个组成部分组合在一起的结果。每次更新 items 表的 quantity 列时，此触发器将执行 SPL 例程 upd_items_p1。

```
CREATE TRIGGER upqty
        UPDATE OF quantity ON items
        BEFORE(EXECUTE PROCEDURE upd_items_p1);
```

如果在数据库服务器处理 CREATE TRIGGER 语句时，触发器定义中的数据库对象（例如，本例中的 SPL 例程 upd_items_p1 ）不存在，那么返回错误。

13.3 使用触发操作

要有效地使用触发器，需要理解触发语句和生成的触发操作之间的关系。在指定发生触发操作的时间（即，BEFORE 、AFTER 或 FOR EACH ROW）时定义此关系。

13.3.1 BEFORE 和 AFTER 触发操作

在触发事件之前或之后发生的触发操作仅执行一次。BEFORE 触发操作在触发语句之前执行，即在触发器事件发生之前执行，AFTER 触发操作在触发语句操作完成之后执行。即使触发语句不处理任何行，BEFORE 和 AFTER 触发操作也会执行。

除其它用法外，还可以使用 BEFORE 和 AFTER 触发操作来确定触发语句的效果。例如，在更新 items 表的 quantity 列之前，可以调用 SPL 例程upd_items_p1 来计算表中所有项的订购总数，如下例所示，该过程将总数存储在名为 old_qty 的全局变量中。

```
CREATE PROCEDURE upd_items_p1()
        DEFINE GLOBAL old_qty INT DEFAULT 0;
        LET old_qty = (SELECT SUM(quantity) FROM items);
        END PROCEDURE;
```

在完成触发更新之后，可以再次计算总数来看看更改了多少。下面的 SPL 例程 upd_items_p2 再次计算了 quantity 的总数并将结果存储在局部变量 new_qty 中。然后，它将 new_qty 与全局变量 old_qty 相比较，以查看所有订单的总量的增长是否超过 50%。若是，该过程将使用 RAISE EXCEPTION 语句来模拟 SQL 错误。

```
CREATE PROCEDURE upd_items_p2()
        DEFINE GLOBAL old_qty INT DEFAULT 0;
        DEFINE new_qty INT;
        LET new_qty = (SELECT SUM(quantity) FROM items);
        IF new_qty > old_qty * 1.50 THEN
        RAISE EXCEPTION -746, 0, 'Not allowed - rule violation';
```

```
        END IF
        END PROCEDURE;
```

下列触发器调用 upd_items_p1 和 upd_items_p2 以防止对 items 表的 quantity 列点进行异常更新：

```
CREATE TRIGGER up_items
        UPDATE OF quantity ON items
        BEFORE(EXECUTE PROCEDURE upd_items_p1())
        AFTER(EXECUTE PROCEDURE upd_items_p2());
```

如果更新使得对所有项的订购总量增长超过 50%，那么 upd_items_p2 中的 RAISE EXCEPTION 语句终止该触发器，并显示错误。当进行事务记录的数据库服务器中的触发器发生故障时，数据库服务器会回滚超过语句和触发操作进行的更改。有关触发器发生故障时所发生的情况的更多信息，请参阅《GBase 8s SQL 指南：语法》中的 CREATE TRIGGER 语句。

13.3.2 FOR EACH ROW 触发操作

FOR EACH ROW 触发操作对触发语句所作用的每一个行执行一次。例如，如果触发语句有下列语法，将对 manu_code列的值为 'KAR' 的 items 表中的每一行执行一次 FOR EACH ROW 触发操作：

```
UPDATE items SET quantity = quantity * 2
        WHERE manu_code = 'KAR';
```

如果触发事件不处理任何行，将不会执行 FOR EACH ROW 触发操作。

对于表上的触发器，如果触发事件为 SELECT 语句，那么该触发器称为选择触发器，并且触发操作在完成时对检索到的行的所有处理之后再执行。但是，触发操作可能不会立即执行；原因是会对查询返回的行的每一个实例执行 FOR EACH ROW 操作。例如，在带有 ORDER BY 子句的 SELECT 语句中，必须先根据 WHERE 子句限定所有行，它们才能排序并返回。

