IBM® DB2® Spatial Extender

用户指南和参考

版本 7

SB84-0249-00

IBM® DB2® Spatial Extender

用户指南和参考

版本 7

SB84-0249-00

在使用本资料和它支持的产品之前，请参阅第297页的『注意事项』中的一般信息。

目录

图

表

关于本书

本书分为两部分。第一部分包含关于 DB2 Spatial Extender 的概念信息，并说明如何在 Windows NT 和 AIX 系统上安装、配置、管理 DB2 Spatial Extender 以及如何对它进行编程。第二部分包括与 DB2 Spatial Extender 一起使用的存储过程、几何图形、函数、信息和目录视图的参考信息。

谁应阅读本书

本书是为设置 Spatial 环境的管理员和开发具有 Spatial 数据的应用程序的应用程序员编写的。

约定

本书使用下列突出显示约定:

黑体 指示命令和图形用户界面 (GUI) 控件 （例如，字段名、文件夹名、菜单选项）。

等高等宽字体
指示输入的编码或文本的示例。

斜体 指示应该用值替换的变量。斜体也指示书名和强调单词。

大写 指示 SQL 关键字和对象（例如，表、视图和服务器）的名称。

如何发送意见

您的反馈能帮助 IBM 提供高质量的信息。请将您对这本书或其他 DB2 文档的任何意见发送给我们。您可以用下列任何方法来提供意见:

- 从 Web 发送您的意见。可在以下网址处访问 “IBM 数据管理” 联机读者意见表: http://www.ibm.com/software/data/rcf
- 通过电子邮件将意见发送至 comments@vnet.ibm.com。务必包括产品的名称、产品的版本号，以及书名和书的第几部分（若适用的话）。若对特定文本发表意见，则请包括该文本的位置（例如，章节标题、表号、页号或帮助主题标题）。

第1部分 使用 DB2 Spatial Extender

第1章 关于 DB2 Spatial Extender

本章介绍 DB2 Spatial Extender，说明其用途，讨论它处理的数据并举例说明如何使用它。本章的末尾有对本书其余部分的快速指南。

DB2 Spatial Extender 的用途

可使用 DB2 Spatial Extender 来创建*地理信息系统* (GIS)：它是一个对象、数据和应用程序的综合体，允许您生成和分析关于地形的 Spatial 信息。*地形*包括构成地球表面的对象和占据地球表面的对象。它们组成了自然环境（如江河、森林、丘陵和沙漠）和人文环境（城市、住宅、办公楼、界标等）。

*Spatial 信息*包括如下事实：

- 地形相对于其周围环境的位置（例如，城市内医院和诊所所在的地点，或城市的住宅与当地地震带的距离）
- 地形彼此相关的方式（例如，某个江河系统包含在特定地区内的信息，或该地区的某些桥架设在该江河系统的支流上的信息）
- 对某种或多种地形使用的测量方法，（例如，办公楼与其地块边界之间的距离，或鸟类保护区的周长）。

Spatial 信息自身或与传统的关系数据库管理系统 (RDBMS) 输出组合在一起，可帮助您设计项目和作出商业和策略决定。例如，假设县福利区的管理员需要验证哪些福利申请者和接受者确实居住在该福利区服务的区域内。 DB2 Spatial Extender 可从服务区域的位置与申请者和接受者的地址得出此信息。

或假设连锁餐馆的拥有者想要在邻近的城市开展业务。要确定在何处开张新餐馆，拥有者需要回答如下问题：这些城市中哪些地方是经常光顾我的餐馆的顾客的集中地？主要的公路在什么地方？哪里的犯罪率最低？竞争对手的餐馆位于何处？ DB2 Spatial Extender 可以直观显示方式产生 Spatial 信息来回答这些问题，基础 RDBMS 可生成标号和文本来说明这些显示。

本书中还有 DB2 Spatial Extender 用途的其他几个示例，主要在第45页的『第7章 检索和分析 Spatial 信息』、第49页的『第8章 编写 DB2 Spatial Extender 的应用程序』和第143页的『第14章 用于 SQL 查询的 Spatial 函数』中。

表示地形的数据

本节提供数据的概述，您将生成、存储和操纵这些数据来获取 Spatial 信息。包括如下主题:

- 数据如何表示地形
- Spatial 数据的性质
- 产生 Spatial 数据的方法

数据如何表示地形

在 DB2 Spatial Extender 中，可用表或视图中的一行或某一行的一部分来表示地形。例如，考虑第3页的『DB2 Spatial Extender 的用途』中提及的两个地形: 办公楼和住宅。在图1中，BRANCHES 表的每行表示银行的一个分支机构。作为变化，图1中 CUSTOMERS 表的每行作为一个整体表示该银行的客户。然而，每行的部分 — 特别是包含客户地址的单元 — 可看作表示该客户的住宅。

BRANCHES

ID	NAME	ADDRESS	CITY	STATE	ZIP
937	Airzone-Multern	92467 Airzone Blvd	San Jose	CA	95141

CUSTOMERS

ID	LAST NAME	FIRST NAME	ADDRESS	CITY	STATE	ZIP	CHECKING	SAVINGS
59-6396	Kriner	Endela	9 Concourt Circle	San Jose	CA	95141	A	A

图 1. 表示地形的表行; 其地址数据表示地形的表行. BRANCHES 表中的数据行表示银行的一个分支机构。 CUSTOMERS 表中的地址数据单元表示客户的住宅。两个表中的名称和地址都是虚构的。

图1中的表包含标识和描述该银行的分支机构和客户的数据。这样的数据称作*属性数据*。

属性数据的子集—表示分支机构和客户地址的值—可转换为产生 Spatial 信息的值。例如，如图1中所示，一个分支机构的地址为 92467 Airzone Blvd., San Jose CA 95141。客户的地址为 9 Concourt Circle, San Jose CA 95141。 DB2 Spatial Extender 可将这些地址转换为指示该分支机构和客户的家所在位置的值，这些值与这些位置周围的事物有关。 第5页的图2显示 BRANCHES 和 CUSTOMERS 表，它们具有用来包含这样的值的新列。

BRANCHES

ID	NAME	ADDRESS	CITY	STATE	ZIP	LOCATION
937	Airzone-Multern	92467 Airzone Blvd	San Jose	CA	95141	

CUSTOMERS

ID	LAST NAME	FIRST NAME	ADDRESS	CITY	STATE	ZIP	LOCATION	CHECKING	SAVINGS
59-6396	Kriner	Endela	9 Concourt Circle	San Jose	CA	95141		A	A

图 2. 添加了 Spatial 列的表. 在每个表中，LOCATION 列将包含对应于地址的坐标。

当地址和类似的标识符用作 Spatial 信息的起始点时，将它们称作*源数据*。 因为从它们派生的值产生 Spatial 信息，所以将这些派生值称作 *Spatial 数据*。 下一节描述 Spatial 数据并介绍其相关的数据类型。

Spatial 数据的性质

许多 Spatial 数据由坐标组成。坐标是表示相对于参考点的位置的数字。例如，纬度是表示相对于赤道的位置的坐标。经度是表示相对于格林尼治子午线的位置的坐标。因此，黄石国家公园的位置由其纬度（北纬 44.45 度）和经度（西经 110.40 度）定义。

纬度、经度、它们的参考点和其他相关参数总称为*坐标系*。 也存在着不是基于纬度和经度的值的坐标系。这些坐标系对位置、参考点和附加辨别参数有它们自己的度量单位。

最简单的 Spatial 数据项由定义单个地形的位置的两个坐标组成。（*数据项*是占据关系表的单元的一个或多个值。）更广泛的 Spatial 数据项由定义直线路径的几个坐标组成，如道路或江河可能形成直线路径。第三种 Spatial 数据项由定义区域的周边的坐标组成，例如，陆地或洪泛区的边缘。这些 Spatial 数据项和 DB2 Spatial Extender 支持的其他 Spatial 数据项在第109页的『第13章 几何图形和相关的 Spatial 函数』中有更详尽的描述。

每个 Spatial 数据项都是 Spatial 数据类型的一个实例。标记位置的两个坐标的数据类型为 ST_Point; 定义直线路径的坐标的数据类型为 ST_LineString; 定义周边的坐标的数据类型为 ST_Polygon。这些类型和 Spatial 数据的其他数据类型都是属于单个层次的结构化类型。有关层次的概述，参见第29页的『关于 Spatial 数据类型』。

Spatial 数据的来源

可用下列方法获取 Spatial 数据:

- 从属性数据派生
- 从其他 Spatial 数据派生
- 调入它

使用属性数据作为源数据

DB2 Spatial Extender 可从地址等属性数据派生 Spatial 数据（如第4页的『数据如何表示地形』中所述）。此过程称作*地理编码*。要查看涉及的顺序，将第5页的图2看作“之前”图片，并将图3看作“之后”图片。 第5页的图2显示 BRANCHES 表和 CUSTOMERS 表都具有指定用于 Spatial 数据的空列。假设 DB2 Spatial Extender 对这些表中的地址进行地理编码以获取对应这些地址的坐标，并将这些坐标放入列中。 图3举例说明了此结果。

分部

ID	NAME	ADDRESS	CITY	STATE	ZIP	LOCATION
937	Airzone-Multern	92467 Airzone Blvd	San Jose	CA	95141	1653 3094

客户

ID	LAST NAME	FIRST NAME	ADDRESS	CITY	STATE	ZIP	LOCATION	CHECKING	SAVINGS
59-6396	Kriner	Endela	9 Concourt Circle	San Jose	CA	95141	953 1527	A	A

图 3. 包括从源数据派生的 Spatial 数据的表. CUSTOMERS 表中的 LOCATION 列包含地理编码器从 ADDRESS、CITY、STATE 和 ZIP 列中的地址派生的坐标。类似地，BRANCHES 表中的 LOCATION 列包含该地理编码器从此表的 ADDRESS、CITY、STATE 和 ZIP 列中的地址派生的坐标。此示例是虚构的；显示模拟的坐标，而非实际的坐标。

DB2 Spatial Extender 使用称作*地理编码器*的函数将属性数据转换为 Spatial 数据，并将此 Spatial 数据放入表列中。有关地理编码器的更多信息，参见第35页的『关于地理编码』。

使用其他 Spatial 数据作为源数据

不仅可从属性数据生成 Spatial 数据，也可从其他 Spatial 数据生成 Spatial 数据。例如，假设在 BRANCHES 表中定义了其分支机构的银行想要知道有多少客户住在每个分支机构的五英里范围内。在该银行可从数据库获取此信息之前，它必须给数据库提供位于每个分支机构周围五英里半径内的区域的定义。 DB2 Spatial Extender 函数 ST_Buffer 可创建这样的定义。 ST_Buffer 使用每个分支机构的坐

标作为输入，可生成给希望的区域的周边定界的坐标。 图4显示具有 ST_Buffer 提供的信息的 BRANCHES 表。

BRANCHES

ID	NAME	ADDRESS	CITY	STATE	ZIP	LOCATION	SALES_AREA
937	Airzone-Multern	92467 Airzone Blvd	San Jose	CA	95141	1653 3094	1002 2001, 1192 3564, 2502 3415, 1915 3394, 1002 2001

图 4. 包括从现存 Spatial 数据派生的新 Spatial 数据的表. SALES_AREA 列中的坐标是 ST_Buffer 函数从 LOCATION 列中的坐标派生的。象 LOCATION 列中的坐标一样，SALES_AREA 列中的坐标也是模拟的；它们不是实际的。

除了 ST_Buffer 之外，DB2 Spatial Extender 还提供了其他几个从现存 Spatial 数据派生新 Spatial 数据的函数。有关 ST_Buffer 和这些其他函数的说明，参见第133页的『从现存几何图形生成新几何图形的函数』。

调入 Spatial 数据

获取 Spatial 数据的第三种方法是从文件中调入它，这些文件的格式是 DB2 Spatial Extender 所支持的格式之一。有关这些格式的说明，参见第273页的『第16章 Spatial 数据的文件格式』。这些文件包含通常应用于地图的数据：人口普查、洪泛区、地震断层等。通过将这样的数据与产生的 Spatial 数据组合使用，可增加可用的地理信息。例如，若市政工程部门需要确定住宅区易遭受什么灾害，它可使用 ST_Buffer 在该社区周围定义一个区域。然后市政工程部门可调入有关洪泛区和地震断层的数据，以查看这些问题区域中的哪些区域与该区域重叠。

如何创建和使用 DB2 Spatial Extender GIS

通过在 DB2 Spatial Extender 和它的基础 DB2 RDBMS 的组合环境中设置 DB2 Spatial Extender 并开发 GIS 项目，可创建 DB2 Spatial Extender GIS。通过实现这些项目，即通过生成和分析设计这些项目提供的 Spatial 和传统信息，来使用 GIS。整个过程涉及执行几组任务。本节介绍可用来执行这些任务的界面，提供这些任务的概述并提供一个方案来说明它们。

DB2 Spatial Extender 界面和相关功能性

本节概述可用来创建 DB2 Spatial Extender GIS（即为它设置资源、获取 Spatial 数据等）和使用它（即生成和分析有关地形的信息）的界面。

可用下列方法创建 DB2 Spatial Extender GIS:

- 使用 “DB2 控制中心” 的 DB2 Spatial Extender 窗口和菜单选项。有关指示，参见:
  - 第21页的『第3章 设置资源』
  - 第29页的『第4章 定义 Spatial 列，将它们注册为层并启用地理编码器自动维护它们』
  - 第35页的『第5章 填充 Spatial 列』
  - 第43页的『第6章 创建 Spatial 索引』
- 运行调用 DB2 Spatial Extender 存储过程的应用程序。有关开发这样的程序的指南，参见第49页的『第8章 编写 DB2 Spatial Extender 的应用程序』。
- 同时使用 “控制中心” 和应用程序。例如，可使用 “控制中心” 调用缺省地理编码器。另外，若想使用另一地理编码器，必须先通过调用应用程序中的 db2gse.gse_register_gc 存储过程将它注册到 DB2 Spatial Extender。（有关非缺省地理编码器的信息，参见第35页的『关于地理编码』。有关 db2gse.gse_register_gc 存储过程的信息，参见第77页的『db2gse.gse_register_gc』。）
- 与其他界面组合使用 “控制中心” 和／或应用程序。例如，要创建一个表来保持 Spatial 函数（如地理编码器）生成的数据，可使用 “命令行处理器” 或 “控制中心” 界面。

可用下列方法使用 DB2 Spatial Extender GIS:

- 用地理浏览器以图形方式显示信息；例如，“环境系统研究学会” (ESRI) 提供的 ArcExplorer
- 显式地从 “DB2 控制中心” 或 “命令行处理器” 提交 SQL 查询
- 从应用程序中提交 SQL 查询

创建和使用 DB2 Spatial Extender GIS 要执行的任务

本节概述用来创建和使用 DB2 Spatial Extender GIS 的任务。用来创建 GIS 的任务涉及设置 DB2 Spatial Extender 和开发 GIS 项目。用来使用 GIS 的任务涉及实现这些项目。此概述从设置 DB2 Spatial Extender 开始，然后转向开发和实现 GIS 项目。本节的末尾指示了概述中描述的任务在实际情况中如何变化。

设置 DB2 Spatial Extender

要设置 DB2 Spatial Extender:

1. 计划和进行准备（决定开发哪些 GIS 项目，决定对 DB2 Spatial Extender 启用什么数据库，选择管理 DB2 Spatial Extender 和开发项目的人员等）。
2. 安装 DB2 Spatial Extender。

3. 放入资源来支持 GIS 项目；例如:

   DB2 Spatial Extender 提供的资源

   这些资源包括系统目录、Spatial 数据类型、Spatial 函数（包括缺省地理编码器）等。设置这些资源的任务称作*对 Spatial 操作启用数据库*。

   用户和／或供应商开发的地理编码器。

   缺省地理编码器将美国地址转换为 Spatial 数据。您的组织和其他组织可提供将外国地址和其他种类的属性数据转换为 Spatial 数据的地理编码器。

有关安装 DB2 Spatial Extender 的指示，参见第15页的『第2章 安装 DB2 Spatial Extender』。有关使用 “控制中心” 放入资源的指示，参见第21页的『第3章 设置资源』。有关使用用于此用途的应用程序的指南，参见第49页的『第8章 编写 DB2 Spatial Extender 的应用程序』。有关说明设置 DB2 Spatial Extender 的整个过程的方案，参见第11页的『综合 Spatial 数据和传统数据的系统』。

开发和实现 GIS 项目

要开发和实现 GIS 项目:

1. 计划和进行准备（设置项目的目标，决定需要什么表和数据，确定要使用什么坐标系等）。
2. 确定要使用什么 Spatial 参考系。坐标值通常包括正整数、负数和小数。然而 DB2 Spatial Extender 必须以正整数格式存储所有坐标值。 *Spatial 参考系*是一组参数，这些参数定义如何将特定坐标系中的负数和小数转换为正整数，以便 DB2 Spatial Extender 可存储它们。在决定对 Spatial 列使用的坐标系之后，需要指定对该列进行必要转换时所用的参考系。若现有 Spatial 参考系满足您的要求，可使用它；否则可创建一个参考系。
3. 定义一个或多个包含 Spatial 数据的列，将它们注册到 DB2 Spatial Extender，并启用地理编码器来自动维护它们。

   注册 Spatial 列涉及将它记录在 DB2 Spatial Extender 目录中。从注册它时开始，它就被称作层，因为根据它生成的信息将给 GIS 为您创建的虚拟地理风景添加一个层。在注册它之后，可对它执行 Spatial 操作；例如，可填充它并在其上定义 Spatial 索引。
4. 填充 Spatial 列:
   - 对于需要地理编码器的项目，设置该地理编码器的参数。然后运行它，以使它在单个操作中对所有可用源数据进行地理编码并将产生的坐标装入某个层中。
   - 对于需要调入 Spatial 数据的项目，调入该数据。

5. 方便存取 Spatial 列。特别地，这涉及定义使 DB2 能快速存取 Spatial 数据的索引，和定义使用户能高效地检索相关数据的视图。在定义这样的视图之后，需要将它的 Spatial 列注册为层。
6. 生成和分析 Spatial 信息和相关的商业信息。这涉及查询 Spatial 列和相关的属性列。在这样的查询中，可包括返回各种信息的 DB2 Spatial Extender 函数；例如，两个地形之间的最小距离，或定义环绕某个地形的区域的坐标。有关返回这些坐标的函数 ST_Buffer 的信息，参见第6页的『使用其他 Spatial 数据作为源数据』和第174页的『ST_Buffer』。有关使用 Spatial 函数的查询的示例，参见第45页的『第7章 检索和分析 Spatial 信息』和第143页的『第14章 用于 SQL 查询的 Spatial 函数』。

有关使用“控制中心”来执行涉及开发 GIS 项目的任务的指示，参见:

- 第21页的『第3章 设置资源』
- 第29页的『第4章 定义 Spatial 列，将它们注册为层并启用地理编码器自动维护它们』
- 第35页的『第5章 填充 Spatial 列』
- 第43页的『第6章 创建 Spatial 索引』

有关使用“控制中心”实现 GIS 项目的指南，参见第45页的『第7章 检索和分析 Spatial 信息』。

有关使用应用程序开发和实现 GIS 项目的指南，参见第49页的『第8章 编写 DB2 Spatial Extender 的应用程序』。

有关说明整个过程的方案，参见第11页的『建立办事处和调整保险费的项目』。

任务组可如何变化

根据您的要求和使用的界面不同，创建和使用 DB2 Spatial Extender GIS 要执行的任务组可在内容和顺序上变化。例如，考虑定义要包含 Spatial 数据的列、将它们注册为层和启用地理编码器以自动维护它们的任务。使用“控制中心”，可从单个窗口一起执行这些任务。然而，若正从程序中调用存储过程，则可分别执行这些任务，并可随意安排执行它们的时间。

方案: 保险公司更新它的 GIS

本节提供一个方案来说明前一节中描述的任务组。

Safe Harbor 房地产保险公司的信息系统环境包括一个“DB2 通用数据库”系统和一个单独的 GIS 数据库管理系统。在一定程度上，查询可检索来自两个系统的数据组合。例如，DB2 表存储有关收入的信息，GIS 表存储公司的分支机构的位置。因此，有可能找出带来指定金额的收入的分支机构的位置。但不能综合来自两个

系统的数据（例如，用户不能将 DB2 列与 GIS 列连接），诸如查询优化这样的 DB2 服务不可用于 GIS。为了克服这些缺点，Safe Harbor 获取了 DB2 Spatial Extender 并建立了新的 GIS 开发部。下列各节描述该部门如何设置 DB2 Spatial Extender 和实现它的第一个项目。

综合 Spatial 数据和传统数据的系统

为了设置 DB2 Spatial Extender，Safe Harbor 的 GIS 开发部按如下步骤进行工作:

1. 该部门准备将 DB2 Spatial Extender 包括在它的 DB2 环境中。例如:
   a. 该部门的管理小组指派一个 Spatial 管理小组来安装和实现 DB2 Spatial Extender，并指派一个 Spatial 分析小组来生成和分析 Spatial 信息。
   b. 因为 Safe Harbor 的商业决策主要是根据客户的要求作出的，因此管理小组决定在包含有关公司客户的信息的数据库中安装 DB2 Spatial Extender。此信息的大部分存储在 CUSTOMERS 表中。

      为了便于引用所选数据库，GIS 开发部的成员将它称作 *GIS 数据库*。然而，他们知道它不仅仅是为 GIS 项目保留的；非 Spatial 应用程序也可象以前一样继续使用它。
2. Spatial 管理小组安装 DB2 Spatial Extender。
3. Spatial 管理小组设置 GIS 项目将需要的资源:
   - 该小组使用 “控制中心” 来提供对 Spatial 操作启用 GIS 数据库的资源。这些资源包括 DB2 Spatial Extender 目录、Spatial 数据类型、Spatial 函数等。
   - 因为 Safe Harbor 正准备将它的业务扩展到加拿大，所以 Spatial 管理小组开始请求加拿大的供应商提供将加拿大地址转换为 Spatial 数据的地理编码器。

建立办事处和调整保险费的项目

为了在 DB2 Spatial Extender 下实现第一个 GIS 项目，GIS 开发部按如下步骤进行工作:

1. 该部门准备开发该项目；例如:
   - 管理小组设置项目的目标:
     – 确定在何处建立新的分支机构。
     – 基于客户距离危险区（高交通事故率区、高犯罪率区、泛洪区、地震断层等）的远近调整保险费。
   - 该 GIS 项目将牵涉到美国的客户和办事处。因此，Spatial 管理小组决定:
     – 使用一些坐标系，这些坐标系可以准确定义 Safe Harbor 在美国开展了业务的地区的位置。
     – 使用缺省地理编码器，因为它设计成可对美国地址进行地理编码。
   - Spatial 管理小组决定达到项目的目标所需要的数据以及将包含此数据的表。

2. Spatial 管理小组使用 “控制中心” 创建两个 Spatial 参考系。一个参考系确定如何将定义办事处位置的坐标转换为 DB2 Spatial Extender 可存储的数据项。另一个参考系确定如何将定义客户住宅位置的坐标转换为 DB2 Spatial Extender 可存储的数据项。
3. Spatial 管理小组使用 “控制中心” 来定义用于包含 Spatial 数据的列、将这些列注册为层和启用地理编码器以自动维护它们:
   - 该小组给 CUSTOMERS 表添加 LOCATION 列。该表已包含客户的地址。缺省地理编码器将把它们转换为 Spatial 数据并将此数据装入 LOCATION 列。
   - 该小组创建 OFFICES 表来包含现在存储在单独的 GIS 中的数据。此数据包括 Safe Harbor 的分支机构的地址、由地理编码器从这些地址派生的 Spatial 数据和定义每个分支机构周围五英里半径范围内的区域的 Spatial 数据 。地理编码器生成的数据将进入 LOCATION 列。定义区域的数据将进入 SALES_AREA 列。
   - 该小组将两个 LOCATION 列和 SALES_AREA 列注册为层。
   - 该小组启用缺省地理编码器以自动维护两个 LOCATION 列。
4. Spatial 管理小组填充 CUSTOMER 表的 LOCATION 列、整个 OFFICES 表和一个新的 HAZARD_ZONES 表:
   - 该小组使用 “控制中心” 填充 CUSTOMER 表的 LOCATION 列:
     a. 该小组指示地理编码器仅在下列情况下将地址的 Spatial 数据插入 LOCATION 列: 地址与其在美国人口普查局的记录中的对应项之间的匹配必须具有 100% 的准确度。(随 DB2 Spatial Extender 一起交付一个地址文件，其中的地址由人口普查局提供。在地理编码器可将源数据中的地址转换为 Spatial 数据之前，地理编码器必须尝试将此地址与其在该文件中的对应项匹配。用户可指定匹配必须精确到多少百分比才将该 Spatial 数据放入表中。此百分比称作*精度*。)
     b. 该小组以批处理方式运行地理编码器，以便它可在一个操作中将表中的所有地址进行地理编码。使该小组沮丧的是，地理编码器拒绝了大约十分之一的地址!
     c. 该小组推测被拒绝的地址肯定是在人口普查局的记录中没有准确匹配的新地址。为了解决该问题，该小组将精度降低至 85。
     d. 该小组再次以批处理方式运行地理编码器。地址被拒绝的比率降至可接受的水平。
   - 该小组使用单独的 GIS 提供的实用程序，将分支机构数据装入文件。然后该小组使用 “控制中心” 将此数据从该文件调入到新的 OFFICES 表。
   - 该小组使用 “控制中心” 创建一个 HAZARD ZONES 表、将它的 Spatial 列注册为层并给它调入数据。该数据来自地图供应商提供的文件。

5. 通过使用“控制中心”，Spatial 管理小组使存取新层更为便利:
   - 该小组为它们创建索引。
   - 该小组创建一个视图来连接来自 CUSTOMERS 和 HAZARD ZONES 表的列。该小组然后将视图的 Spatial 列注册为层。
6. Spatial 分析小组运行查询以获取将帮助它达到初始目标的信息: 确定在何处建立新的分支机构和根据客户距离危险区的远近调整保险费。

第2章 安装 DB2 Spatial Extender

本章提供安装 DB2 Spatial Extender 的指示。将讨论下列主题:

- 『DB2 Spatial Extender 配置』
- 『系统需求』
- 第16页的『安装 DB2 Spatial Extender』
- 第18页的『验证安装』
- 第19页的『安装之后的考虑事项』
- 第19页的『下一步做什么？』

DB2 Spatial Extender 配置

DB2 Spatial Extender 系统由 “DB2 通用数据库”、DB2 Spatial Extender 和地理浏览器（例如 ArcExplorer）组成。通常，对 Spatial 操作启用的数据库位于服务器上。使用客户机应用程序通过 DB2 Spatial Extender 存储过程和 Spatial 查询来存取 Spatial 数据。另外，还可用地理浏览器查看 Spatial 数据。

图5说明 DB2 Spatial Extender 的体系结构。

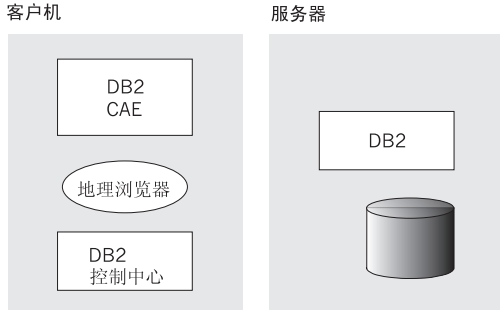


图 5. 客户机-服务器设置

系统需求

本节说明 DB2 Spatial Extender 的软件和硬件需求。

受支持的操作系统

DB2 Spatial Extender 可安装在下列操作系统上:

- AIX 4.2 或更高版本
- 带 “服务包 5” 的 Windows NT 4.0 或更高版本

必需的数据库软件

在安装 DB2 Spatial Extender 之前, 必须已在系统上安装和配置 DB2 软件。 表1列示 DB2 Spatial Extender 客户机部件和 DB2 Spatial Extender 服务器部件的数据库软件需求。

表 1. 最小软件需求

部件	软件
客户机	DB2 管理客户机版本 7.1[1]
服务器	下列项之一: • DB2 通用数据库企业版的版本 7.1 • DB2 通用数据库企业 – 扩展版的版本 7.1[2]

注:

1. 若不计划使用 “DB2 控制中心” 、存取 Spatial 数据的地理浏览器或 DB2 Spatial Extender 样本程序, 可使用 “DB2 管理客户机” 的低级版本。
2. 虽然可与 “DB2 通用数据库企业扩充版” 一起使用 DB2 Spatial Extender, 但不能在海量并行处理 (MPP) 环境中跨多个节点对 Spatial 索引进行分区。

磁盘空间需求

表2列示 DB2 Spatial Extender 的建议磁盘空间需求。

表 2. 磁盘空间需求

DB2 Spatial Extender 部件	磁盘空间
DB2 Spatial Extender 服务器库（包括 DB2 Spatial Extender 服务器库、地理编码器参考数据和文档）	600 MB
DB2 Spatial Extender 客户机支持（包括样本程序数据）	15 MB

安装 DB2 Spatial Extender

本节提供在 Windows NT 和 AIX 操作系统上安装 DB2 Spatial Extender 所需的信息。

开始之前

若尚未在客户机工作站上安装 “DB2 管理客户机”（包括 “控制中心” 和运行期客户机的管理工具）并安装 “DB2 通用数据库企业版” 或 “DB2 通用数据库企业扩充版”，则安装它们。安装它们的指示在相应*快速入门*丛书中。

在 Windows NT 系统上安装 DB2 Spatial Extender

要在 Windows NT 系统上安装 DB2 Spatial Extender:

1. 用拥有必需的管理许可权的用户名注册到系统。
2. 关闭任何其他程序。
3. 将 CD-ROM 插入驱动器。安装启动板打开。
4. 可选: 单击**发行版注意事项**检查 “DB2 Spatial Extender 发行版注意事项” 以获取对安装过程的任何更改，然后返回到 DB2 Spatial Extender 启动板。
5. 单击**安装**。
6. 响应安装程序的提示。联机帮助可用于指导您完成剩余的步骤。要调用联机帮助，单击**帮助**或按 F1 键。

当安装完成时，DB2 Spatial Extender 被安装在目录 %DB2PATH%（例如，c:\sqllib）下。

在 AIX 系统上安装 DB2 Spatial Extender

要在 AIX 系统上安装 DB2 Spatial Extender:

1. 注册为 root。
2. 将 CD-ROM 插入驱动器。
3. 将 CD-ROM 安装在 AIX 系统上。有关安装 CD-ROM 的信息，参考 *IBM DB2 通用数据库 UNIX 版快速入门*一书。
4. 通过输入下列命令进入安装 CD-ROM 的目录:

   ```
   cd /cdrom
   ```

   其中 *cdrom* 是 CD-ROM 驱动器在 AIX 上的安装位置。
5. 输入 **db2setup** 命令启动 DB2 安装程序。“安装 DB2 Spatial Extender” 窗口打开。

   注: 因为 DB2 安装程序会扫描系统以获取信息，所以它将缓慢启动。
6. 从 “安装 DB2 Spatial Extender” 窗口的产品列表中，选择想要安装的产品并单击**确认**。

   有关安装 DB2 Spatial Extender 的更多信息或帮助，单击**帮助**。

当安装完成时，DB2 Spatial Extender 被安装在 /usr/lpp/db2_07_01 目录中。

验证安装

在安装 DB2 Spatial Extender 之后，可通过使用 DB2 Spatial Extender 样本程序验证它的安装。在可运行样本程序之前，必须创建 SAMPLE 数据库并使样本程序成为可执行程序。

注: 确保使用 DB2 Spatial Extender make 实用程序生成的文件中指定的编译器。

要对 Windows NT 编译和运行样本程序:

1. 用具有管理员特权的用户 ID 注册。
2. 在命令行提示符处，输入 **db2sampl** 来创建 DB2 SAMPLE 数据库。
3. 在命令行提示符处输入下列命令:

   ```
   cd %DB2PATH%\samples\spatial
   ```

注: 为了执行 3 步骤并继续验证安装，必须拥有缺省 DB2 实例 (DB2–DB2)。

4. 输入 **make rungsedemo**。
5. 输入 **rungsedemo.exe**。
6. 检查程序运行时显示的错误信息和完成信息。

要对 AIX 编译和运行样本程序:

1. 注册为 root。
2. 创建或更新 DB2 实例。
3. 在命令行提示符处，输入 **db2sampl** 来创建 DB2 SAMPLE 数据库。
4. 在命令行提示符处输入下列命令:

   ```
   cd $DB2INSTANCE/sqllib/samples/spatial
   ```

注: 为了执行 4 步骤并继续验证安装，必须拥有您创建或更新的 DB2 实例。

5. 输入 **make rungsedemo**。
6. 输入 **rungsedemo**。
7. 检查程序运行时出现的错误和完成信息。

有关样本程序的信息，参见第49页的『第8章 编写 DB2 Spatial Extender 的应用程序』。

安装之后的考虑事项

在成功安装 DB2 Spatial Extender 之后，需要考虑:

- 下载 ArcExplorer
- 运行 DB2 实例更新实用程序

下载 ArcExplorer

IBM 将 ArcExplorer Java 3.0 作为样本程序分发，或者，您可从 ESRI Web 站点 http://www.esri.com 获得它。

有关安装和使用 ArcExplorer 的更多信息，参考 *Using ArcExplorer* 一书，也可从 ESRI Web 站点上获得该书。

ArcExplorer 需要 “Java® 2 运行期环境”（标准版或企业版）V1.2.2，可从 Sun Web 站点 http://java.sun.com 免费获取它。

要点: DB2 通用数据库 V7.1 与 IBM JDK 1.1.8 一起交付。当为 ArcExplorer 安装 JRE 1.2.2 时，将它放在与 DB2 不同的目录中。记住适当地设置 CLASSPATH 环境变量。

运行 DB2 实例更新实用程序 (db2iupdt)

db2iupdt 实用程序更新指定的 DB2 实例以:

- 允许该实例获取新的系统配置。
- 允许该实例存取与安装或除去某些产品选项相关的功能。

在 AIX 上，此实用程序位于 /usr/lpp/db2_07_01 中。若需要帮助，在命令行上输入 `db2iupdt -h` 以打开帮助菜单。在 Windows NT 操作系统上，db2iupdt 位于 \sqllib\bin 目录中。进入那个目录输入该命令。有关此命令的完整说明，参考 *IBM DB2 Universal Database Command Reference*。

下一步做什么?

在安装 DB2 Spatial Extender 之后，可使用 “DB2 控制中心” 设置 GIS 环境并开始使用 Spatial 信息。

要从 “控制中心” 调用 DB2 Spatial Extender:

1. 从 “控制中心” 窗口中，展开对象树直到找到您想要运行 DB2 Spatial Extender 的服务器下的**数据库**文件夹。
2. 单击**数据库**文件夹。数据库显示在该窗口右边的内容窗格中。

3. 用鼠标右键单击想要使用的数据库，并在弹出菜单中单击想要执行的 Spatial 操作。

有关从 “控制中心” 使用 DB2 Spatial Extender 的更多信息，参考:

- 第21页的『第3章 设置资源』
- 第29页的『第4章 定义 Spatial 列，将它们注册为层并启用地理编码器自动维护它们』
- 第35页的『第5章 填充 Spatial 列』
- 第43页的『第6章 创建 Spatial 索引』

第3章 设置资源

安装 DB2 Spatial Extender 之后，就可给数据库提供创建 Spatial 列和操纵 Spatial 数据时所需的资源了。本章概述这些资源，并描述使这些资源可用的两种任务：对 Spatial 操作启用数据库和创建 Spatial 参考系。

资源库存

可用来创建 Spatial 列和操纵 Spatial 数据的资源包括：

- *参考数据*：地址，DB2 Spatial Extender 检查该地址以验证想要进行地理编码的地址
- 对 Spatial 操作启用数据库的资源：存储过程、Spatial 函数及其他资源
- 用户和供应商提供的非缺省地理编码器
- Spatial 参考系

本节讨论对 Spatial 操作启用数据库的参考数据和资源。有关非缺省地理编码器的信息，参见第35页的『关于地理编码』。有关 Spatial 参考系的信息，参见第23页的『关于坐标系和 Spatial 参考系』。

参考数据

*参考数据*由美国人口普查局收集的美国最新地址组成。在缺省地理编码器可将数据库中的地址转换为坐标之前，它必须先将地址的一部分或全部与参考数据中的地址匹配。

当安装 DB2 Spatial Extender 时，参考数据变得可供您使用。有关此数据需要的磁盘空间量，参见第16页的『磁盘空间需求』。 要验证在 AIX 上正确装入了该数据，在 $DB2INSTANCE/sqllib/gse/refdata/ 目录中查找它。要验证在 Windows NT 上正确装入了该数据，在 %DB2PATH%\gse\refdata\ 目录中查找它。

对 Spatial 操作启用数据库的资源

安装 DB2 Spatial Extender 之后执行的第一个任务是对 Spatial 操作启用数据库。这涉及启动一个操作，该操作导致 DB2 Spatial Extender 给数据库装入下列资源：

- 存储过程。当从 “控制中心” 请求操作时，DB2 Spatial Extender 调用这些存储过程之一来执行该操作。

- Spatial 数据类型。必须对将存储 Spatial 数据的每个表列或视图列指定 Spatial 数据类型。有关更多信息，参见第29页的『关于 Spatial 数据类型』。
- DB2 Spatial Extender 的目录表和视图。某些操作依赖 DB2 Spatial Extender 目录。例如，在可填充具有 Spatial 数据类型的列之前，必须在目录中将该列注册为层。有关层的信息，参见第9页的『开发和实现 GIS 项目』。
- Spatial 索引类型。它允许您为层定义索引。
- Spatial 函数。可用多种方式使用这些函数来处理 Spatial 数据；例如，确定地形之间的关系和生成更多的 Spatial 数据。这些函数中的一个是缺省地理编码器。它将美国地址转换为坐标，然后将这些坐标插入 Spatial 列。 有关 Spatial 函数的更多信息，参见第109页的『第13章 几何图形和相关的 Spatial 函数』和第143页的『第14章 用于 SQL 查询的 Spatial 函数』。 有关缺省地理编码器的更多信息，参见第35页的『关于地理编码』。
- 称作 DB2GSE 的模式，它包含刚才列示的对象。