REFERENCING 子句

当创建 FOR EACH ROW 触发操作时，通常必须在触发操作语句中指示您引用的是触发语句生效之前还是之后的列值。例如：假定您想要跟踪对items 表的 quantity 列的更新。为此，创建下表以记录该活动：

```
CREATE TABLE log_record
        (item_num       SMALLINT,
        ord_num         INTEGER,
        username        CHARACTER(8),
        update_time     DATETIME YEAR TO MINUTE,
```

```
        old_qty         SMALLINT,
        new_qty          SMALLINT);
```

要为此表中的 old_qty 和 new_qty 列提供值，必须能够引用 items 表中的 quantity 的旧值和新值。即，触发语句作用之前和之后的值。REFERENCING 子句可使您做的这一点。

REFERENCING 子句允许您创建可与列名组合起来的两个前缀，一个用于引用列的旧值，另一个用于引用列的新值。这些前缀称为相关名。可以根据您的要求创建一个或两个相关名。您指出使用关键字 OLD 和 NEW 创建的哪个相关名。下面的 REFERENCING 子句创建相关名 pre_upd 和post_upd 来引用行中的旧值和新值：

```
REFERENCING OLD AS pre_upd NEW AS post_upd
```

当更新 items 表中的某行中的 quantity 时，以下触发操作将在 log_record 中创建一行。INSERT 语句引用 item_num 和 order_num 列的旧值并引用 quantity 列的新值和旧值。

```
FOR EACH ROW(INSERT INTO log_record
        VALUES (pre_upd.item_num, pre_upd.order_num, USER,
        CURRENT, pre_upd.quantity, post_upd.quantity));
```

在 REFERENCING 子句中定义的相关名应用于触发语句作用的所有行。

重要： 如果引用未被相关名限定的列名，数据库服务器不会专门在触发表的定义中搜索该列。必须总是将相关名与 FOR EACH ROW 触发操作中的 SQL 语句中的列名结合使用，除非该语句独立有效，而与触发操作无关。有关更多信息，请参阅《GBase 8s SQL 指南：语法》中的 CREATE TRIGGER 语句。

WHEN 条件

作为表上的触发器的选项，可在 WHEN 子句之前加上触发操作以使该操作依赖于测试结果。WHEN 子句由关键字 WHEN 以及跟随在其后的括在圆括号中的条件语句所组成。在 CREATE TRIGGER 语句中，WHEN 子句跟在关键字 BEFORE 、AFTER 或 OR EACH ROW 之后，触发操作列表之前。

当 WHEN 条件存在时，如果它求值为 true，那么按触发操作的出现顺序执行这些操作。如果 WHEN 条件求值为 false 或 unknown，那么不执行触发操作列表中的操作。如果触发器指定 FOR EACH ROW，那么还将针对每一行对条件进行求值。

在下面的触发器示例中，仅当 WHEN 子句中的条件为 true （即，如果更新后单价高于更新前单价的两倍）时才执行触发操作：

```
CREATE TRIGGER up_price
        UPDATE OF unit_price ON stock
        REFERENCING OLD AS pre NEW AS post
        FOR EACH ROW WHEN(post.unit_price > pre.unit_price * 2)
        (INSERT INTO warn_tab
        VALUES(pre.stock_num, pre.manu_code, pre.unit_price,
```

```
        post.unit_price, CURRENT));
```

有关 WHEN 条件的更多信息，请参阅《GBase 8s SQL 指南：语法》中的 CREATE TRIGGER 语句。

13.3.3 将 SPL 例程用作触发操作

触发器最强大的功能可能是能够将 SPL 例程作为触发操作进行调用，调用 SPL 例程的 EXECUTE PROCEDURE 或 EXECUTE FUNCTION 语句允许您将数据从触发表传递至 SPL 例程，还允许您使用由 SPL 例程返回的数据更新触发表。SPL 还允许您定义变量、对其指定数据、进行比较以及使用过程语句来完成触发操作内的复杂任务。

将数据传至 SPL 例程

可以在 EXECUTE PROCEDURE 或 EXECUTE FUNCTION 语句的参数列表中将数据传递至 SPL 例程。以下示例中的 EXECUTE PROCEDURE 语句将值从 items 表的 quantity 和 total_price 列传递至 SPL 例程 calc_totpr：

```
CREATE TRIGGER upd_totpr
    UPDATE OF quantity ON items
    REFERENCING OLD AS pre_upd NEW AS post_upd
    FOR EACH ROW(EXECUTE PROCEDURE calc_totpr(pre_upd.quantity,
        post_upd.quantity, pre_upd.total_price) INTO total_price);
```

将数据传递至 SPL 例程允许您在该例程执行的操作中使用数据值。

使用 SPL

在之前触发列中的 EXECUTE PROCEDURE 语句调用以下示例所示的 SPL 例程。在更新 items 表中的 quantity 时，该过程使用 SPL 计算需要对total_price 列作出的更改。该过程接收 quantity 的旧值和新值以及 total_price 的旧值。它用旧的总价除以旧的数值来得出单价。然后用新的数量乘以单价来得出新的总价。

```
CREATE PROCEDURE calc_totpr(old_qty SMALLINT, new_qty SMALLINT,
            total MONEY(8)) RETURNING MONEY(8);
            DEFINE u_price LIKE items.total_price;
            DEFINE n_total LIKE items.total_price;
            LET u_price = total / old_qty;
            LET n_total = new_qty * u_price;
            RETURN n_total;
            END PROCEDURE;
```

在本示例中，SPL 允许触发器派生不能直接从触发表获得的数据。

用 SPL 例程中的数据更新非触发列

在触发操作内，EXECUTE PROCEDURE 语句的 INTO 子句允许您更新触发表中的非触发列。下例中的 EXECUTE PROCEDURE 语句调用包含 INTO 子句（该子句引用 total_price 列）的 calc_totpr SPL 过程：

```
FOR EACH ROW(EXECUTE PROCEDURE calc_totpr(pre_upd.quantity,
                post_upd.quantity, pre_upd.total_price) INTO total_price);
```

更新到 total_price 中的值是 SPL 过程结束时由 RETURN 语句返回的。对触发语句作用的每一行更新 total_price 列。

13.4 触发器例程

可以定义称为触发器例程专用 SPL 例程，此类例程只能从触发器操作的 FOR EACH ROW 段进行调用。与 EXECUTE FUNCTION 或 EXECUTE PROCEDURE 例程可以从触发操作列表中调用的普通 UDR 不同，触发器例程包含自己的 REFERENCING 子句，可用于为触发操作修改的行中原有列和新列值定义相关名。这些相关名可以在触发器例程中的 SPL 语句中引用，为触发操作可在表或视图中修改数据的方式提供更大的灵活性。

触发器例程也可使用称为 DELETING 、INSERTING 、SELECTING 和 UPDATING 触发器类型的布尔运算符，以标识已调用触发器例程的触发器的类型。触发器例程还可以调用 mi_trigger* 例程（有时称为触发器自省类型）来获取关于已调用触发器例程的上下文的信息。

触发器例程由包含 WITH TRIGGER REFERENCES 关键字的 EXECUTE FUNCTION 或 EXECUTE PROCEDURE 语句调用。这些语句必须从触发操作的 FOR EACH ROW 段中调用触发器例程，而不是从 BEFORE 或 AFTER 段中进行调用。

有关支持定义和执行触发器例程的 SQL 的 CREATE FUNCTION 、CREATE PROCEDURE 、EXECUTE FUNCTION 和 EXECUTE PROCEDURE 语句的语法特征的信息，请参阅《GBase 8s SQL 指南：语法》。有关 mi_trigger* 例程的更多信息，请参阅《GBase 8s DataBlade API 程序员指南》。