有关如何使用“控制中心”启动装入这些资源的指示，参见『对 Spatial 操作启用数据库』。有关在应用程序中使用例程来执行相同任务的指南，参见第49页的『第8章 编写 DB2 Spatial Extender 的应用程序』。

对 Spatial 操作启用数据库

要查明对 Spatial 操作启用数据库需要什么授权，参见第66页的『授权』。

要从“控制中心”对 Spatial 操作启用数据库:

1. 从“控制中心”窗口中，展开对象树直到找到您想要运行 DB2 Spatial Extender 的服务器下的**数据库**文件夹。
2. 单击**数据库**文件夹。数据库显示在该窗口右边的内容窗格中。
3. 用鼠标右键单击想要的数据库，并在弹出菜单中单击 **Spatial Extender —> 启用**。 DB2 Spatial Extender 给该数据库提供允许您创建和使用 Spatial 列和 Spatial 数据的资源。

提示项: 在可对 Spatial 操作启用数据库之前，必须将 DB2 Spatial Extender 安装在该数据库驻留的服务器上。

创建 Spatial 参考系

本节描述 Spatial 参考系和坐标系之间的关系，并说明如何从“控制中心”创建 Spatial 参考系。

关于坐标系和 Spatial 参考系

本节接着第5页的『Spatial 数据的性质』继续讨论坐标系。然后扩展第9页的『开发和实现 GIS 项目』中提供的 Spatial 参考系的定义。它还提供用于确定对 Spatial 参考系的参数指定什么值的指南。

坐标系、坐标和度量单位

可以按照覆盖特定地理区域的虚构网格来想象坐标系。示例包括覆盖地球的网格、覆盖一个国家的网格或覆盖州内某个地区的网格。区域中的每个地形位于东西网格线和南北网格线的交点。一个值称作 *X 坐标*，指示该位置位于东西网格线上何处。另一个值为 *Y 坐标*，指示该位置位于南北网格线上何处。两个值都使该位置参考网格的中心，即*原点*。

原点处的 X 和 Y 坐标都为零。从原点向东，X 坐标为正；从原点向西， X 坐标为负。类似地，从原点向北，Y 坐标为正；从原点向南，Y 坐标为负。为了说明此分布，考虑如下一般化示例：坐标系 A 包括覆盖大都会区域的网格。 X 坐标 7 指示从此网格的原点向东 7 个度量单位的位置。 X 坐标 -9.5 指示从该原点向西 9.5 个度量单位的位置。

Spatial 列中的每个数据项包括如下内容： (1) 定义地形的位置的 X 坐标和 Y 坐标，或 (2) 定义地形各个部分的位置或定义地形覆盖的区域的多个 X 和 Y 坐标。还可包括其他两种值：*Z 坐标*和*度量单位*。与 X 和 Y 坐标不一样，Z 坐标和度量单位在 DB2 Spatial Extender 中不用于定义位置或区域。而是只传递 GIS 应用程序所需的信息。Z 坐标通常指示地形的高度或深度。高于原点的 Z 坐标为正；低于原点的 Z 坐标为负。度量单位是数字的；它可传递任何种类的信息。例如，假设在 GIS 中表示油井。若要求应用程序处理表示地震数据的拍摄点 ID 的值，可将这些值存储为度量单位。

Spatial 参考系、偏移和比例因子

如『坐标系、坐标和度量单位』中所述，坐标可为负且可用小数表示。这对于度量单位也成立。然而，为了减少存储器额外开销， DB2 Spatial Extender 将每个坐标和度量单位存储为非负整数（即存储为正整数或零）。因此，必须将实际的负坐标和小数坐标和度量单位转换为非负整数，以便 DB2 Spatial Extender 可存储它们。另外，需要告知 DB2 Spatial Extender 如何进行该转换。通过设置某些参数执行此操作。将用于转换特定地理区域内的坐标和度量单位的参数设置统称为 *Spatial 参考系*。

可用下列方法创建 Spatial 参考系:

- 确定要表示的地形的最小负坐标和度量单位。（负值离 0 越远，它越小。X 坐标 –10 比 X 坐标 –5 小； –100 的度量单位比 –50 的度量单位小。）

- 指定*偏移因子*（或简称为*偏移*）为这样一些值：当从负坐标和度量单位减去这些值时，余下非负数。
- 指定*比例因子*为这样一些值：当将这些值与小数坐标和度量单位相乘时，产生其精度至少与小数坐标或度量单位相同的整数。例如，考虑具有四位精度的坐标：92.77。可将它乘以比例因子 100 来获取具有四位精度的整数：9277。

确定最小负坐标和度量单位

在为 Spatial 参考系设置参数之前，需要确定包含您想要查询的地形的地理区域中的最小负 X 坐标、Y 坐标、Z 坐标和度量单位。通过回答下列问题可查明这些值是什么:

- 在正在表示的地形中，有一些地形位于正在使用的坐标系原点的西边吗？若有，什么 X 坐标指示最西边的地形的西部边缘的位置？（该答案将是正处理的负 X 坐标的最小值。）例如，若正表示油井，而一些油井位于原点的西边，什么 X 坐标指示最西边的油井的位置？
- 有一些地形位于原点的南边吗？若有，什么 Y 坐标指示最南边的地形的南部边缘的位置？（该答案将是正处理的负 Y 坐标的最小值。）例如，若正表示油井，而一些油井位于原点的南边，什么 Y 坐标指示最南边的油井的位置？
- 若打算使用 Z 坐标定义深度，则哪个地形最深，且哪个 Z 坐标表示此地形的最低点？（该答案将是正处理的负 Z 坐标的最小值。）
- 若打算将度量单位包括在 Spatial 数据中，则有一些值为负吗？若有，最小负度量单位是什么？

在确定最小负坐标和度量单位之后，给每个值添加等于该值 5-10% 的量。例如，若最小负 X 坐标是 –100，则可给它添加 –5。本书将产生的数字称作*扩大的值*。

指定偏移因子

下一步，指定 DB2 Spatial Extender 应使用什么偏移因子将负坐标和度量单位转换为非负数:

- 在决定想要扩大的 X 值为多少之后，指定这样一个偏移，当从此值减去该偏移时余下零。 DB2 Spatial Extender 然后将从所有负 X 坐标减去此数以获得正值。DB2 Spatial Extender 也将从所有其他 X 坐标减去此数。

  例如，若扩大后的 X 值为 –105，则需要从它减去 –105 以获取 0。DB2 Spatial Extender 然后将从与正在表示的地形相关的所有 X 坐标减去 –105。因为这些坐标都不超过 –100，所以从该减法操作中得出的所有值都将为正。
- 类似地，指定这样一些偏移，当从扩大的 Y 值、扩大的 Z 值和扩大的度量单位减去这些偏移时余下 0。

从 X 坐标减去的偏移称作*辅助 X 坐标*。从 Y 坐标、Z 坐标和度量单位减去的偏移分别称作*辅助 Y 坐标*、*辅助 Z 坐标* 和*辅助 M 坐标*。有关从 “控制中心” 指定这些参数的指示，参见第26页的『从 “控制中心” 创建 Spatial 参考系』。

指定比例因子

下一步，指定 DB2 Spatial Extender 应使用什么比例因子将小数坐标和度量单位转换为整数:

- 指定这样一个比例因子，当将它乘以小数 X 坐标或小数 Y 坐标时产生一个 32 位的整数。建议使此比例因子为 10 的幂: 10 的一次幂 (10)、10 的二次幂 (100)、 10 的三次幂 (1000) 或更大的幂（若有必要的话）。要决定比例因子应为 10 的多少次幂:
  1. 确定哪些 X 和 Y 坐标为（或可能为）小数。例如，假设在将处理的各个 X 和 Y 坐标中，确定有三个坐标为小数: 1.23、5.1235 和 6.789。
  2. 记下具有最长小数精度的小数坐标。然后确定可对此坐标乘以 10 的多少次幂才能产生相等精度的整数。例如: 在当前示例中的三个小数坐标中，5.1235 具有最长的小数精度。将它乘以 10 的四次幂 (10000) 会产生整数 51235。
  3. 确定刚才描述的乘法产生的整数是否太长以致不能存储为 32 位数据项。51235 不太长。但假设除了 1.23、5.11235 和 6.789 之外，您的 X 和 Y 坐标范围包括第四个小数值 10006.789876。因为此坐标的小数位比其他三个坐标的小数位长，所以应将*此*坐标 — 而不是 5.1235 — 乘以 10 的幂。为了将它转换为整数，可将它乘以 10 的六次幂 (1000000)。但产生的值 10006789876 太长以致不能存储为 32 位数据项。若 DB2 Spatial Extender 尝试存储它，则结果将不可预测。

     要避免此问题，选择这样一个 10 的幂，当将它与原来的坐标相乘时，产生一个小数，DB2 Spatial Extender 可将该小数截断为可存储的整数，同时精度损失最小。在此情形下，可选择 10 的四次幂 (10000)。 10000 乘以 10006.789876 产生 100067898.76。DB2 Spatial Extender 会将此数字截断为 100067898，稍微降低了它的精度。
- 若要表示的地形具有小数 Z 坐标，遵循前面的过程确定这些坐标的比例因子。若这些地形与小数度量单位相关，则也遵循此过程确定这些度量单位的比例因子。

X 和 Y 坐标的比例因子称作 *XY 单位*。 Z 坐标和度量单位的比例因子分别称作 *Z 单位*和 *M 单位*。有关从 “控制中心” 指定这些参数的指示，参见第26页的『从 “控制中心” 创建 Spatial 参考系』。

从“控制中心”创建 Spatial 参考系

本节给出了从“控制中心”创建 Spatial 参考系的步骤的概述。概述后有如何完成每个步骤的详细信息。

执行这些步骤不需要任何授权。

从“控制中心”创建 Spatial 参考系的步骤的概述:

1. 打开“创建 Spatial 参考系”窗口。
2. 指示想要使用哪个坐标系。
3. 为想要创建的 Spatial 参考系指定标识符。
4. 确定什么范围的坐标和度量单位适用于您想要查询的地形。
5. 指定可用于将负的或小数坐标和度量单位转换为 DB2 Spatial Extender 可存储的数据项的值。
6. 告知 DB2 Spatial Extender 创建想要的 Spatial 参考系。

从“控制中心”创建 Spatial 参考系的详细步骤:

1. 打开“创建 Spatial 参考系”窗口。
   a. 从“控制中心”窗口中，展开对象树直到找到您想要运行 DB2 Spatial Extender 的服务器下的**数据库**文件夹。
   b. 单击**数据库**文件夹。数据库显示在该窗口右边的内容窗格中。
   c. 用鼠标右键单击您对 Spatial 数据启用的数据库，并在弹出菜单中单击 **Spatial Extender —> Spatial 参考系**。 “Spatial 参考系”窗口打开。
   d. 从“Spatial 参考系”窗口中，单击**创建**。 “创建 Spatial 参考系”窗口打开。
2. 从“创建 Spatial 参考系”窗口中，使用**坐标系**字段指示所要使用的坐标系。
3. 为要创建的 Spatial 参考系指定标识符。
   - 在**名称**字段中，为该系统输入一个 1 到 64 个字符的名称。

     限制: 不要指定另一 Spatial 参考系的名称。数据库中的任何两个 Spatial 参考系不能具有相同名称。
   - 在 ID 字段中，输入一个数字标识符。它必须为整数。

     限制: 不要指定另一 Spatial 参考系的 ID。数据库中的任何两个 Spatial 参考系不能具有相同 ID。
4. 使用“控制中心”外面的媒体 — 例如，纸张或白板 — 可确定适用于要表示的地形的最小负坐标和度量单位。有关如何执行此操作的指南，参见第24页的『确定最小负坐标和度量单位』。

5. 从“创建 Spatial 参考系”窗口中，指定将负的或小数坐标和度量单位转换为 DB2 Spatial Extender 支持的数据项（即转换为 32 位非负整数）的值。
   a. 指定将负的或小数 X 坐标转换为非负整数的值:
      - 在**偏移**列中最接近 **X** 的字段中，指定辅助 X 坐标:
        – 若在第26页的4步骤中标识的 X 坐标范围内的任何值为负，则输入一个辅助 X 坐标，当从最小负坐标减去此值时，余下一个正数。有关指南，参见第24页的『指定偏移因子』。
        – 若所有 X 坐标为非负数，则输入为 0 的辅助 X 坐标。
      - 在**比例因子**列中，在 **X** 的最右边的字段中指定 XY 单位。此 XY 单位应为这样一个值: 当将它乘以任何小数 X 坐标或小数 Y 坐标时，产生一个可存储为 32 位数据项的整数，且精度损失最小。有关指南，参见第25页的『指定比例因子』。

        在 **X** 的最右边的字段中指定 XY 单位之后，它也将出现在 **Y** 的最右边的字段中。
   b. 指定一个这样的辅助 Y 坐标，它将允许 DB2 Spatial Extender 将负 Y 坐标转换为正值。在**偏移**列中最接近 **Y** 的字段中执行此操作:
      - 若在第26页的4步骤中标识的 Y 坐标范围内的任何值为负，则输入这样一个辅助 Y 坐标，当从最小负坐标减去此值时，余下一个正数。有关指南，参见第24页的『指定偏移因子』。
      - 若所有 Y 坐标都为正，输入为 0 的辅助 Y 坐标。
   c. 若要将 Z 坐标包括在 Spatial 数据中，指定将负的或小数 Z 坐标转换为非负整数的值:
      - 在**偏移**列中最接近 **Z** 的字段中，输入一个辅助 Z 坐标:
        – 若在第26页的4步骤中标识的 Z 坐标范围内的任何值为负，则输入这样一个辅助 Z 坐标，当从最小负坐标减去此值时，余下一个正数。有关指南，参见第24页的『指定偏移因子』。
        – 若所有 Z 坐标都为非负数，则输入为 0 的辅助 Z 坐标。
      - 在**比例因子**列中，在 **Z** 的最右边的字段中指定 Z 单位。此 Z 单位应为这样一个值: 当将它乘以任何小数 Z 坐标时，产生一个可存储为 32 位数据项的整数，且精度损失最小。有关指南，参见第25页的『指定比例因子』。
   d. 若打算将度量单位包括在 Spatial 数据中，指定将负或小数度量单位转换为非负整数的值:
      - 在**偏移**列中最接近**线性**标号的字段中，输入辅助 M 坐标:

- 若在第26页的4步骤中标识的度量单位的范围内的任何值为负，则输入这样一个辅助 M 坐标，当从最小负度量单位减去此值时，余下一个正数。有关指南，参见第24页的『指定偏移因子』。
- 若所有度量单位为正，输入为 0 的辅助 M 坐标。

• 在**比例因子**列中，在**线性**标号的最右边的字段中输入 M 单位。此 M 单位应为这样一个值：当将它乘以任何小数度量单位时，产生可存储为 32 位数据项的整数，且精度损失最小。有关指南，参见第25页的『指定比例因子』。

6. 单击**确认**以创建想要的 Spatial 参考系。

第4章 定义 Spatial 列，将它们注册为层并启用地理编码器自动维护它们

在为 DB2 Spatial Extender GIS 设置资源之后，就可以创建将包含 Spatial 数据的对象了。例如，若需要新表来包含 Spatial 数据，则可定义它们，给想要数据进入的列指定 Spatial 数据类型。若需要给现存表添加 Spatial 列，也可执行那样的操作。

当给新表或现存表提供 Spatial 列时，需要将此列注册为层。另外，若计划让地理编码器填充该列，可在将该列注册为层时，启用该地理编码器来自动维护它。用以下方法进行这种启用: DB2 Spatial Extender 定义一些触发器，将这些触发器编码为无论何时 Spatial 列的相应属性列接收新的或更新的数据时就调用地理编码器。当地理编码器被调用时，它将新的或更新的数据转换为 Spatial 数据，并将此 Spatial 数据放入 Spatial 列。

在为表定义 Spatial 列之后，若选择，可基于此表列创建视图列。在将该表列注册为层之后，必须将该视图列定义为层。

本章讨论可给 Spatial 列指定的数据类型的性质和用法。然后，本章说明如何使用“控制中心”为表定义 Spatial 列，如何将此列注册为层和如何启用地理编码器来维护它。最后，本章说明如何使用“控制中心”将视图列注册为层。

关于 Spatial 数据类型

本节介绍 Spatial 列所需要的数据类型，并为如何选择 Spatial 列的数据类型提供指南。

当对 Spatial 操作启用数据库时，DB2 Spatial Extender 给该数据库提供结构化数据类型的层次结构。 第30页的图6显示了此层次结构。在此图中，可例示的类型具有白色背景；不可例示的类型具有阴影背景。

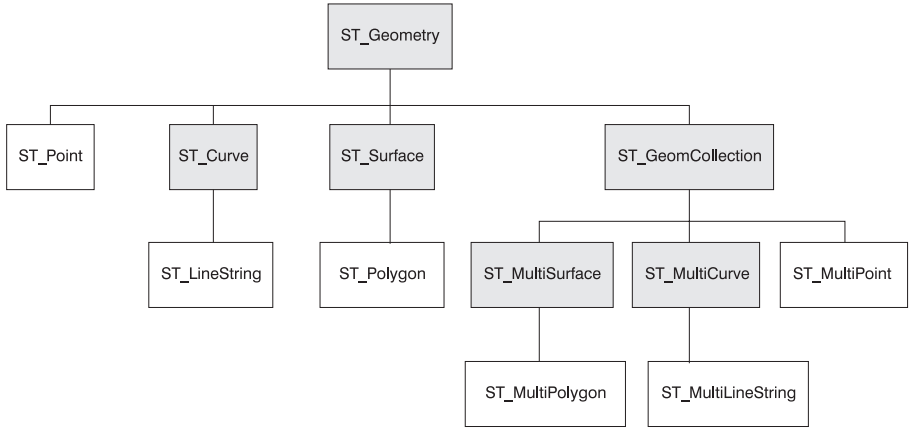


图 6. Spatial 数据类型的层次结构. 在白色框中命名的数据类型是可例示的。在阴影框中命名的数据类型是不可例示的。

图6中的层次结构包括:

- 某些地形的数据类型，可认为这些地形形成了单个单元；例如，单个住宅和孤立的湖泊。
- 由多个单元或组分组成的地形的数据类型；例如，公路系统和山脉。
- 所有种类的地形的数据类型。

单个单元的地形的数据类型

使用 ST_Point、ST_Linestring 和 ST_Polygon 存储定义可理解为形成单个单元的地形占据的空间的坐标:

- 当想要指示由分离的地形占据的空间中的点时，使用 ST_Point。地形可以很小，比如水井；可以很大，比如城市；或中等大小，比如建筑群或公园。在每种情况下，空间中的点可位于东西坐标线（例如纬线）和南北坐标线（例如经线）的交点。 ST_Point 数据项包括定义这种交点的值 — X 坐标和 Y 坐标。 X 坐标指示交点在东西线上的位置；Y 坐标指示交点在南北线上的位置。
- 对定义由线性地形占据的空间的坐标使用 ST_Linestring；例如，街道、运河和管道线。
- 当想要指示多边地形覆盖的空间范围时，使用 ST_Polygon；例如，福利区、森林或野生动物的栖息地。 ST_Polygon 数据项由定义这种地形的周边的坐标组成。