13.5 表层次结构中的触发器

当您在超表上定义触发器时，表层次结构中的所有子表也会继承该触发器。因此，当您对层次结构中的表执行操作时，可对层次结构中作为对其定义触发器的表的子表的任何表执行触发器。

13.6 Select 触发器

当 CREATE TRIGGER 语句将指定表上的任何查询定义为其触发事件（

```
SELECT ON table
```

或

```
SELECT ON column-list ON table
```

）时，生成的触发对象是指定表的 Select 触发器。同一触发器还可以被包含将此表作为其基本表的触发列的视图上的查询激活。但是，SELECT 语句不能是视图上 INSTEAD OF 触发器的触发事件。

如果 CREATE TRIGGER 语句在嵌入 Select 触发事件的定义中也包含列列表，并且指定表上后续查询的投影列表不包含任何指定的列，那么该查询不能是此 Select 触发器的触发事件。

警告：

Select 触发器不建议用于审计。不要出于执行应用程序指定审计的目的而在表或其列的子集上尝试创建 Select 触发器。一般情况下，通过创建 Select 触发器来跟踪表上的 Select 动作的数量，以在每次用户查询某个表时将审计记录插入到审计表中是不可能的。

例如，假设您在表 AuditedTable 上定义了 Select 触发器，且对 AuditedTable 持有 Select 特权的用户发出了以下查询：

```
SELECT a.* FROM (SELECT * FROM AuditedTable) AS a;
```

数据库服务器不发出错误，但是 AuditedTable 上的 SELECT 触发器不会被此查询激活。包含集合运算符（例如 UNION 或 INTERSECT）的查询，或者其它 Select 触发器不支持的语法，将会被基于 Select 触发器的审计记录策略无视。

因为执行 Select 触发器的大量的限制（部分在本章中列出），生成的 Select 触发操作通常仅对应于试图枚举的任何逻辑 Select 事件的子集（它可能为空）。

13.6.1 执行触发操作的 SELECT 语句

当创建 select 触发器时，仅某些类型的 SELECT 语句可以执行对该触发器定义的操作。Select 触发器仅当下列类型的 SELECT 语句执行：

- 独立的 SELECT 语句t
- SELECT 语句的选择列表中的集合子查询
- 嵌入用户定义例程的 SELECT 语句
- 视图

独立 SELECT 语句

假设您对表定义了下面的 Select 触发器：

```
CREATE TRIGGER hits_trig SELECT OF col_a ON tab_a
        REFERENCING OLD AS hit
        FOR EACH ROW (INSERT INTO hits_log
        VALUES (hit.col_a, CURRENT, USER));
```

当触发列出现在独立 SELECT 语句的选择列表中时执行 Select 触发器。以下语句针对数据库服务器返回的行的每个实例执行 hits_trig 触发器上的触发操作：

```
SELECT col_a FROM tab_a;
```

查询投影列表中的集合子查询

当触发列出现在位于其它 SELECT 语句的投影列表中的集合子查询中时，将执行 Select 触发器。以下语句针对集合子查询返回的行的每个实例执行hits_trig 触发器上的触发操作：

```
SELECT MULTISET(SELECT col_a FROM tab_a) FROM ...
```

嵌入在用户定义例程中的 SELECT 语句

对嵌入在用户定义的例程（UDR）中的 SELECT 语句定义的选择触发器仅在以下情况下执行触发操作：

- UDR 出现在 SELECT 语句的选择列表中
- UDR 使用 EXECUTE PROCEDURE 语句调用

假设您创建包含语句 SELECT col_a FROM tab_a 的例程 new_proc。下面的每条语句针对嵌入的 SELECT 语句所返回行的每一个实例执行 hits_trig触发器的触发操作：

```
SELECT new_proc() FROM tab_b;
        EXECUTE PROCEDURE new_proc;
```

视图

Select 触发器对其基础表包含触发列的引用的视图执行触发操作。但是，不能在视图上定义 Select 触发器。

假设您创建了下列视图：

```
CREATE VIEW view_tab AS
        SELECT * FROM tab_a;
```

以下语句针对视图返回的行的每个实例执行 hits_trig 触发器上的触发操作：

```
SELECT * FROM view_tab;
```

```
SELECT col_a FROM tab_a;
```

13.6.2 执行 Select 触发器的限制

下列类型的 SELECT 语句不会触发 Select 触发器上的任何操作。

- 触发列不在投影列表中（例如，出现在 SELECT 语句的 WHERE 子句中的列不会执行 Select 触发器）。
- 引用远程表的 SELECT 语句。
- SELECT 语句调用聚集函数或 OLAP 窗口聚集函数。
- SELECT 语句包含集合运算符或（UNION 、UNION ALL 、INTERSECT 、MINUS 或 EXCEPT）。
- SELECT 语句包含 DISTINCT 或 UNIQUE 关键字。
- 包含 SELECT 语句的 UDR 表达式不在投影列表中。
- SELECT 语句出现在 INSERT INTO 语句中。
- SELECT 语句出现在滚动游标中。
- 触发器是级联 Select 触发器。

  级联 Select 触发器是其操作包含 SPL 例程的触发器，该例程本身具有触发 select 语句。但是，不执行级联 Select 触发器的操作，数据库服务器也不返回错误。

13.6.3 在表层次结构中的表的 Select 触发器

当您对超表定义 select 触发器，表层次结构中的所有子表也会继承此触发器。

有关覆盖和禁用继承触发器的信息，请参阅表层次结构中的触发器。

13.7 可重入触发器

可重入触发器指的是其中触发操作可引用触发表的情况。换句话说，也就是触发器事件和触发操作可作用于同一个表。例如，假设下面的 UPDATE 语句表示触发事件：

```
UPDATE tab1 SET (col_a, col_b) = (col_a + 1, col_b + 1);
```

以下触发操作是合法的，因为列 col_c 不是触发事件已更新的列：

```
UPDATE tab1 SET (col_c) = (col_c + 3);
```

在前面的示例中，对 col_a 或 col_b 的触发操作可能是非法的，因为触发操作不能是引用触发事件所更新的列的 UPDATE 语句。

重要： Select 触发器不能是可重入触发器。如果触发事件为 SELECT 语句，那么不能对同一个表执行触发操作。

有关描述在哪些情况下触发器可为或不可为可重入触发器的规则的列表，请参阅《GBase 8s SQL 指南：语法》中的 CREATE TRIGGER 语句。

13.8 视图上的 INSTEAD OF 触发器

视图是使用 CREATE VIEW 语句创建并使用 SELECT 语句定义的虚拟表。每个视图由若干行列集合组成，它们是在您每次通过查询引用该视图时，由其视图定义中的 SELECT 语句返回的。要在视图的基本表中插入、更新或删除行，可以定义 INSTEAD OF 触发器。

与表上的触发器不同，视图上的 INSTEAD OF 触发器导致 GBase 8s 忽略触发事件，而只执行触发操作。

有关 CREATE VIEW 语句和 INSTEAD OF 触发器语法和规则的信息，包括将对视图插入行的 INSTEAD OF 触发器的示例，请参阅《GBase 8s SQL 指南：语法》。

13.8.1 使用 INSTEAD OF 触发器对视图进行更新

在创建一个或多个表之后（如下例中名为 dept 和 emp 的表），然后又创建了基于 dept 和 emp 的视图（如名为 manager_info 的视图）之后，使用 INSTEAD OF 触发器更新该视图。

以下 CREATE TRIGGER 语句创建 manager_info_update，这是一个 INSTEAD OF 触发器，用来通过 manager_info 视图更新 dept 和 emp 表中的行。

```
CREATE TRIGGER manager_info_update
        INSTEAD OF UPDATE ON manager_info
        REFERENCING NEW AS n
        FOR EACH ROW
        (EXECUTE PROCEDURE updtab (n.empno, n.empname, n.deptno,));

        CREATE PROCEDURE updtab (eno INT, ename CHAR(20), dno INT,)
        DEFINE deptcode INT;
        UPDATE dept SET manager_num = eno where deptno = dno;
        SELECT deptno INTO deptcode FROM emp WHERE empno = eno;
        IF dno !=deptcode THEN
        UPDATE emp SET deptno = dno WHERE empno = eno;
```

```
        END IF;
        END PROCEDURE;
```