  在某些情况下，可对同一地形使用 ST_Polygon 和 ST_Point。例如，假设您需要关于几套公寓的 Spatial 信息。若想要表示每套公寓所在空间中的点，使用 ST_Point 来存储定义每个这种点的 X 和 Y 坐标。在另一方面，若想要表示每套公寓覆盖的区域，使用 ST_Polygon 来存储定义每个这种区域的周边的坐标。

多单元地形的数据类型

使用 ST_MultiPoint、ST_MultiLineString 和 ST_MultiPolygon 来存储由多个单元组成的地形所占据的空间的坐标:

- 当想要表示由分离的单元组成的地形和想要指示每个组分占据的空间中的点时，使用 ST_MultiPoint。 ST_MultiPoint 数据项包括定义这种地形的每个组分的位置的 X 和 Y 坐标对。例如，考虑这样一个表，它的行表示群岛，它的列包括一个 ST_MultiPoint 列。此列中的每个数据项包括定义每个群岛中岛屿的位置的 X 和 Y 坐标对。
- 当想要表示由线性单元组成的地形和需要关于每个单元占据的空间的信息时，使用 ST_MultiLineString。ST_MultiLineString 数据项由定义这种空间的坐标组成。例如，考虑这样一个表，它的行表示江河系统，它的列包括一个 ST_MultiLineString 列。此列中的每个数据项包括在每个系统中定义江河路径的多组坐标。
- 当想要表示由多边单元组成的地形和需要关于每个单元占据的空间的信息时，使用 ST_MultiPolygon。例如，考虑这样一个表，它的行表示中西部的县，它的列包括一个 ST_MultiPolygon 列。此列包含关于农田的信息。特别地，该列中的每个数据项包括定义特定县内农田的周边的多组坐标。

所有地形的数据类型

当不能确切知道要使用哪种数据类型时，可使用 ST_Geometry。因为 ST_Geometry 是其他数据类型所属的层次的根，所以 ST_Geometry 列可存储可在指定了其他数据类型的列中存储的任何或所有值。

注意: 缺省值地理编码器可将地址转换为 ST_Point 或 ST_Geometry 数据项。因此，若计划使用此地理编码器填充 Spatial 列，必须对此列指定 ST_Point 或 ST_Geometry 数据类型。

为表定义 Spatial 列，将此列注册为层并启用地理编码器来维护它

本节给出了为表定义 Spatial 列、将此列注册为层并启用地理编码器来维护它的步骤的概述。概述后有如何完成每个步骤的详细信息。

要查明将表列注册为层所需要的授权，参见第79页的『授权』。要查明启用地理编码器维护此列所需要的授权，参见第63页的『授权』。

为表定义 Spatial 列、将此列注册为层并启用地理编码器来维护它的步骤的概述:

1. 若 Spatial 列将是新表的一部分，则创建此表。
2. 打开 “创建 Spatial 层” 窗口。

3. 给表添加 Spatial 列并指示想要将此列注册为层；或指示想要将现存列注册为层。
4. 指示将对该层使用哪个 Spatial 参考系。
5. 若该层将包含调入的数据或从另一 Spatial 列生成的数据，则告知 DB2 Spatial Extender 创建该层。
6. 若该层将包含从属性数据派生的数据:
   a. 指定哪列或哪些列包含此属性数据。
   b. 指示想要启用地理编码器来维护该层。
   c. 告知 DB2 Spatial Extender 创建该层。

为表定义 Spatial 列、将此列注册为层并启用地理编码器来维护它的详细步骤:

1. 若该 Spatial 列将是新表的一部分，则创建此表:
   - 使用您选择的界面（例如“控制中心”或“命令行处理器”）创建该表。
   - 若计划使用地理编码器，则包括一至十列供地理编码器操作。地理编码器不能接受十列以上的数据作为输入。
   - 包括将要注册为层的 Spatial 列，或在3步骤中定义此列。

   若想要使用现存表，则转至下一步骤。
2. 打开“创建 Spatial 层”窗口。
   a. 从“控制中心”窗口中，展开对象树直到找到用于 Spatial 操作的数据库中表的**表**文件夹。
   b. 单击**表**文件夹。一些表显示在窗口右边的内容窗格中。
   c. 用鼠标右键单击想要的表并单击弹出菜单中的 **Spatial Extender —> Spatial 层**。 “Spatial 层”窗口打开。
   d. 从“Spatial 层”窗口中，单击**创建**。“创建 Spatial 层”窗口打开。
3. 从“创建 Spatial 层”窗口中，或给表添加 Spatial 列，指示想要将此列注册为层；或指示想要将现存列注册为层。
   - 若想要给表添加 Spatial 列并将此列定义为层:
     a. 在**层列**字段中，为该列输入名称。
     b. 在**列类型**字段中，选择或输入想要该列具有的数据类型。有关可允许的数据类型的讨论，参见第29页的『关于 Spatial 数据类型』。
   - 若想要将现存列定义为层，在**层列**字段中选择它。

     限制: 不要选择已定义为层的列。
4. 在**Spatial 参考系名称**字段中，指定将用于该层的 Spatial 参考系的名称。

5. 若想要该层包含调入的数据或从另一 Spatial 列生成的数据，单击**确认**以注册它。
6. 若想要该层包含从属性数据派生的数据:
   a. 指定哪列或哪些列包含此属性数据:
      1) 在**可用列**框中选择这些列。最多可选择十列。
      2) 单击 **>** 按钮和 / 或 **>>** 按钮，以在**选择的列**框中列示选择的列。
   b. 若想要启用地理编码器来维护该层:
      1) 选择**启用自动地理编码器**校验框。
      2) 在**名称**字段中，选择想要使用的地理编码器的名称。
      3) 在**精度级别**字段中，以百分比的形式指定一个程度，输入记录必须与参考数据中的对应记录匹配到该程度才能被处理。此百分比称作*精度*。例如，假设地理编码器读取包含地址 557 Bailey, San Jose 94120 的输入记录。若精度为 100 且此地址与它在参考数据中的对应项之间的匹配不是 100% 的准确，地理编码器将拒绝它。若精度为 75 且该记录与它的参考数据对应项之间的匹配准确度至少在 75% 以上，地理编码器将处理它。
      4) 若该地理编码器是供应商提供的，则使用**特性**框指定想要使用的任何供应商提供的地理编码参数。
   c. 单击**确认**将所选列注册为层，并（若已请求的话）启用地理编码器来维护该列。

将视图列注册为层

要查明将视图列注册为层所需要的授权，参见第79页的『授权』。

要将视图列注册为层:

1. 打开“创建 Spatial 层”窗口。
   a. 从“控制中心”窗口中，展开对象树直到找到用于 Spatial 操作的数据库中视图的**视图**文件夹。
   b. 单击**视图**文件夹。一些视图显示在该窗口右边的内容窗格中。
   c. 用鼠标右键单击想要的视图并单击弹出菜单中的 **Spatial Extender —> Spatial 层**。 “Spatial 层”窗口打开。
   d. 从“Spatial 层”窗口中，单击**创建**。 “创建 Spatial 层”窗口打开。
2. 使用**层列**框指定想要注册为层的列。
3. 在**基础 Spatial 层**字段中，指定所选视图列基于的表列的名称。必须已将此表列注册为层。

4. 单击**确认**以将指定的视图列注册为层。

第5章 填充 Spatial 列

在将 Spatial 列注册为层后，即可随时向这些列提供 Spatial 数据。如第6页的『Spatial 数据的来源』中所述，提供此数据有三种方法：使用某一称作地理编码器的函数，从属性数据中派生此数据；使用其他函数从其他 Spatial 数据派生此数据；或从文件中调入此数据。本章：

- 介绍地理编码并说明如何使用“控制中心”以批处理方式对属性数据进行地理编码
- 讨论调入与调出数据并说明如何使用“控制中心”将数据调入和调出 GIS

有关能从现存 Spatial 数据中派生新的 Spatial 数据的函数的更多信息，参见第133页的『从现存几何图形生成新几何图形的函数』。

使用地理编码器

本节描述地理编码过程并说明如何从“控制中心”以批处理方式运行地理编码器。

关于地理编码

本节区分地理编码器与其源的基本差别。本节将描述了运行地理编码器的两种方式，并介绍在计划使用地理编码器时要考虑的因素。

使用 DB2 Spatial Extender，您可以：

- 使用与 DB2 Spatial Extender 一起提供的缺省地理编码器。
- 插入第三方供应商开发的地理编码器。
- 插入自己的地理编码器。

缺省地理编码器对美国地址进行地理编码，并可以将它们转换为 ST_Point 数据或 ST_Geometry 数据。若需要存储其他 Spatial 数据类型的数据，则可以插入地理编码器来生成这样的数据。若需要表示美国以外的地点或没有地址的地点（例如在土质方面不同的农田）的 Spatial 数据，您也可以插入地理编码器来满足该需要。

在可使用插入的地理编码器之前，它必须被注册。用户与供应商可以使用 db2gse.gse_register_gc 存储过程注册它。它不能从“控制中心”注册。有关 db2gse.gse_register_gc 的信息，参见第77页的『db2gse.gse_register_gc』。有关使用 DB2 Spatial Extender 存储过程的一般信息，参见第57页的『第9章 存储过程』。

地理编码器以两种方式运行：

- 以*批处理方式*运行时，它尝试在单个操作中将 Spatial 列中所有现存的源数据转换为 Spatial 数据，并用那些数据填充该列。您可以从“运行地理编码器”窗口中启动此操作。或者，您可以编写一个应用程序来调用 db2gse.gse_run_gc 存储过程，在该应用程序中启动它。
- 以*增量方式*运行时，地理编码器在表中插入了新数据或更新某数据时对该数据进行转换，并将产生的 Spatial 值放置在某个列中，以使该列保持最新。该功能是通过“插入触发器”和“更新触发器”来激活的，您可以通过“创建 Spatial 层”窗口请求这些触发器。或者，您可以编写一个应用程序来调用 db2gse.gse_enable_autogc 存储过程，在该应用程序中请求它们。

  增量地理编码也称为*自动地理编码*。

当计划使用地理编码器时，您可能会考虑以下因素:

1. 使用“控制中心”时，在使用“运行地理编码器”窗口之前，一般使用“创建 Spatial 层”窗口。这指的是在启动批处理地理编码之前，您可以让 DB2 Spatial Extender 为以增量方式运行地理编码而设置触发器。因此，可以在以批处理方式运行地理编码之前以增量方式运行地理编码。以批处理方式处理所有源数据时，地理编码器将对它以增量方式处理过的相同数据进行地理编码。这种冗余不会导致重复（当产生 Spatial 数据两次时，第二次产生的数据将覆盖第一次的数据）。但它可能会使性能降低。其中一种解决方法是，将设置触发器延迟到进行批处理地理编码之后。
2. 若准备以批处理方式进行地理编码时设置了触发器，则建议释放它们，直到批处理地理编码结束。可从“运行地理编码器”窗口释放它们，或编写一个应用程序来调用 db2gse.gse_disable_autogc 存储过程，在该应用程序中释放它们。若使用“运行地理编码器”窗口， DB2 Spatial Extender 在地理编码结束时自动重新激活它们。若使用 db2gse.gse_disable_autogc 存储过程，可通过调用 db2gse.gse_enable_autogc 存储过程重新激活它们。
3. 若想要以批处理方式运行地理编码器来填充具有索引的 Spatial 列，应先禁用或卸下该索引。否则，若在地理编码器运行时索引仍可操作，性能将严重降低。若正在使用“控制中心”，则可从“运行地理编码器”窗口禁用该索引。当地理编码结束时， DB2 Spatial Extender 将自动重新启用该索引。若正使用应用程序，可用 SQL DROP 语句卸下该索引。若这样做，一定要记下该索引的参数，以便可在批处理地理编码结束之后重新创建它。
4. 当地理编码器读取源数据的记录时，它尝试将该记录与参考数据中的对应项进行匹配。为了让地理编码器处理该记录，匹配必须准确到某种程度（称作*精度*）。例如，精度 85 指的是源记录与它在参考数据中的对应项之间的匹配必须至少为 85%，才能使该源记录被处理。

   您指定精度应为什么值。要知道可能需要调整它。例如，假设精度为 100。若许多源记录包含比参考数据更新的地址，则这些记录和参考数据之间的 100% 准

确的匹配将是不可能的。结果，地理编码器将拒绝这些记录。总的来说，若地理编码器产生的 Spatial 数据似乎不够或大部分不准确，可尝试通过更改精度并再次运行地理编码器来解决此问题。

以批处理方式运行地理编码器

本节给出了从“控制中心”以批处理方式运行地理编码器的步骤的概述。概述后有如何完成每个步骤的详细信息。

要查明以批处理方式运行地理编码器所需要的授权，参见第84页的『授权』。

以批处理方式运行地理编码器的步骤的概述:

1. 打开“运行地理编码器”窗口。
2. 指示想要使用哪个地理编码器。
3. 禁用可能影响地理编码器性能的对象。
4. 指定在 DB2 发出确认之前要对多少记录进行地理编码。
5. 指示想要地理编码器如何运行。
6. 告诉 DB2 Spatial Extender 运行地理编码器。

以批处理方式运行地理编码器的详细步骤:

1. 打开“运行地理编码器”窗口。
   a. 从“控制中心”窗口中，展开对象树直到找到对 Spatial 操作启用的数据库中的**表**文件夹。
   b. 单击**表**文件夹。一些表显示在窗口右边的内容窗格中。
   c. 在内容窗格中用鼠标右键单击想要的表，并单击弹出菜单中的 **Spatial 层**。“Spatial 层”窗口打开。
   d. 从“Spatial 层”窗口:
      1) 选择在想要填充的列上定义的层。
      2) 单击**运行地理编码器**按钮。“运行地理编码器”窗口打开。
2. 若想要使用缺省地理编码器，则让显示此缺省地理编码器的名称的**名称**框保持原样。否则，使用该框选择想要的地理编码器。
3. 禁用可能影响地理编码器性能的对象:
   - 若想要填充的列具有索引，则选择**在地理编码过程中暂时禁用 Spatial 索引**校验框。
   - 若已设置触发器来为此列激活增量地理编码，则选择**在地理编码过程中暂时禁用 Spatial 触发器**校验框。

当在“运行地理编码器”窗口上单击**确认**时，将自动重新启用索引和触发器。

4. 使用**落实范围**旋钮指定在 DB2 执行落实之前对多少记录进行地理编码。例如，若想要 DB2 一次落实 100 个已进行地理编码的记录，则指定数字 100。

   提示: 若想要 DB2 仅在处理所有记录之后才执行落实，则指定零。

5. 使用**地理编码器参数**组框中的字段，指示想要地理编码器如何操作:
   - 使用**精度级别**旋钮，以百分比形式指定源记录和它们的参考数据对应项之间的匹配应准确到什么程度。有关精度的更多信息，参见第35页的『关于地理编码』。
   - 若正在使用供应商提供的地理编码器，且想要使用它支持的特性，则使用**特性**框设置这些特性。
   - 若想要仅对所选表中行的子集进行地理编码，则使用 **WHERE 子句**框编写将为想要的行指定标准的 SELECT WHERE 子句。此子句可引用表中的任何列。

     仅输入标准。省略关键字 WHERE。例如，若表具有 STATE 列，且想要仅对此列中包含值 MA 的那些行进行地理编码，输入:

     ```
     STATE='MA'
     ```

6. 单击**确认**以运行地理编码器。

调入和调出数据

本节描述调入和调出数据的过程，并说明如何使用“控制中心”:

- 将数据从数据交换文件调入到新表或现存表
- 将数据从数据交换文件调入到现存表
- 将数据从表调出到数据交换文件

关于调入和调出

本节列示调入和调出 Spatial 数据的原因。它还讨论数据交换文件，它们用作调出源和调入目标之间的接口。

可使用 DB2 Spatial Extender 从数据交换文件调入 Spatial 数据以及将 Spatial 数据调出到数据交换文件。考虑下列方案:

- 您的 GIS 包含表示办事处、客户和其他商业有关事项的 Spatial 数据。您想要向此数据补充表示您所在机构的人文环境（城市、街道、观光点等）的 Spatial 数据。您想要的数据可从地图供应商那里获得。可使用 DB2 Spatial Extender 从供应商提供的数据交换文件调入该数据。

- 您想要将 Spatial 数据从 Oracle 系统迁移到您的 DB2 Spatial Extender GIS。通过使用 Oracle 实用程序将数据装入数据交换文件进行迁移。然后使用 DB2 Spatial Extender 将数据从此文件调入到您已对 Spatial 操作启用的数据库。
- 您想要使用 GIS 浏览器对客户显示可视的 Spatial 信息。浏览器仅需要要使用的文件；它不需要与数据库连接。可使用 DB2 Spatial Extender 将数据调出到数据交换文件，然后使用浏览器实用程序将数据装入到浏览器。

“控制中心”支持 DB2 Spatial Extender 的两种数据交换文件：形状文件和 ESRI_SDE 传送文件。形状文件通常用于调入来自文件系统的数据，并用于将数据调出到将装入文件系统的文件。 ESRI_SDE 传送文件通常用于调入来自 ESRI 数据库的数据。

将数据调入到新表或现存表

本节给出了将数据从形状文件或 ESRI_SDE 传送文件，调入到新表或现存表的步骤的概述。概述后有如何完成每个步骤的详细信息。

要查明调入形状数据所需要的授权，参见第75页的『授权』。要查明调入 ESRI_SDE 数据所需要的授权，参见第73页的『授权』。

将数据调入到新表或现存表的步骤的概述:

1. 打开“调入 Spatial 数据”窗口。
2. 指定包含将调入的数据的文件的路径、名称和格式。
3. 指定在每次落实之前调入多少记录。
4. 若想要将 Spatial 数据调入将要创建的表，则为此表提供名称并为将包含该数据的列提供名称。若将 Spatial 数据调入现存表，则指示哪一列将包含该数据。
5. 指定哪个 Spatial 参考系将与该数据相关。
6. 指定文件来收集调入失败的记录。
7. 告诉 DB2 Spatial Extender 调入数据，若从此窗口定义了一个表，则还要创建该表并将要包含该数据的列注册为层。

将数据调入到新表或现存表的详细步骤:

1. 打开“调入 Spatial 数据”窗口。
   a. 从“控制中心”窗口中，展开对象树直到找到正在运行 DB2 Spatial Extender 的服务器下的**数据库**文件夹。
   b. 单击**数据库**文件夹。一些数据库显示在该窗口右边的内容窗格中。
   c. 用鼠标右键单击想要将数据调入到的数据库，并在弹出菜单中单击 **Spatial Extender —> 调入 Spatial 数据**。“调入 Spatial 数据”窗口打开。
2. 指定包含将调入的数据的文件的路径、名称和格式:

a. 使用**文件名**字段指定路径和名称。

b. 使用**文件格式**框指定格式。格式可为:

Shape 这是缺省值。

ESRI_SDE

若指定此格式，**Spatial 参考系名称**字段缺省为与此格式相关的 Spatial 参考系的名称。

3. 使用**落实范围**字段，指定在每次落实之前想要调入的记录数。例如，要让 DB2 一次落实 100 个记录，则指定数字 100。

提示: 若想要 DB2 仅在处理所有记录之后才执行落实，则指定零。

4. 指定将包含数据的表和列。

a. 使用**层模式**框为数据将调入到的表指定模式。

b. 指定表和列:

- 若表尚不存在:
  1) 在**层表**字段中，为表输入名称。
  2) 在**层列**字段中，为将包含调入的数据的列输入名称。 DB2 Spatial Extender 将自动将此列注册为层。
- 若表已存在:
  1) 在**层表**字段中，指定该表。它必须已包含想要调入的数据进入的列。另外，必须已将此列注册为层。
  2) 在**层列**字段中，指定将包含调入的数据的列的名称。

5. 在 **Spatial 参考系名称**字段中，输入或选择将与此数据相关的 Spatial 参考系。（若数据将来自 ESRI_SDE 传送文件，则相关 Spatial 参考系的名称自动显示在该字段中。）

6. 在**异常文件**字段中，为新文件指定路径和名称，调入失败的记录可收集到该文件。稍后可修正这些记录并从此文件调入它们。

DB2 Spatial Extender 将创建此文件；不要指定已存在的文件。

7. 单击**确认**以调入该数据。另外，若为尚不存在的表提供名称，则将创建此表且将把包含该数据的列注册为层。另外，将创建您指定的异常文件。

将数据调入到现存表

本节给出了将数据从形状文件或 ESRI_SDE 传送文件调入到现存表的步骤的概述。概述后有如何完成每个步骤的详细信息。

要查明调入形状数据所需要的授权，参见第75页的『授权』。要查明调入 ESRI_SDE 数据所需要的授权，参见第73页的『授权』。

将数据调入到现存表的步骤的概述:

1. 打开“调入 Spatial 数据”窗口。
2. 指定包含将调入的数据的文件的路径和名称。
3. 指定在每次落实之前调入多少记录。
4. 指定将包含要调入的 Spatial 数据的列。
5. 指定哪个 Spatial 参考系将与此数据相关。
6. 指定文件来收集调入失败的记录。
7. 告知 DB2 Spatial Extender 调入数据，若指定了尚未创建的列，则还要创建此列并将它注册为层。

将数据调入到现存表的详细步骤:

1. 打开“调入 Spatial 数据”窗口。
   a. 从“控制中心”窗口中，展开对象树直到找到想要将数据调入到的数据库的**表**文件夹。
   b. 单击**表**文件夹。一些表显示在窗口右边的内容窗格中。
   c. 用鼠标右键单击要将数据调入到的表，并在弹出菜单中单击 **Spatial Extender —> 调入 Spatial 数据**。“调入 Spatial 数据”窗口打开。
2. 在**文件名**框中，指定包含将调入的数据的文件的路径和名称。
3. 使用**落实范围**框指定在每次落实之前想要调入的记录数。例如，要让 DB2 一次落实 100 个记录，则指定数字 100。

   提示: 若想要 DB2 仅在处理所有记录之后才执行落实，则指定零。
4. 指定将包含要调入的 Spatial 数据的列。
   - 若表中尚不存在该列，则使用**层列**框为该列输入名称。
   - 若该列已存在，则使用**层列**框选择或输入该列的名称。
5. 使用 **Spatial 参考系名称**框指定哪个 Spatial 参考系将与调入的数据相关。
   - 若将列添加到表，则输入或选择 Spatial 参考系的名称。
   - 若调入的数据将插入现存列，则让 **Spatial 参考系名称**框保持原样。该框显示缺省 Spatial 参考系的名称。
6. 在**异常文件**字段中，为新文件指定路径和名称，调入失败的记录可收集到该文件。稍后可修正这些记录并从此文件调入它们。

   DB2 Spatial Extender 将创建此文件；不要指定已存在的文件。

7. 单击**确认**以调入数据。另外，若指定尚不存在的列，则将创建此列并将它注册为层。另外，将创建您指定的异常文件。

将数据调出到形状文件

本节给出了将数据调出到形状文件的步骤的概述。概述后有如何完成每个步骤的详细信息。

要查明执行这些步骤所需要的授权，参见第71页的『授权』。

从形状文件调出数据的步骤的概述:

1. 打开“调出 Spatial 数据”窗口。
2. 指定包含将调出的 Spatial 数据的列。
3. 若想要调出数据行的子集，对 DB2 Spatial Extender 标识此子集。
4. 指定想要将数据调出到的文件的路径和名称。
5. 告知 DB2 Spatial Extender 调出该数据。

将数据调出到形状文件的详细步骤:

1. 打开“调出 Spatial 数据”窗口。
   a. 从“控制中心”窗口中，展开对象树直到找到包含 Spatial 数据的数据库中的**表**或**视图**文件夹。
   b. 单击**表**或**视图**文件夹。一些表或视图显示在该窗口右边的内容窗格中。
   c. 用鼠标右键单击包含将要调出的数据的表或视图，并在弹出菜单中单击 **Spatial Extender —> 调出 Spatial 数据**。“调出 Spatial 数据”窗口打开。
2. 在**层列**字段中，指定包含将调出的 Spatial 数据的列的名称。
3. 若想要调出表行的子集，则使用 **WHERE 子句**框输入一个 WHERE 子句，为想要的行指定标准。在此子句中，仅可引用将从中调出数据的表或视图中的列。

   仅输入标准。省略关键字 WHERE。例如，若表或视图具有 STATE 列，您想要仅对此列中包含 MA 值的那些行进行地理编码，则输入:

   ```
   STATE='MA'
   ```

4. 在**文件名**框中，指定要将数据调出到的文件的路径和名称
5. 单击**确认**以调出数据。

第6章 创建 Spatial 索引

本章说明如何使用“控制中心”为 Spatial 数据创建索引。

在给 Spatial 列填充数据之后，就可以创建 Spatial 索引了。一般的索引结构，比如 B 树，对表数据执行线性一维排序。已对 Spatial 操作启用的表数据不会被存储为单个项目，而是存储为二维项目。例如，Spatial 几何图形（比如多边形）由一个 Spatial 列或层中的几个坐标值组成。因为 B 树索引不能处理 Spatial 数据类型，所以 DB2 Spatial Extender 创建了称作*网格索引*的专利索引技术。网格索引基于 B 树索引，已增强 B 树索引来处理二维数据和对 Spatial 列执行索引。网格索引支持三层，并设计成可基于大量的对象、大小和数据分布提供良好的性能。有关 Spatial 索引的更多信息，参见第101页的『第12章 Spatial 索引』。

要查明创建 Spatial 索引所需要的授权，参见第67页的『授权』。

使用“控制中心”创建 Spatial 索引

要使用“控制中心”创建 Spatial 索引:

1. 在对象树中，选择**表**文件夹。所有现存表显示在内容窗格中。
2. 从内容窗格中，用鼠标右键单击想要为其创建索引的表，并在弹出菜单中单击 **Spatial Extender —> Spatial 索引**。 “Spatial 索引”窗口打开。
3. 从“Spatial 索引”窗口中，单击**创建**。 “创建 Spatial 索引”窗口打开。
4. 在**名称**字段中，输入想要创建的新 Spatial 索引的名称。

   注: 不需要指定模式。 DB2 Spatial Extender 将为您自动添加模式和创建全限定名。
5. 在**层列**字段中，选择要为其创建索引的层。

   层是定义或注册到 DB2 Spatial Extender 的 Spatial 列。
6. 在**网格大小**字段中，输入想要指定给每个字段的网格大小值。

   通过增加单元大小输入网格级: **最细**、**中等**和**最粗**。因此，第二级必须具有比第一级更大的单元大小，第三级必须具有比第二级更大的单元大小。

确定网格单元大小

确定正确的网格大小是通过试错法实现的。建议根据正在为其建立索引的对象的大约大小设置网格大小。设置得太小或太大的网格大小可能导致性能降低。设置得太小的大小会在索引搜索期间影响关键字／对象比。在这种情况下，会创建太多的关键字并返回大量的侯选项。对于设置得太大的网格大小，初始索引搜索返回少量的侯选项，但在最后的表扫描期间性能可能会降低。

有关选择网格单元大小和网格级数的更多信息，参见第108页的『选择网格单元大小』。

第7章 检索和分析 Spatial 信息

构建 Spatial 索引之后，就随时可以使用 Spatial 表。本章讨论与检索和分析 Spatial 数据相关的问题。本章概述了各种检索方法，并提供了使用 Spatial 函数进行表查询的示例。

执行 Spatial 分析的方法

可以通过在下列任何程序设计环境中使用 SQL 和 Spatial 函数来执行 Spatial 分析:

- 地理浏览器（例如 ESRI's ArcExplorer）。

  有关使用 ArcExplorer 的更多信息，参考 *Using ArcExplorer*，可以在 ESRI Web 站点 http://www.esri.com 找到它。
- 交互式 SQL 语句。
- 用户开发的应用程序（例如，ODBC、JDBC 和嵌入式 SQL）。

可以从“DB2 命令中心”、“DB2 命令窗口”或命令行处理程序中启动应用程序。

构建 Spatial 查询

本节讨论如何构建利用 Spatial 函数和谓词的 Spatial 查询。

Spatial 函数与 SQL

DB2 Spatial Extender 包括对 Spatial 数据执行各种操作的函数。本节中的示例显示如何使用 Spatial 函数来构建您自己的 Spatial 查询。

表3提供了 Spatial 函数列表和他们能执行的操作类型。

表 3. Spatial 函数与操作

函数类型	操作示例
计算	计算两点间的距离
比较	查找洪泛区内的所有客户
数据交换	把数据转换为受支持的格式
转换	将 5 英里的半径添加到一个点中

有关 Spatial 函数的更多信息，参见第109页的『第13章 几何图形和相关的 Spatial 函数』和第143页的『第14章 用于 SQL 查询的 Spatial 函数』。

示例1: 比较

下列查询查找客户与每个百货商店的平均距离。在此示例中使用的 Spatial 函数为: ST_Distance 和 ST_Within。

```
SELECT s.id, AVG(db2gse.ST_Distance(c.location,s.location))
FROM customers c, stores s
WHERE db2gse.ST_Within(c.location,s.zone)=1
GROUP BY s.id
```

示例2: 数据交换

下列查询查找那些驻住在圣弗朗西斯科海湾区的客户位置。此示例中使用的 Spatial 函数为: ST_AsText（数据交换）和 ST_Within。ST_AsText 将 c.location 列中的 Spatial 数据转换成 OGC TEXT 格式。

```
SELECT db2gse.ST_AsText(c.location,cordref(1))
FROM customers c
WHERE db2gse.ST_Within(c.location,:BayArea)=1
```

示例 3: 计算

下列查询查找长度超过 10.5 英里的所有街道。在此示例中所使用的 Spatial 函数为: ST_Length。

```
SELECT s.name,s.id
FROM street s
WHERE db2gse.ST_Length(s.path) > 10.5
```

示例 4: 转换

此查询查找驻住在洪泛区或距其边界 2 英里内的客户。在此示例中所使用的 Spatial 函数为: ST_Buffer（转换）和 ST_Within。

```
SELECT c.name,c.phoneNo,c.address
FROM customers c
WHERE db2gse.ST_Within(c.location,ST_Buffer(:floodzone,2))=1
```

Spatial 谓词与 SQL

专门有一组称作 Spatial 谓词的 Spatial 函数可以提高查询性能。执行 Spatial 谓词（如 ST_Overlaps，比较两个多边形看它们是否重叠）需要很多时间和内存。因此，使执行成本最小的优化技术很重要。根据本节后面描述的规则使用 Spatial 谓词时， DB2 查询优化器使用 Spatial 索引提高查询性能。有关 Spatial 谓词的更多信息，参见第122页的『谓词函数』。用于开发 Spatial 索引的 Spatial 谓词为:

- ST_Contains
- ST_Crosses
- ST_Disjoint

- ST_Distance
- ST_Envelope
- ST_Equals
- ST_Intersects
- ST_Overlaps
- ST_Touches
- ST_Within

有关所有 Spatial 函数和 Spatial 谓词的完整列表，参见第143页的『第14章 用于 SQL 查询的 Spatial 函数』。

索引开发规则

若您想使用 Spatial 谓词来优化 Spatial 查询，需要遵循下列规则:

- 必须在 WHERE 子句中使用谓词。
- 谓词必须在比较的左边。例如:

  ```
  WHERE db2gse.ST_Within(c.location,:BayArea)=1
  ```

- 相等性比较必须使用整常数 1。

  ```
  WHERE db2gse.ST_Within(c.location,:BayArea)=1
  ```

- 必须在谓词中使用 Spatial 列作为搜索目标，并且必须在那一列上创建 Spatial 索引。

索引开发示例

表4显示了创建 Spatial 查询来开发 Spatial 索引的正确方法和错误方法。

表 4. 索引开发规则

Spatial 查询	违反的规则

`SELECT *`
`FROM customers c`
`WHERE db2gse.ST_Within(c.location,:BayArea)=1`
	此示例中没有违反条件。

`SELECT *`
`FROM customers c`
`WHERE db2gse.ST_Distance(c.location,:SanJose)<10`
	此示例中没有违反条件。

`SELECT *`
`FROM customers c`
`WHERE db2gse.ST_Length(c.location)>10`
	必须在 WHERE 子句中使用谓词。（ST_Length 是 Spatial 函数，但不是谓词。）

`SELECT *`
`FROM customers c`
`WHERE 1=db2gse.ST_Within(c.location,:BayArea)`
	谓词必须在比较的左边。

表 4. 索引开发规则 (续)

Spatial 查询	违反的规则

`SELECT *`
`FROM customers c`
`WHERE db2gse.ST_Within(c.location,:BayArea)=2`
	相等性比较必须使用整常数 1。

`SELECT *`
`FROM customers c`
`WHERE db2gse.ST_Within(:SanJose,:BayArea)=1`
	必须在谓词中使用 Spatial 列作为搜索目标，并且必须在那一列上创建 Spatial 索引。 （SanJose 和 BayArea 不是 Spatial 列，因此不能具有与它们相关的索引。）

第8章 编写 DB2 Spatial Extender 的应用程序

本章说明如何使用 DB2 Spatial Extender 样本程序编写应用程序来使用和定制 Spatial 信息。包括下列主题:

- 使用样本程序
- 样本程序步骤

使用样本程序

DB2 Spatial Extender 样本程序使应用程序设计变得更容易。使用样本程序，您可以:

- 使例程 Spatial 过程自动化
- 将样本编码剪切并粘贴到您自己的应用程序中
- 了解创建和维护对 Spatial 操作启用的数据库一般所需要的步骤

使用样本程序来编码 DB2 Spatial Extender 的复杂任务，例如，编写一个使用数据库接口来调用 DB2 Spatial Extender 存储过程的应用程序。从样本程序中，您可以复制并定制您的应用程序。若不熟悉 DB2 Spatial Extender 的程序设计步骤，可以运行样本程序，它会详细显示出每一步骤。然而，首先必须创建样本程序。可以通过样本 make 实用程序生成的文件来创建。有关创建和运行样本程序的指示，参见第18页的『验证安装』。

样本程序步骤

第50页的表5显示样本程序步骤、相关存储过程和每一步骤的说明。用于调用存储过程的 C 函数显示在第50页的表5的“操作”列中，并用圆括号内将它括起。有关存储过程的更多信息，参见第57页的『第9章 存储过程』。样本程序基于第10页的『方案: 保险公司更新它的 GIS』中介绍的方案。

表 5. DB2 Spatial Extender 样本程序

样本程序步骤	操作	说明
启用 / 禁用 Spatial 数据库	1. 启用 Spatial 数据库 (gseEnableDB) 2. 禁用 Spatial 数据库 (gseDisableDB) 3. 启用 Spatial 数据库 (gseEnableDB)	1. 这是使用 DB2 Spatial Extender 所需要的第一步。已对 Spatial 操作启用的数据库有一组 Spatial 类型、一组 Spatial 函数、一组 Spatial 谓词、一种新的索引类型和一组管理表和视图。 2. 当已对错误数据库启用 Spatial 能力时，通常执行此步骤。当您禁用某个 Spatial 数据库时，就会除去一组 Spatial 类型、一组 Spatial 函数、一组 Spatial 谓词、一种新的索引类型和一组管理表和视图。 **注：** 若有创建的对象取决于由启用的数据库过程创建的对象，则禁用数据库将失效。例如，用类型 ST_Point 的 Spatial 列创建的表会使禁用数据库失效。之所以会出现此情况，是因为该表取决于类型 ST_Point，而类型 ST_Point 却是禁用数据库过程想要卸下的。 3. 与 1 一样。
注册 Spatial 参考系	1. 注册 CUSTOMERS 表的 LOCATION 列的 Spatial 参考系 (gseEnableSref) 2. 注册 OFFICES 表的 LOCATION 列的 Spatial 参考系 (gseEnableSref) 3. 注销 OFFICES 表的 LOCATION 列的 Spatial 参考系 (gseDisableSref) 4. 重新注册 OFFICES 表的 ZONE 列的 Spatial 参考系 (gseEnableSref(	1. 此步骤定义新的 Spatial 参考系 (SRS)，准备将其用来解释 CUSTOMERS 表的 Spatial 数据。Spatial 参考系包括某种格式的几何图形数据，可以将几何图形数据以该格式存储在对 Spatial 操作启用的数据库的列中。 在将 SRS 注册到某一特定层后，可将适合于那一层的坐标存储在相关的 CUSTOMERS 表列中。 2. 此步骤定义新的 Spatial 参考系 (SRS)，准备将其用来解释 OFFICES 层的 Spatial 数据。必须为每一个表层定义一个 SRS。OFFICES 表层可能需要一个不同于 CUSTOMERS 表层的相关 SRS。 3. 若给该层或 Spatial 列指定了错误的 SRS 参数，则执行此步骤。当您注销 OFFICES 表层的 SRS 时，就会除去具有其相关参数的定义。 4. 此步骤定义新的 Spatial 参考系 (SRS)，准备将其用来解释 OFFICES 层的 Spatial 数据。

表 5. DB2 Spatial Extender 样本程序 (续)