在创建了表、视图、触发器和 SPL 例程以后，数据库服务器将下面的 UPDATE 语句视作触发事件：

```
UPDATE manager_info
        SET empno = 3666, empname = "Steve"
        WHERE deptno = 01;
```

此触发 UPDATE 语句不会得到执行，但是此事件将造成执行触发器操作，即调用 updtab() SPL 例程。SPL 例程中的 UPDATE 语句将值更新到manager_info 视图的 emp 和 dept 基本表中。

13.9 跟踪触发操作

如果触发操作并未按您所期望的那样运行，那么将其放置在 SPL 例程中，并使用 SPL TRACE 语句来监视其操作。在启动跟踪之前，必须使用 SET DEBUG FILE TO 语句将输出定向到文件。

13.9.1 SQL 例程中的 TRACE 语句的示例

以下示例显示了添加到 SPL 例程 items_pct 中的 TRACE 语句。SET DEBUG FILE TO 语句将跟踪输出定向至路径名所指定的文件。TRACE ON 语句开始跟踪过程中的语句和变量。

```
CREATE PROCEDURE items_pct(mac CHAR(3))
DEFINE tp MONEY;
DEFINE mc_tot MONEY;
DEFINE pct DECIMAL;
SET DEBUG FILE TO 'pathname';

TRACE 'begin trace';
TRACE ON;
LET tp = (SELECT SUM(total_price) FROM items);
LET mc_tot = (SELECT SUM(total_price) FROM items
    WHERE manu_code = mac);
LET pct = mc_tot / tp;
IF pct > .10 THEN
    RAISE EXCEPTION -745;
```

```
END IF
TRACE OFF;
END PROCEDURE;

CREATE TRIGGER items_ins
INSERT ON items
REFERENCING NEW AS post_ins
FOR EACH ROW(EXECUTE PROCEDURE items_pct (post_ins.manu_code));
```

13.9.2 TRACE 输出的示例

以下示例显示了 items_pct 过程中的样本跟踪输出，这些输出出现在 SET DEBUG FILE TO 语句所指定的文件中。这些输出显示过程变量、过程参数、返回值和错误代码的值。

```
trace expression :begin trace
        trace on
        expression:
        (select (sum total_price)
        from items)
        evaluates to $18280.77 ;
        let    tp = $18280.77
        expression:
        (select (sum total_price)
        from items
        where (= manu_code, mac))
        evaluates to $3008.00 ;
        let    mc_tot = $3008.00
        expression:(/ mc_tot, tp)
        evaluates to 0.16
        let    pct = 0.16
        expression:(> pct, 0.1)
        evaluates to 1
        expression:(- 745)
        evaluates to -745
        raise exception :-745, 0, ''
        exception : looking for handler
        SQL error = -745 ISAM error = 0    error string =    = ''
        exception : no appropriate handler
```

有关如何使用 TRACE 语句诊断 SPL 例程中的逻辑错误的更多信息，请参阅创建和使用 SPL 例程。

13.10 生成错误消息

当触发器因为 SQL 语句而失败时，数据库服务器将返回适用于特定失败原因的 SQL 错误号。

当触发操作是 SPL 例程时，可以使用两个保留错误号的其中之一针对错误情况生成错误消息。第一个是错误号 -745，它具有通用且固定的错误消息。第二个是错误号 -746，它允许您提供最多 70 字节的信息正文。

13.10.1 应用固定错误消息

可以将错误号 -745 应用于并非 SQL 错误的任何触发器故障。下列固定消息用于此错误：-745 Trigger execution has failed。

可以在 SPL 中将此消息应用于 RAISE EXCEPTION 语句。在以下示例中，如果 new_qty 大于 1.50 倍的 old_qty，那么生成错误号 -745 ：

```
CREATE PROCEDURE upd_items_p2()
        DEFINE GLOBAL old_qty INT DEFAULT 0;
        DEFINE new_qty INT;
        LET new_qty = (SELECT SUM(quantity) FROM items);
        IF new_qty > old_qty * 1.50 THEN
        RAISE EXCEPTION -745;
        END IF
        END PROCEDURE
```

如果您正在使用 DB-Access，那么错误 -745 消息的文本在屏幕的底部显示，如下图所示。

图: 带有固定消息的错误消息 -745

```
Press CTRL-W for Help
SQL: New Run   Modify   Use-editor   Output   Choose Save   Info   Drop   Exit
Modify the current SQL statements using the SQL editor.

---------------------- stores8@myserver --------- Press CTRL-W for Help ----

INSERT INTO items VALUES( 2, 1001, 2, 'HRO', 1, 126.00);
```

```
745: Trigger execution has failed.
```

如果触发器在 SQL API 中通过 SQL 语句调用包含错误的过程，数据库服务器将把 SQL 错误变量设置为 -745，并将其返回至程序。要显示消息正文，遵循 GBase 8s 应用程序开发工具提供的过程以检索 SQL 错误消息的正文。

13.10.2 生成可变错误消息

错误号 -746 允许您提供错误消息的正文。就像前面的示例，如果 new_qty 大于 1.50 倍的 old_qty，以下示例也将生成错误。但是，在本例中，错误号为 -746，并且消息正文 Too many items for Mfr. 的项过多是作为 RAISE EXCEPTION 语句中的第三个参数提供的。有关此语句的语法和使用的更多信息，请参阅创建和使用 SPL 例程 中的 RAISE EXCEPTION 语句。

```
CREATE PROCEDURE upd_items_p2()
    DEFINE GLOBAL old_qty INT DEFAULT 0;
    DEFINE new_qty INT;
    LET new_qty = (SELECT SUM(quantity) FROM items);
    IF new_qty > old_qty * 1.50 THEN
        RAISE EXCEPTION -746, 0, 'Too many items for Mfr.';
    END IF
END PROCEDURE;
```

如果使用 DB-Access 提交触发语句，并且如果 new_qty 大于 old_qty，那么您将得到下图显示的结果。

图: 带有用户指定消息正文的错误号 -746

```
Press CTRL-W for Help

SQL:  New  Run  Modify  Use-editor  Output  Choose  Save  Info  Drop
Exit

Modify the current SQL statements using the SQL editor.
```

```
---------- store7@myserver --------- Press CTRL-W for Help -----

INSERT INTO items VALUES( 2, 1001, 2, 'HRO', 1, 126.00);









746: Too many items for Mfr.
```

如果在 SQL API 中通过 SQL 语句调用触发器，那么数据库服务器将 sqlcode 设置为 -746，并在 SQL 通信区域（SQL;CA）的sqlerrm 字段中返回消息正文。有关如何使用 SQL;CA 的更多信息，请参阅您的 SQL API 出版物。

13.11 总结

为介绍触发器，本章讨论了下列主题：

- CREATE TRIGGER 语句的各个组成部分
- 可作为触发事件的 DML 语句的类型
- 可作为触发操作的 SQL 语句的类型
- 如何创建 BEFORE 和 AFTER 触发操作以及如何使用它们来确定触发语句的影响
- 如何创建 FOR EACH ROW 触发操作，以及如何使用 REFERENCING 子句引用触发语句执行之前和之后的列值
- 视图上的 INSTEAD OF 触发器，其触发事件将被忽略，但触发操作可以修改视图的基本表
- 将 SPL 例程用作触发操作的好处
- 将触发例程作为触发操作调用的特殊功能
- 在触发操作的执行异常时如何跟踪它们
- 如何在触发操作内生成两种类型的错误消息

GBASE®

南大通用数据技术股份有限公司
General Data Technology Co., Ltd.

微信二维码

技术支持热线：400-013-9696