样本程序步骤	操作	说明
创建 Spatial 表	1. 通过添加 LOCATION 列改变 CUSTOMERS 表 (gseSetupTables) 2. 创建 OFFICES 表 (gseSetupTables)	1. CUSTOMERS 表表示在数据库中已经有存储了好几年的商业数据。 ALTER TABLE 语句添加类型为 ST_Point 的新列 (LOCATION)。此列将在后续步骤中通过对地址列进行地理编码来填充。 2. OFFICES 表表示: 在其他数据中，保险公司的每一个办事处的销售范围。整个表将在后续步骤中用非 DB2 数据库的属性数据填充。此步骤涉及从 SHAPE 文件中将属性数据调入至 OFFICES 表中。
注册 Spatial 层	1. 将 CUSTOMERS 表中的 LOCATION 列注册为层 (gseRegisterLayer) 2. 将 OFFICES 表的 ZONE 列注册为层 (gseRegisterLayer)	这些步骤将 LOCATION 列和 ZONE 列注册为 DB2 Spatial Extender 的层。在 DB2 Spatial Extender 实用程序（如地理编码器）可填充或存取 Spatial 列之前，需要先将其注册为层。
填充 Spatial 层	1. 对 CUSTOMERS 表的 LOCATION 列的地址数据进行地理编码 (gseRunGC) 2. 使用追加方式装入 OFFICES 表 (gseImportShape) 3. 使用创建方式装入 HAZARD_ZONE 表 (gseImportShape)	1. 此步骤通过调用地理编码器实用程序来执行批处理地理编码。当需要对表的有效部分进行地理编码或重新进行地理编码时，通常执行批处理地理编码。 2. 此步骤将以 SHAPE 文件的格式存在的 Spatial 数据装入 OFFICES 表。因为 OFFICES 表存在并且注册了 OFFICES/ZONE 层，所以装入实用程序将把新的记录追加到现存的表中。 3. 此步骤将以 SHAPE 文件的格式存在的 Spatial 数据装入 HAZARD_ZONE 层。因为该表和层不存在，所以装入实用程序在装入数据之前会创建该表并注册该层。

表 5. DB2 Spatial Extender 样本程序 (续)

样本程序步骤	操作	说明
启用 Spatial 索引	1. 启用 CUSTOMERS 表的 LOCATION 列的 Spatial 索引 (gseEnableIdx) 2. 启用 OFFICES 表的 ZONE 列的 Spatial 索引 (gseEnableIdx) 3. 启用 OFFICES 表的 LOCATION 列的 Spatial 索引 (gseEnableIdx) 4. 启用 HAZARD_ZONE 表的 BOUNDRY 列的 Spatial 索引 (gseEnableIdx)	这些步骤启用 CUSTOMERS、OFFICES 和 HAZARD_ZONE 表的 Spatial 索引。
启用自动地理编码	1. 启用 CUSTOMERS 表的 LOCATION 和 ADDRESS 列的自动编码 (gseEnableAutoGC)	此步骤打开地理编码器的自动调用。使用自动地理编码会使 CUSTOMERS 表的 LOCATION 和 ADDRESS 列相互同步，以便随后执行插入和更新操作。
插入 / 更新 CUSTOMERS 表	1. 插入具有不同街道的某些记录 (gseInsDelUpd) 2. 用新地址更新某些记录 (gseInsDelUpd)	这些步骤演示如何在 CUSTOMERS 表的 LOCATION 列上执行插入和更新操作。一旦启用自动地理编码，当在 LOCATION 列中插入或更新来自 ADDRESS 列中的信息时，会自动对该信息进行地理编码。在先前步骤中启用了此进程。
禁用自动地理编码	1. 禁用 CUSTOMERS 层的自动地理编码 (gseDisableAutoGC) 2. 禁用 CUSTOMERS 层的 Spatial 索引 (gseDisableIdxCustomersLayer)	在准备下一步骤（下一步骤包括对整个 CUSTOMERS 表的重新进行地理编码）时，这些步骤禁用地理编码器和 Spatial 索引的自动调用。若您正在装入大量的地理数据，建议您在装入数据之前禁用 Spatial 索引，在完成装入之后再启用它。
对 CUSTOMERS 表重新进行地理编码	1. 用较低的精度级 –90% 代替 100% 再次对 CUSTOMERS 层进行地理编码 (gseRunGC) 2. 重新启用 CUSTOMERS 层的 Spatial 索引 (gseEnableIdx) 3. 用较低的精度级 –90% 代替 100% 重新启用自动地理编码 (gseEnableAutoGC)	这些步骤再次以批处理的方式运行地理编码器，用新精度级重新启用自动地理编码，并重新启用 Spatial 索引和自动地理编码。当 Spatial 管理员发现地理编码过程中故障率较高时，建议执行此操作。若将精度级设置为 100%，则对地址进行地理编码可能会失败，因为在引用数据中找不到匹配的地址。通过减小精度级，地理编码器就比较容易找到匹配数据。以批处理的方式对表重新进行地理编码之后，会再次启用自动地理编码和 Spatial 索引以便于对 Spatial 索引和 Spatial 列进行增量维护，以便随后执行插入和更新操作。

表 5. DB2 Spatial Extender 样本程序 (续)

样本程序步骤	操作	说明
创建视图并将其 Spatial 列注册为视图层	1. 根据 CUSTOMERS 表与 HAZARD_ZONE 表的连接，创建视图 HIGHRISK_CUSTOMERS (gseCreateView) 2. 将视图的 Spatial 列注册为视图层 (gseRegisterLayer)	这些步骤创建视图并将其 Spatial 列注册为视图层。
执行 Spatial 分析	1. 查找客户与每个办事处的平均距离 (ST_Within, ST_Distance) 2. 查找每个办事处的客户平均收入和保险费 (ST_Within) 3. 查找未被任何现存的办事处覆盖的客户 (ST_Within) 4. 查找每个办事处与之重叠的危险区域数 (ST_Overlaps) 5. 查找与特定客户位置最近的办事处，假设办事处位于办事处区域的质心 (ST_Distance, ST_Centriod) 6. 查找其位置在一个特定的危险区域边界附近的客户 (ST_Buffer, ST_Overlaps) 7. 查找那些被一特定的办事处覆盖的高危险客户 （所有步骤都利用 gseRunSpatialQueries）	这些步骤使用 DB2 SQL 语言中的 Spatial 谓词和函数执行 Spatial 分析。可能时，DB2 查询优化器利用 Spatial 列上的 Spatial 索引来提高查询性能。
将 Spatial 层调出到文件	调出 highRiskCustomers 层 (gseExportShape)	该步骤显示将查询结果调出到 SHAPE 文件的示例。将查询结果调出到另一个文件格式允许第三方工具（例如 SRI ArcInfo）使用该信息。

第2部分 参考资料

第9章 存储过程

本章说明了一些存储过程，这些存储过程使您能够使用 DB2 Spatial Extender 构建地理信息系统。当从 “控制中心” 启用并使用 DB2 Spatial Extender 时，隐式地调用这些存储过程。例如，当您在 DB2 Spatial Extender 窗口单击**确认**时， DB2 为您调用与该窗口相关的存储过程。或者，您可以在应用程序中显式地调用存储过程。建议在这样的程序中包括头文件 db2gse.h。此文件包含您为存储过程参数指定的常量的宏定义。在 AIX 上，该文件存储在 $DB2INSTANCE/sqllib/include/ 目录中。在 Windows NT上，该文件存储在 %DB2PATH%\include\ 目录中。

注意:

存储过程输入参数的所有字符串常量都是区分大小写的。要查明哪些参数需要这些常量，参见本章中的表。

在可调用存储过程之前，无论是隐式还是显式调用，都必须首先连接至安装有 DB2 Spatial Extender 的数据库。使用的第一个存储过程是 db2gse.gse_enable_db。它对 Spatial 操作启用数据库。只有在启用数据库之后，才可以使用其他存储过程。

存储过程的实现归档在 DB2 Spatial Extender 服务器上的 db2gse 库中。

您可以使用下列列表按存储过程的名称或按存储过程实现的任务来查看存储过程。第一个列表列示以下名称:

- 第79页的『db2gse.gse_register_layer』
- 第84页的『db2gse.gse_run_gc』
- 第86页的『db2gse.gse_unregist_gc』
- 第87页的『db2gse.gse_unregist_layer』

第二个列表显示存储过程实现的任务。

- 创建 Spatial 列的索引（参见第67页的『db2gse.gse_enable_idx』）。
- 创建 Spatial 参考系（参见第69页的『db2gse.gse_enable_sref』）。
- 禁用地理编码器以便它不能自动使 Spatial 列与对应的属性列保持同步（参见第59页的『db2gse.gse_disable_autogc』）。
- 禁用数据库中 Spatial 操作的支持（参见第61页的『db2gse.gse_disable_db』）。
- 卸下 Spatial 参考系（参见第62页的『db2gse.gse_disable_sref』）。
- 启用数据库以支持 Spatial 操作（参见第66页的『db2gse.gse_enable_db』）。
- 启用地理编码器以自动使 Spatial 列与对应的属性列保持同步（参见第63页的『db2gse.gse_enable_autogc』）。
- 将层及其相关的表调出至形状文件（参见第71页的『db2gse.gse_export_shape』）。
- 从 ESRI_SDE 传送文件调入层及其相关的表（参见第73页的『db2gse.gse_import_sde』）。
- 从形状文件调入层及其相关的表（参见第75页的『db2gse.gse_import_shape』）。
- 注册不同于缺省地理编码器的地理编码器（参见第77页的『db2gse.gse_register_gc』）。
- 将 Spatial 列注册为层（参见第79页的『db2gse.gse_register_layer』）。
- 以批处理方式运行地理编码器（参见第86页的『db2gse.gse_unregist_gc』）。
- 注销不同于缺省地理编码器的地理编码器（参见第87页的『db2gse.gse_unregist_layer』）。
- 注销层（参见第87页的『db2gse.gse_unregist_layer』）。

有关执行这些任务的顺序的更多信息，参见第3页的『第1章 关于 DB2 Spatial Extender』和第49页的『第8章 编写 DB2 Spatial Extender 的应用程序』。

db2gse.gse_disable_autogc

使用此存储过程卸下或临时禁用使 Spatial 列与其相关属性列保持同步的触发器。例如，在以批处理方式对属性列中的值进行地理编码时，建议禁用这些触发器。有关更多信息，参见第35页的『关于地理编码』。

有关用于调用此存储过程的代码示例，参见样本程序中的 C 函数 gseDisableAutoGc。有关此程序的信息，参见第49页的『第8章 编写 DB2 Spatial Extender 的应用程序』。

授权

调用此存储过程所使用的用户 ID 必须具有形式为权限、特权或一组特权的授权；具体为:

- 对数据库的 SYSADM 或 DBADM 权限，该数据库包含一张表，其中定义了正要卸下或临时禁用的触发器。
- 对此表的 CONTROL 特权。
- 对此表的 ALTER、SELECT 和 UPDATE 特权。

输入参数

表 6. db2gse.gse_disable_autogc 存储过程的输入参数。

名称	数据类型	说明
operMode	SMALLINT	指示是卸下还是临时禁用触发器。 此参数不能为空。 **注解**: 要卸下触发器，使用 GSE_AUTOGC_DROP 宏。要临时禁用这些触发器，使用 GSE_AUTOGC_INVALIDATE 宏。要查明什么值与这些宏相关，查阅 db2gse.h 文件。 在 AIX 上，此文件存储在 $DB2INSTANCE/sqllib/include/ 目录中。在 Windows NT 上，该文件存储在 %DB2PATH%\include\ 目录中。
layerSchema	VARCHAR(30)	layerTable 参数中指定的表或视图所属的模式名。 此参数可为空。 **注解**: 若不为 layerSchema 参数提供值，该参数将缺省为调用 db2gse.gse_disable_autogc 存储过程所使用的用户 ID。

表 6. db2gse.gse_disable_autogc 存储过程的输入参数。 (续)

名称	数据类型	说明
layerTable	VARCHAR(128)	表名，该表中定义了想要卸下或临时禁用的触发器。 此参数不能为空。
layerColumn	VARCHAR(128)	已对 Spatial 操作启用的列的名称，该列由想要卸下或临时禁用的触发器维护。 此参数不能为空。

输出参数

表 7. db2gse.gse_disable_autogc 存储过程的输出参数。

名称	数据类型	说明
msgCode	INTEGER	与此存储过程的调用程序可返回的信息相关的代码。
Reserved	VARCHAR(1024)	DB2 Spatial Extender 服务器上构造的完整错误信息。

db2gse.gse_disable_db

使用此存储过程除去一些资源: 这些资源允许 DB2 Spatial Extender 存储 Spatial 数据并支持对此数据执行的操作。

此存储过程的用途是帮助您解决如下问题: 即在对 Spatial 操作启用数据库之后，但在向该数据库添加任何 Spatial 表列或数据*之前*出现的问题。例如，在对 Spatial 操作启用某个数据库之后，决定对另一个数据库使用 DB2 Spatial Extender。只要未定义任何 Spatial 列或调入任何 Spatial 数据，就可以调用此存储过程将所有 Spatial 资源从第一个数据库中除去。

有关用于调用此存储过程的代码示例，参见样本程序中的 C 函数 gseDisableDB。有关此程序的信息，参见第49页的『第8章 编写 DB2 Spatial Extender 的应用程序』。

授权

调用此存储过程所使用的用户 ID 必须对要除去 DB2 Spatial Extender 资源的数据库具有 SYSADM 或 DBADM 权限。

输出参数

表 8. db2gse.gse_disable_db 存储过程的输出参数。

名称	数据类型	说明
msgCode	INTEGER	与此存储过程的调用程序可返回的信息相关的代码。
Reserved	VARCHAR(1024)	DB2 Spatial Extender 服务器上构造的完整错误信息。

db2gse.gse_disable_sref

使用此存储过程卸下 Spatial 参考系。在处理此存储过程时，从 DB2GSE.SPATIAL_REF_SYS 目录视图中除去有关 Spatial 参考系的信息。有关此视图的信息，参见第99页的『DB2GSE.SPATIAL_REF_SYS』。

有关用于调用此存储过程的代码示例，参见样本程序中的 C 函数 gseDisableSref。有关此程序的信息，参见第49页的『第8章 编写 DB2 Spatial Extender 的应用程序』。

授权

不需要。

输入参数

表 9. db2gse.gse_disable_sref 存储过程的输入参数。

名称	数据类型	说明
srId	INTEGER	要卸下的 Spatial 参考系的数字标识符。 此参数不能为空。

输出参数

表 10. db2gse.gse_disable_sref 存储过程的输出参数。

名称	数据类型	说明
msgCode	INTEGER	与此存储过程的调用程序可返回的信息相关的代码。
Reserved	VARCHAR(1024)	DB2 Spatial Extender 服务器上构造的完整错误信息。

限制

在可卸下 Spatial 参考系之前，必须注销使用该参考系的任何层。若这样的层保持注册状态，将拒绝卸下 Spatial 参考系的请求。

db2gse.gse_enable_autogc

使用此存储过程执行下列操作:

- 创建将使 Spatial 列与其相关属性列保持同步的触发器。每次将值插入属性列，或在属性列中更新值时，触发器将调用注册的地理编码器，对插入或更新的值进行地理编码，并将结果数据放在 Spatial 列中。
- 在临时禁用触发器之后重新激活触发器。
- 确定哪个函数将用来对插入或更新的值进行地理编码。

有关用于调用此存储过程的代码示例，参见样本程序中的 C 函数 gseEnableAutoGC。有关此程序的信息，参见第49页的『第8章 编写 DB2 Spatial Extender 的应用程序』。

授权

调用此存储过程所使用的用户 ID 必须具有形式为权限、特权或一组特权的授权;具体为:

- 对数据库的 SYSADM 或 DBADM 权限，该数据库包含一张表，其中定义了此存储过程创建的触发器。
- 对此表的 CONTROL 特权。
- 对此表的 ALTER、SELECT 和 UPDATE 特权。

输入参数

表 11. db2gse.gse_enable_autogc 存储过程的输入参数。

名称	数据类型	说明
operMode	SMALLINT	其值指示是要新建启动地理编码的触发器，还是在临时禁用之后重新激活它。 此参数不能为空。 **注解**: 要创建触发器，使用 GSE_AUTOGC_CREATE 宏。要重新激活触发器，使用 GSE_AUTOGC_RECREATE 宏。要查明什么值与这些宏相关，查阅 db2gse.h 文件。在 AIX 上，此文件存储在 $DB2INSTANCE/sqllib/include/ 目录中。在 Windows NT 上，该文件存储在 %DB2PATH%\include\ 目录中。

表 11. db2gse.gse_enable_autogc 存储过程的输入参数。 (续)

名称	数据类型	说明
layerSchema	VARCHAR(30)	layerTable 参数中指定的表所属的模式的名称。 此参数可为空。 **注解**: 若不为 layerSchema 参数提供值，该参数将缺省为调用 db2gse.gse_enable_autogc 存储过程所使用的用户 ID。
layerTable	VARCHAR(128)	此存储过程创建或重新激活的触发器将处理的表的名称。 此参数不能为空。
layerColumn	VARCHAR(128)	将由此存储过程创建或重新激活的触发器维护的 Spatial 列的名称。 此参数不能为空。
gcId	INTEGER	将由此存储过程创建或重新激活的插入和更新触发器调用的地理编码器的标识符。 若 operMode 参数设置为 GSE_AUTOGC_CREATE，则此参数不能为空。若 operMode 设置为 GSE_AUTOGC_RECREATE，则此参数可为空。
precisionLevel	INTEGER	为使地理编码器能够成功处理源数据，源数据与对应的参考数据的匹配必须达到的程度。 若 operMode 参数设置为 GSE_AUTOGC_CREATE，则此参数不能为空。若 operMode 设置为 GSE_AUTOGC_RECREATE，则此参数可为空。 **注解**: 精度级别的范围是 1% 至 100%。
vendorSpecific	VARCHAR(256)	供应商提供的技术信息；例如，供应商用来设置参数的文件的路径和名称。 若 operMode 参数设置为 GSE_AUTOGC_CREATE，则此参数不能为空。若 operMode 设置为 GSE_AUTOGC_RECREATE，则此参数可为空。

输出参数

表 12. db2gse.gse_enable_autogc 存储过程的输出参数。

名称	数据类型	说明
msgCode	INTEGER	与此存储过程的调用程序可返回的信息相关的代码。
Reserved	VARCHAR(1024)	DB2 Spatial Extender 服务器上构造的完整错误信息。

限制

- layerColumn 参数必须引用已注册为表层的列。
- 若 operMode 参数设置为 GSE_AUTOGC_CREATE, 则必须为 gcId 参数指定已注册的地理编码器的标识符。

db2gse.gse_enable_db

使用此存储过程为数据库提供存储 Spatial 数据和支持操作所需要的资源。这些资源包括 Spatial 数据类型、Spatial 索引类型、目录表和视图、提供的函数以及其他存储过程。

有关用于调用此存储过程的代码示例，参见样本程序中的 C 函数 gseEnableDB。有关此程序的信息，参见第49页的『第8章 编写 DB2 Spatial Extender 的应用程序』。

授权

调用存储过程所使用的用户 ID 必须对正要启用的数据库具有 SYSADM 或 DBADM 权限。

输出参数

表 13. db2gse.gse_enable_db 存储过程的输出参数。

名称	数据类型	说明
msgCode	INTEGER	与此存储过程的调用程序可返回的信息相关的代码。
Reserved	VARCHAR(1024)	DB2 Spatial Extender 服务器上构造的完整错误信息。

db2gse.gse_enable_idx

使用此存储过程为 Spatial 列创建索引。

有关用于调用此存储过程的代码示例，参见样本程序中的 C 函数 gseEnableIdx。有关此程序的信息，参见第49页的『第8章 编写 DB2 Spatial Extender 的应用程序』。

授权

调用此存储过程所使用的用户 ID 必须具有下列权限或特权之一:

- 对数据库的 SYSADM 或 DBADM 权限，该数据库包含一张表，系统根据该表使用所启用的索引。
- 对此表的 CONTROL 或 INDEX 特权。

输入参数

表 14. db2gse.gse_enable_idx 存储过程的输入参数。

名称	数据类型	说明
layerSchema	VARCHAR(30)	layerTable 参数中指定的表所属的模式的名称。 此参数可为空。 **注解**: 若不为 layerSchema 参数提供值，该参数将缺省为调用 db2gse.gse_enable_idx 存储过程所使用的用户 ID。
layerTable	VARCHAR(128)	表名，该表中定义了要创建的索引。 此参数不能为空。
layerColumn	VARCHAR(128)	要借助正要创建的索引搜索的已对 Spatial 操作启用的列的名称。 此参数不能为空。
indexName	VARCHAR(128)	要创建的索引的名称。 此参数不能为空。 **注解**: 不要指定模式名。DB2 Spatial Extender 自动为 layerSchema 参数所引用的模式指定索引。
gridSize1	DOUBLE	其数值指示最细的索引网格应该具有的粒度。 此参数不能为空。

表 14. db2gse.gse_enable_idx 存储过程的输入参数。 (续)

名称	数据类型	说明
gridSize2	DOUBLE	其数值指示以下两种情形之一: (1) 此索引没有第二个网格; (2) 第二个网格应该具有的粒度。 此参数可为空。 **注解**: 若将没有第二个网格，则指定 0。若想要第二个网格，则该网格必须比 gridSize1 所指示的网格粒度小。
gridSize3	DOUBLE	其数值指示以下两种情形之一: (1) 此索引没有第三个网格; (2) 第三个网格应该具有的粒度。 此参数可为空。 **注解**: 若将没有第三个网格，则指定 0。若想要第三个网格，则该网格必须比 gridSize2 所指示的网格粒度小。

输出参数

表 15. db2gse.gse_enable_idx 存储过程的输出参数。

名称	数据类型	说明
msgCode	INTEGER	与此存储过程的调用程序可返回的信息相关的代码。
Reserved	VARCHAR(1024)	DB2 Spatial Extender 服务器上构造的完整错误信息。

db2gse.gse_enable_sref

此存储过程用来指定如何将特定坐标系中的负数和小数转换为正整数，以便 DB2 Spatial Extender 能够存储这些数值。指定的内容统称为*Spatial 参考系*。在处理此存储过程时，有关 Spatial 参考系的信息被添加至 DB2GSE.SPATIAL_REF_SYS 目录视图.有关此视图的信息，参见第99页的『DB2GSE.SPATIAL_REF_SYS』。

有关用于调用此存储过程的代码示例，参见样本程序中的 C 函数 gseEnableSref。有关此程序的信息，参见第49页的『第8章 编写 DB2 Spatial Extender 的应用程序』。

授权

不需要。

输入参数

表 16. db2gse.gse_enable_sref 存储过程的输入参数。

名称	数据类型	说明
srId	INTEGER	Spatial 参考系的数字标识符。 此参数不能为空。 **注解**: 此标识符在已对 Spatial 操作启用的数据库中必须是唯一的。
srName	VARCHAR(64)	Spatial 参考系的简短说明。 此参数不能为空。 **注解**: 此说明在已对 Spatial 操作启用的数据库中必须是唯一的。
falsex	DOUBLE	一个数，当从负的 X 坐标值减去该数时，余下非负数（即正数或零）。 此参数不能为空。
falsey	DOUBLE	一个数，当从负的 Y 坐标值减去该数时，余下非负数（即正数或零）。 此参数不能为空。
xyunits	DOUBLE	一个数值，当它与小数 X 坐标或小数 Y 坐标相乘时，得到可存储为 32 位数据项的整数。 此参数不能为空。

表 16. db2gse.gse_enable_sref 存储过程的输入参数。 (续)

名称	数据类型	说明
falsez	DOUBLE	一个数，当从负的 Z 坐标值减去该数时，余下非负数（即正数或零）。 此参数不能为空。
zunits	DOUBLE	一个数，当它与小数 Z 坐标相乘时，得到可存储为 32 位数据项的整数。 此参数不能为空。
falsem	DOUBLE	一个数，当从负的度量单位减去该数时，余下非负数（即正数或零）。 此参数不能为空。
munits	DOUBLE	一个数，当它与小数度量单位相乘时，得到可存储为 32 位数据项的整数。 此参数不能为空。
scId	INTEGER	从其中导出 Spatial 参考系的坐标系的数字标识符。要查明坐标系的数字标识符是什么，查阅 DB2GSE.COORD_REF_SYS 目录视图 第97页的『DB2GSE.COORD_REF_SYS』。 此参数不能为空。

输出参数

表 17. db2gse.gse_enable_sref 存储过程的输出参数。

名称	数据类型	说明
msgCode	INTEGER	与此存储过程的调用程序可返回的信息相关的代码。
Reserved	VARCHAR(1024)	DB2 Spatial Extender 服务器上构造的完整错误信息。

db2gse.gse_export_shape

此存储过程用来将层及其相关表调出至形状文件，或创建新的形状文件并将层及其相关表调出至此新文件。

有关用于调用此存储过程的代码示例，参见样本程序中的 C 函数 gseExportShape。有关此程序的信息，参见第49页的『第8章 编写 DB2 Spatial Extender 的应用程序』。

授权

调用此存储过程所使用的用户 ID 必须对要调出的表具有 SELECT 特权。

输入参数

表 18. db2gse.gse_export_shape 存储过程的输入参数。

名称	数据类型	说明
layerSchema	VARCHAR(30)	layerTable 参数中指定的表所属的模式的名称。 此参数可为空。 **注解**: 若不为 layerSchema 参数提供值，该参数将缺省为调用 db2gse.gse_export_shape 存储过程所使用的用户 ID。
layerTable	VARCHAR(128)	要调出的表的名称。 此参数不能为空。
layerColumn	VARCHAR(30)	已注册为要调出的层的列名。 此参数不能为空。
fileName	VARCHAR(128)	要将指定层调出至其中的形状文件的名称。 此参数不能为空。
whereClause	VARCHAR(1024)	SQL WHERE 子句的主体。它对要进行地理编码的记录集定义限制。该子句可引用您要调出的表中的任何属性列。 此参数可为空。

输出参数

表 19. db2gse.gse_export_shape 存储过程的输出参数。

名称	数据类型	说明
msgCode	INTEGER	与此存储过程的调用程序可返回的信息相关的代码。
Reserved	VARCHAR(1024)	DB2 Spatial Extender 服务器上构造的完整错误信息。

限制

每次只能调出一层。

db2gse.gse_import_sde

此存储过程用来将 SDE 传送文件调入至已允许进行 Spatial 操作的数据库。该存储过程可以两种方法操作:

- 若 SDE 传送文件的目标是具有已注册的层列的现存表，则 DB2 Spatial Extender 将该文件的数据装入该表。
- 否则，DB2 Spatial Extender 将创建一个具有 Spatial 列的表，将此列注册为层，然后将该文件的数据装入该层和该表的其他列。

SDE 传送文件中指定的 Spatial 参考系将与注册至 DB2 Spatial Extender 的 Spatial 参考系进行比较。若指定的系统与注册的系统匹配，则传送数据中的本机和十进制值在装入之后将按注册系统规定的方式进行修改。若指定的系统与注册系统中的任何一个都不匹配，则 DB2 Spatial Extender 将创建新的 Spatial 参考系以规定如何修改。

授权

当您将数据调入至现存表时，调用此存储过程所使用的用户 ID 必须具有下列权限或特权之一:

- 对数据库的 SYSADM 或 DBADM 权限，该数据库包含要将数据调入至其中的表。
- 对此表的 CONTROL 特权。

在必须创建要将数据调入至其中的表时，调用此存储过程所使用的用户 ID 必须具有下列权限或特权之一:

- 对数据库的 SYSADM 或 DBADM 权限，该数据库包含要创建的表。

输入参数

表 20. db2gse.gse_import_sde 存储过程的输入参数。

名称	数据类型	说明
layerSchema	VARCHAR(30)	layerTable 参数中指定的表或视图所属的模式名。 此参数可为空。 **注解**: 若不为 layerSchema 参数提供值，该参数将缺省为调用 db2gse.gse_import_sde 存储过程所使用的用户 ID。
layerTable	VARCHAR(128)	要装入 SDE 传送数据的表的名称。 此参数不能为空。

表 20. db2gse.gse_import_sde 存储过程的输入参数。 (续)

名称	数据类型	说明
layerColumn	VARCHAR(30)	已注册为要装入 SDE 传送文件 Spatial 数据的层的列名。 此参数不能为空。
fileName	VARCHAR(128)	要调入的 SDE 传送文件的名称。 此参数不能为空。
commitScope	INTEGER	每个检查点的记录数。 此参数可为空。

输出参数

表 21. db2gse.gse_import_sde 存储过程的输出参数。

名称	数据类型	说明
msgCode	INTEGER	与此存储过程的调用程序可返回的信息相关的代码。
Reserved	VARCHAR(1024)	DB2 Spatial Extender 服务器上构造的完整错误信息。

db2gse.gse_import_shape

此存储过程用来将形状文件调入已对 Spatial 操作启用的数据库。该存储过程可以两种方法操作:

- 若形状文件的目标是具有已注册的层列的现存表，则 DB2 Spatial Extender 将该文件的数据装入该表。
- 否则，DB2 Spatial Extender 将创建一个具有 Spatial 列的表，将此列注册为层，然后将该文件的数据装入该层和该表的其他列。

有关用于调用此存储过程的代码示例，参见样本程序中的 C 函数 gseImportShape。有关此程序的信息，参见第49页的『第8章 编写 DB2 Spatial Extender 的应用程序』。

授权

调用此存储过程所使用的用户 ID 必须具有下列权限或特权之一:

- 对数据库的 SYSADM 或 DBADM 权限，该数据库包含要装入调入的形状数据的表。
- 对此表的 CONTROL 特权。

输入参数

表 22. db2gse.gse_import_shape 存储过程的输入参数。

名称	数据类型	说明
layerSchema	VARCHAR(30)	layerTable 参数中指定的表或视图所属的模式名。 此参数可为空。 **注解**: 若不为 layerSchema 参数提供值，该参数将缺省为调用 db2gse.gse_import_shape 存储过程所使用的用户 ID。
layerTable	VARCHAR(128)	要装入调入的形状文件的表的名称。 此参数不能为空。
layerColumn	VARCHAR(30)	已注册为要装入形状数据的层的列名。 此参数不能为空。
fileName	VARCHAR(128)	要调入的形状文件的名称。 此参数不能为空。

表 22. db2gse.gse_import_shape 存储过程的输入参数。 (续)

名称	数据类型	说明
exceptionFile	VARCHAR(128)	存储不能调入的形状的文件路径和名称。这是新文件，该文件将在 db2gse.gse_import_shape 存储过程运行时创建。 此参数不能为空。
srId	INTEGER	Spatial 参考系的标识符，该 Spatial 参考系用于要装入形状数据的层。 此参数可为空。 **注解**: 若未指定此标识符，则将把内部转换设置为形状文件的最大可能分辨率。
commitScope	INTEGER	每个检查点的记录数。 此参数可为空。

输出参数

表 23. db2gse.gse_import_shape 存储过程的输出参数。

名称	数据类型	说明
msgCode	INTEGER	与此存储过程的调用程序可返回的信息相关的代码。
Reserved	VARCHAR(1024)	DB2 Spatial Extender 服务器上构造的完整错误信息。

db2gse.gse_register_gc

此存储过程用来注册缺省地理编码器之外的地理编码器。要找出地理编码器是否已注册，查阅 DB2GSE.SPATIAL_GEOCODER 目录视图（在 第98页的『DB2GSE.SPATIAL_GEOCODER』 中有说明）。

授权

调用此存储过程所使用的用户 ID 必须对包含此存储过程所注册的地理编码器的数据库具有 SYSADM 或 DBADM 权限。

输入参数

表 24. db2gse.gse_register_gc 存储过程的输入参数。

名称	数据类型	说明
gcId	INTEGER	要注册的地理编码器的数字标识符。 此参数不能为空。 **注解**: 此标识符在该数据库中必须是唯一的。
gcName	VARCHAR(64)	要注册的地理编码器的简短说明。 此参数不能为空。 **注解**: 此说明必须是该数据库中的唯一性字符串。
vendorName	VARCHAR(64)	提供您要注册的地理编码器的供应商名称。 此参数不能为空。
primaryUDF	VARCHAR(256)	要注册的地理编码器的全限定名。 此参数不能为空。
precisionLevel	INTEGER	为使地理编码器能够成功处理源数据，源数据与对应的参考数据的匹配必须达到的程度。 此参数不能为空。 **注解**: 精度级别的范围是 1% 至 100%。
vendorSpecific	VARCHAR(256)	供应商提供的技术信息；例如，供应商用来设置参数的文件的路径和名称。 此参数可为空。
geoArea	VARCHAR(256)	要进行地理编码的地理区域。 此参数可为空。

表 24. db2gse.gse_register_gc 存储过程的输入参数。 (续)

名称	数据类型	说明
description	VARCHAR(256)	供应商提供的注释 此参数可为空。

输出参数

表 25. db2gse.gse_register_gc 存储过程的输出参数。

名称	数据类型	说明
msgCode	INTEGER	与此存储过程的调用程序可返回的信息相关的代码。
Reserved	VARCHAR(1024)	DB2 Spatial Extender 服务器上构造的完整错误信息。

db2gse.gse_register_layer

此存储过程用来将 Spatial 列注册为层。在处理此存储过程时，有关所注册的层的信息被添加至 DB2GSE.GEOMETRY_COLUMNS 目录视图。有关此视图的信息，参见第97页的『DB2GSE.GEOMETRY_COLUMNS』。

有关用于调用此存储过程的代码示例，参见样本程序中的 C 函数 gseRegisterLayer。有关此程序的信息，参见第49页的『第8章 编写 DB2 Spatial Extender 的应用程序』。

授权

调用此存储过程所使用的用户 ID 必须具有下列权限或特权之一:

- 对于表层:
  - 对包含此层所属表的数据库的 SYSADM 或 DBADM 权限。
  - 对此表的 CONTROL 或 ALTER 特权。
- 对于视图层:
  - 对包含以下两项的基表的 SELECT 特权: (1) 此层要进行地理编码的地址数据; (2) 从地理编码产生的 Spatial 数据。

输入参数

表 26. db2gse.gse_register_layer 存储过程的输入参数。

名称	数据类型	说明
layerSchema	INTEGER(30)	layerTable 参数中指定的表或视图所属的模式名。 此参数可为空。 **注解**: 若不为 layerSchema 参数提供值，该参数将缺省为调用 db2gse.gse_register_layer 存储过程所使用的用户 ID。
layerTable	VARCHAR(128)	表或视图的名称，该表或视图包含正要注册为层的列。 此参数不能为空。
layerColumn	VARCHAR(128)	正要注册为层的列的名称。若该列不存在，则 DB2 Spatial Extender 将创建它。 此参数不能为空。

表 26. db2gse.gse_register_layer 存储过程的输入参数。 (续)

名称	数据类型	说明
layerTypeName	VARCHAR(64)	正要注册为层的列的数据类型。必须以大写字母指定数据类型；例如: `ST_POINT` 若该列是在处理此存储过程时将创建的表列，则此参数不能为空。否则，若该列是表或视图中的现存列，则此参数可为空。
srId	INTEGER	用于此层的 Spatial 参考系的标识符。 对于表层，此参数不能为空。当您注册视图层时，DB2 Spatial Extender 忽略此参数。
geoSchema	VARCHAR(30)	在将视图列注册为层时应用。geoSchema 参数是作为该列所属视图基础的表的模式。 当您将视图列注册为层时，此参数可为空。当您将表列注册为层时，DB2 Spatial Extender 忽略此参数。 **注解**: 若不为 geoSchema 参数提供值，则该参数将缺省为 layerSchema 参数的值。
geoTable	VARCHAR(128)	在将视图列注册为层时应用。geoTable 参数是作为列所属视图基础的表的名称。 当您将视图列注册为层时，此参数不能为空。当您将表列注册为层时，DB2 Spatial Extender 忽略此参数。
geoColumn	VARCHAR(128)	在将视图列注册为层时应用。geoColumn 参数是作为此视图列基础的表列的名称。 当您将视图列注册为层时，此参数不能为空。当您将表列注册为层时，DB2 Spatial Extender 忽略此参数。
nAttributes	SMALLINT	包含此层将进行地理编码的源数据的列数。 当您将表列注册为层时，此参数可为空。当您将视图列注册为层时，DB2 Spatial Extender 忽略此参数。

表 26. db2gse.gse_register_layer 存储过程的输入参数。 (续)

名称	数据类型	说明
attr1Name	VARCHAR(128)	包含此层将进行地理编码的源数据的第一列名称。 当您将表列注册为层时，此参数可为空。当您将视图列注册为层时，DB2 Spatial Extender 忽略此参数。 若打算使用缺省地理编码器，则需要在 attr1Name 列中存储街道地址。
attr2Name	VARCHAR(128)	包含此层将进行地理编码的源数据的第二列名称。 当您将表列注册为层时，此参数可为空。当您将视图列注册为层时，DB2 Spatial Extender 忽略此参数。 若打算使用缺省地理编码器，则需要在 attr2Name 列中存储城市名。
attr3Name	VARCHAR(128)	包含此层将进行地理编码的源数据的第三列名称。 当您将表列注册为层时，此参数可为空。当您将视图列注册为层时，DB2 Spatial Extender 忽略此参数。 若打算使用缺省地理编码器，则需要在 attr3Name 列中存储州的名称或简称。
attr4Name	VARCHAR(128)	包含此层将进行地理编码的源数据的第四列名称。 当您将表列注册为层时，此参数可为空。当您将视图列注册为层时，DB2 Spatial Extender 忽略此参数。 若打算使用缺省地理编码器，则需要在 attr4Name 列中存储邮政编码。
attr5Name	VARCHAR(128)	包含此层将进行地理编码的源数据的第五列名称。 当您将表列注册为层时，此参数可为空。当您将视图列注册为层时，DB2 Spatial Extender 忽略此参数。 缺省地理编码器忽略 Attr5Name 列。

表 26. db2gse.gse_register_layer 存储过程的输入参数。 (续)

名称	数据类型	说明
attr6Name	VARCHAR(128)	包含此层将进行地理编码的源数据的第六列名称。 当您将表列注册为层时，此参数可为空。当您将视图列注册为层时，DB2 Spatial Extender 忽略此参数。 缺省地理编码器忽略 Attr6Name 列。
attr7Name	VARCHAR(128)	包含此层将进行地理编码的源数据的第柒列名称。 当您将表列注册为层时，此参数可为空。当您将视图列注册为层时，DB2 Spatial Extender 忽略此参数。 缺省地理编码器忽略 Attr7Name 列。
attr8Name	VARCHAR(128)	包含此层将进行地理编码的源数据的第捌列名称。 当您将表列注册为层时，此参数可为空。当您将视图列注册为层时，DB2 Spatial Extender 忽略此参数。 缺省地理编码器忽略 Attr8Name 列。
attr9Name	VARCHAR(128)	包含此层将进行地理编码的源数据的第玖列名称。 当您将表列注册为层时，此参数可为空。当您将视图列注册为层时，DB2 Spatial Extender 忽略此参数。 缺省地理编码器忽略 Attr9Name 列。
attr10Name	VARCHAR(128)	包含此层将进行地理编码的源数据的第拾列名称。 当您将表列注册为层时，此参数可为空。当您将视图列注册为层时，DB2 Spatial Extender 忽略此参数。 缺省地理编码器忽略 Attr10Name 列。

输出参数

表 27. db2gse.gse_register_layer 存储过程的输出参数。

名称	数据类型	说明
msgCode	INTEGER	与此存储过程的调用程序可返回的信息相关的代码。

表 27. db2gse.gse_register_layer 存储过程的输出参数。 (续)

名称	数据类型	说明
Reserved	VARCHAR(1024)	DB2 Spatial Extender 服务器上构造的完整错误信息。

限制

- 若正要将视图列注册为层，它必须基于已注册为层的表列。
- 包含将为正要注册的层进行地理编码的数据的属性列不能超过十个。

db2gse.gse_run_gc

此存储过程用来以批处理方式运行地理编码器。有关此任务的信息，参见第37页的『以批处理方式运行地理编码器』。

有关用于调用此存储过程的代码示例，参见样本程序中的 C 函数 gseRunGC。有关此程序的信息，参见第49页的『第8章 编写 DB2 Spatial Extender 的应用程序』。

授权

调用此存储过程所使用的用户 ID 必须具有下列权限或特权之一:

- 对数据库的 SYSADM 或 DBADM 权限，该数据库包含指定的地理编码器要进行操作的表。
- 对此表的 CONTROL 或 UPDATE 特权。

输入参数

表 28. db2gse.gse_run_gc 存储过程的输入参数。

名称	数据类型	说明
layerSchema	VARCHAR(30)	layerTable 参数中指定的表或视图所属的模式名。 此参数可为空。 **注解**: 若不为 layerSchema 参数提供值，该参数将缺省为调用 db2gse.gse_run_gc 所使用的用户 ID。
layerTable	VARCHAR(128)	表名，该表包含要在其中插入经过地理编码的数据的列。 此参数不能为空。
layerColumn	VARCHAR(128)	要在其中插入经过地理编码的数据的列的名称。 此参数不能为空。
gcId	INTEGER	想要运行的地理编码器的标识符。 此参数可为空。 要查明已注册的地理编码的标识符，查阅 DB2GSE.SPATIAL_GEOCODER 目录视图。

表 28. db2gse.gse_run_gc 存储过程的输入参数。 (续)

名称	数据类型	说明
precisionLevel	INTEGER	为使地理编码器能够成功处理源数据，源数据与对应的参考数据的匹配必须达到的程度。 此参数可为空。 **注解**: 精度级别的范围是 1% 至 100%。
vendorSpecific	VARCHAR(256)	供应商提供的技术信息；例如，供应商用来设置参数的文件的路径和名称。 此参数可为空。
whereClause	VARCHAR(256)	WHERE 子句的主体。它对要进行地理编码的记录集定义限制。该子句可引用地理编码器要处理的表中的任何属性列。 此参数可为空。
commitScope	INTEGER	每个检查点的记录数。 此参数可为空。

输出参数

表 29. db2gse.gse_run_gc 存储过程的输出参数。

名称	数据类型	说明
msgCode	INTEGER	与此存储过程的调用程序可返回的信息相关的代码。
Reserved	VARCHAR(1024)	DB2 Spatial Extender 服务器上构造的完整错误信息。

db2gse.gse_unregist_gc

此存储过程用来注销不同于缺省地理编码器的地理编码器。

有关想要注销的地理编码器的信息，查阅 DB2GSE.SPATIAL_GEOCODER 目录视图；参见第98页的『DB2GSE.SPATIAL_GEOCODER』。

授权

调用此存储过程所使用的用户 ID 必须对包含要注销的地理编码器的数据库具有 SYSADM 或 DBADM 权限。

输入参数

表 30. db2gse.gse_unregist_gc 存储过程的输入参数。

名称	数据类型	说明
gcId	INTEGER	要注销的地理编码器的标识符。 此参数不能为空。

输出参数

表 31. db2gse.gse_unregist_gc 存储过程的输出参数。

名称	数据类型	说明
msgCode	INTEGER	与此存储过程的调用程序可返回的信息相关的代码。
Reserved	VARCHAR(1024)	DB2 Spatial Extender 服务器上构造的完整错误信息。

db2gse.gse_unregist_layer

此存储过程用来注销层。该存储过程执行此任务的方式如下:

- 将层的定义从 DB2 Spatial Extender 目录表中除去。
- 删除 DB2 Spatial Extender 对此层的基表设置的检查约束，以确保该层的 Spatial 数据符合该层的 Spatial 参考系的需求。
- 卸下用来在添加、更改或除去地址数据时更新 Spatial 列的触发器。

在处理该存储过程时，从 DB2GSE.GEOMETRY_COLUMNS 元视图中除去有关该层的信息。有关此视图的信息，参见第97页的『DB2GSE.GEOMETRY_COLUMNS』。

授权

调用此存储过程所使用的用户 ID 必须具有下列权限或特权之一:

- 对于表层:
  - 对包含此层的基表的数据库的 SYSADM 或 DBADM 权限。
  - 对此表的 CONTROL 或 ALTER 特权。
- 对于视图层:
  - 对包含以下两项的基表的 SELECT 特权: (1) 此层将进行地理编码的地址数据; (2) 从地理编码产生的 Spatial 数据。

输入参数

表 32. db2gse.gse_unregist_layer 存储过程的输入参数。

名称	数据类型	说明
layerSchema	VARCHAR(30)	layerTable 参数中指定的表所属的模式的名称。 此参数可为空。 **注解**: 若不为 layerSchema 参数提供值，该参数将缺省为调用 db2gse.gse_unregister_layer 存储过程所使用的用户 ID。
layerTable	VARCHAR(128)	表名，该表包含 layerColumn 参数中指定的列。 此参数不能为空。
layerColumn	VARCHAR(128)	Spatial 列的名称，该 Spatial 列已定义为您想要注销的层。 此参数不能为空。

输出参数

表 33. db2gse.gse_unregist_layer 存储过程的输出参数。

名称	数据类型	说明
msgCode	INTEGER	与此存储过程的调用程序可返回的信息相关的代码。
Reserved	VARCHAR(1024)	DB2 Spatial Extender 服务器上构造的完整错误信息。

限制

若定义为视图层的视图列基于定义为表层的表列，则直到注销该视图层之后，才能注销此表层。

第10章 信息

本章汇集了 DB2 Spatial Extender 返回给用户的信息。每个信息都有一个标识符。以字母 E 结束的标识符表示错误信息；以 W 结束的标识符表示警告信息；以 I 结束的标识符表示一般信息。

DBA7200E 选择了 10 列以上作为地理编码器的输入

解释: 最多可选择 10 列作为地理编码器的输入。

用户响应: 将列名从“选择的列”框移动到“可用列”框，直到“选择的列”框只列示十个名称或更少为止。

DBA7201E 未对 Spatial Extender 操作启用数据库。

解释: 在可使用 Spatial Extender 之前，必须对 Spatial Extender 启用数据库。

用户响应: 用鼠标右键单击数据库并从菜单中选择 Spatial Extender —> 启用。

GSE0000I 操作成功完成。

GSE0001E Spatial Extender 不能在用户 ID “<user-id>” 下执行请求的操作 (“<operation-name>”)。

解释: 在不具有执行此操作的特权或权限的用户 ID 下请求了此操作。

用户响应: 查阅文档以查明正确的授权是什么或从 Spatial Extender 管理员那里获取正确的授权。

GSE0002E “<value>” 不是 “<argument-name>” 自变量的有效值。

解释: 输入的值不正确或有拼写错误。

用户响应: 查阅文档或咨询 Spatial Extender 管理员以查明需要指定什么值或值的范围。

GSE0003E 因为未指定自变量 “<argument-name>”，所以 Spatial Extender 不能执行请求的操作。

解释: 未指定此操作所需的自变量。

用户响应: 给自变量 “<argument-name>” 指定想要的值；然后再次请求该操作。

GSE0004W 未对自变量 “<argument-name>” 求值。

解释: 请求的操作不使用自变量 “<argument-name>”。

用户响应: 无需操作。

GSE0005E Spatial Extender 不能处理您创建名为 “<object-name>” 的对象的请求。

解释: 对象 “<object-name>” 已存在，或您没有创建它的适当许可权。它可以是表、列、触发器、索引、文件或其他种类的对象。

用户响应: 若 “<object-name>” 是想要的对象，则无需操作。否则，应正确地指定该名称并验证您具有创建该对象的正确许可权。

GSE0006E Spatial Extender 不能对已启用或注册的对象 “<object-name>” 执行请求的操作。

解释: 已启用或已注册对象 “<object-name>”，或它已存在。它可以是层、索引、Spatial 参考系、坐标系、地理编码器或其他种类的对象。

用户响应: 确保对象 “<object-name>” 存在并再次提交请求。

GSE0007E Spatial Extender 不能对尚未启用或注册的对象 “<object-name>” 执行请求的操作。

解释: 尚未启用或注册对象 “<object-name>”。它可以是层、索引、Spatial 参考系、Spatial 坐标系、地理编码器或其他种类的对象。

用户响应: 启用或注册对象 “<object-name>”。然后再次提交请求。

GSE0008E 发生了意外的 SQL 错误 (“<sql-error-message>”)。

用户响应: 在 SQL 错误信息 “<sql-error-message>” 中查找与 SQLCODE 相关的详细信息。必要时，与 IBM 服务代表联系。

GSE0009E 不能对已存在的对象 “<object-name>” 执行请求的操作。

解释: “<object-name>” 已存在于数据库或操作系统中。它可以是文件、表、视图、列、索引、触发器或其他种类的对象。

用户响应: 确保在尝试存取对象时正确指定它。必要时，可删除该对象。

GSE0010E 不能对可能不存在的对象 “<object-name>” 执行请求的操作。

解释: 数据库或操作系统中不存在 “<object-name>”。它可以是文件、表、视图、列、索引、触发器、文件或其他种类的对象。

用户响应: 确保具有存取该对象的正确许可权。若具有此许可权而该对象不存在，则需要创建它。

GSE0011E Spatial Extender 不能禁用或注销对象 “<object-name>”。

解释: “<object-name>” 依赖于另一对象。“<object-name>” 可以是 Spatial 参考系、层、地理编码器或其他种类的对象。

用户响应: 查阅文档以查找 “<object-name>” 可依赖什么种类的对象。然后除去 “<object-name>” 所依赖的特定对象。

GSE0012E 因为未将全限定 Spatial 列 “<layer-schema.layer-name.layer-column>” 注册为表层，所以 Spatial Extender 无法处理您的请求。

解释: 在可执行与全限定 Spatial 列 “<layer-schema.layer-name.layer-column>” 相关的某些操作之前，必须将该列注册为表层（例如，启用它的索引，启用地理编码器以批处理方式填充它或自动更新它）。

用户响应: 通过检查 Spatial Extender 目录中的 DB2GSE.GEOMETRY_COLUMNS 视图，确保已将全限定 Spatial 列 “<layer-schema.layer-name.layer-column>” 注册为表层。还要确保包含此列的表也包括有效的对应属性列。

GSE0013E 未对 Spatial 操作启用数据库。

解释: 未对 Spatial 操作启用数据库。因此，Spatial Extender 目录不存在。

用户响应: 对 Spatial 操作启用数据库。

GSE0014E 已对 Spatial 操作启用数据库。

解释: 已对 Spatial 操作启用数据库。

用户响应: 验证是否已按照您的要求启用了数据库。必要时，禁用数据库。

GSE0498E 发生了如下错误: **“<error-message>”**。

GSE0499W **Spatial Extender** 发出了如下警告: **“<warning-message>”**。

GSE0500E 指定的操作方式 **(“<operation-mode>”)** 无效。

解释: 请求的操作不支持指定的方式。

用户响应: 查阅文档以查明该操作支持什么方式。

GSE1001E **Spatial Extender** 无法注册名为 **“<schema-name.view-name.column-name>”** 的视图层和基于 **Spatial** 列 **“<schema-name.table-name.column-name>”** 的视图层。

解释: 尚未将指定的 Spatial 列 (“<schema-name.table-name.column-name>”) 注册为表层。

用户响应: 将 “<schema-name.table-name.column-name>” 列注册为表层。

GSE1002E **Spatial Extender** 无法注册名为 **“<schema-name.view-name.column-name>”** 的视图层和基于表 **“<schema-name.table-name>”** 的视图层。

解释: 指定的表 (“<schema-name.table-name>”) 不会直接或间接作为视图 “<schema-name.view-name.column-name>” 的基础。

用户响应: 查明视图 “<schema-name.view-name.column-name>” 的基表是什么，并指定此表。

GSE1003E **Spatial Extender** 无法存取名为 **“<schema-name.object-name>”** 的表或视图中名为 **“<column-name>”** 的列。

解释: 表或视图 “<schema-name.object-name>” 没有名为 “<column-name>” 的列。

用户响应: 检查表或视图 “<schema-name.object-name>” 的定义以查明想要的列的正确名称。

GSE1004E **Spatial Extender** 无法将全限定 **Spatial** 列 **“<schema-name.table-name.column-name>”** 注册为表层。

解释: 列 “<schema-name.table-name.column-name>” 不具有 Spatial 数据类型，或与基表不相关。

用户响应: 为列 “<schema-name.table-name.column-name>” 定义 Spatial 数据类型，或确保此列是本地基表的一部分。

GSE1005E 为视图层指定的 **Spatial** 参考系 **(“<view-layer-spatial-reference-id>”)** 不同于对此层的基础表层使用的 **Spatial** 参考系 **(“<table-layer-spatial-reference-id>”)**。

解释: 视图层的 Spatial 参考系必须与基础表层的 Spatial 参考系相同。

用户响应: 为视图层指定基础表层的 Spatial 参考系。

GSE1006E 因为 **“<spatial-reference-id>”** 是无效的 **Spatial** 参考系 **ID**，所以 **Spatial Extender** 无法注册您请求的层。

解释: 尚未启用或注册指定的参考系 (“<spatial-reference-id>”)。

用户响应: 启用或注册该 Spatial 参考系。然后再次提交请求以注册该层。

GSE1007E **当 Spatial Extender 尝试将 Spatial 列 (“<column-name>”) 添加至表 “<schema-name.table-name>” 不成功时，可能发生了 SQL 错误 (SQLSTATE “<sqlstate>”)。**

用户响应: 查看与 SQLSTATE “<sqlstate>” 相关的信息。

GSE1008E **因为视图层的 Spatial 数据类型 “<layer-column-type>” 与基础表层 “<geo-schema.geo-name.geo-column>” 的 Spatial 数据类型 “<geo-column-type>” 不匹配，所以 DB2 Spatial Extender 无法注册视图层 “<layer-schema.layer-name.layer-column>”。**

解释: 视图层 “<layer-schema.layer-name.layer-column>” 的 Spatial 数据类型必须与该层的基础表层 “<geo-schema.geo-name.geo-column>” 的 Spatial 数据类型匹配。这两种数据类型之间的不一致性会在处理 Spatial 数据时导致二义性。

用户响应: 确保视图层和它的基础表层具有相同的 Spatial 数据类型。

GSE1020E **“<’spatial-reference-id>” 是无效的 Spatial 参考系 ID。**

解释: 尚未启用标识符为 “<’spatial-reference-id>” 的 Spatial 参考系。

用户响应: 确保已启用指定的 Spatial 参考系。

GSE1021E **Spatial Extender 无法启用 Spatial 参考系 “<spatial-reference-id>”，因为对应的 Spatial 坐标系 ID “<spatial-coordinate-id>” 无效。**

解释: 未在 Spatial Extender 目录中定义标识符为 “<spatial-coordinate-id>” 的坐标系。

用户响应: 通过检查 Spatial Extender 目录中的 DB2GSE.COORD_REF_SYS 视图，验证坐标系标识符 “<spatial-coordinate-id>”。

GSE1030E **因为 “<schema-name.table-name>” 不是基表，所以 Spatial Extender 不能对它启用地理编码器。**

解释: 包含想要进行地理编码的源数据的对象必须为基表。

用户响应: 确保包含想要进行地理编码的源数据的列是基表的一部分。

GSE1031E **Spatial Extender 无法启用地理编码器 “<geocoder-id>” 自动以创建方式对层 “<layer-schema.layer-name.layer-column>” 进行操作。**

解释: 可能的解释如下:

- 已启用该地理编码器自动更新层 “<layer-schema.layer-name.layer-column>”。
- 已临时使该地理编码器对此层无效。
- 未对此层定义任何源数据列。

用户响应: 若已临时使该地理编码器无效，则启用它以 “重新创建” 方式自动操作。

GSE1032E Spatial Extender 无法启用地理编码器 “<geocoder-id>” 自动以重新创建方式对层 “<layer-schema.layer-name.layer-column>” 进行操作。

解释: 可能的解释如下:

- 已启用该地理编码器自动更新层 “<layer-schema.layer-name.layer-column>”。
- 先前未使该地理编码器对此层无效。
- 未对此层定义任何源数据列。

用户响应: 若先前未以卸下方式禁用该地理编码器，或从未对此层定义它，则启用它以便以 “创建” 方式自动操作。

GSE1033E 当 Spatial Extender 尝试给包含层 “<layer-schema.layer-name.layer-column>” (SQLSTATE “<sqlstate>”) 的列的表添加触发器时，发生 SQL 错误。

解释: 触发器的用途是维护属性列（地理编码器的输入来自这些列）和 Spatial 列（地理编码器的输出进入该列）之间的数据完整性。当 DB2 尝试创建这些触发器时，发生 SQL 错误。

用户响应: 查看与 SQLSTATE “<sqlstate>” 相关的信息。

GSE1034E Spatial Extender 无法以卸下方式对层 “<layer-schema.layer-name.layer-column>” 禁用地理编码器 “<geocoder-id>”。

解释: 可能的解释如下:

- 从未启用该地理编码器自动更新层 “<layer-schema.layer-name.layer-column>”。
- 已以卸下方式禁用该地理编码器。

用户响应: 在尝试禁用该地理编码器之前，确定它的状态。例如，注册它了吗？启用它了吗？然后决定是否需要以卸下方式禁用它。例如，若从未启用它，则根本不必禁用它。

GSE1035E Spatial Extender 无法以 “使其无效” 方式对层 “<layer-schema.layer-name.layer-column>” 禁用地理编码器 “<geocoder-id>”。

解释: 可能的解释如下:

- 从未启用该地理编码器自动更新层 “<layer-schema.layer-name.layer-column>”。
- 已以 “使其无效” 方式或卸下方式禁用该地理编码器。

用户响应: 在尝试禁用该地理编码器之前，确定它的状态。例如，注册它了吗？启用它了吗？然后决定是否需要以 “使其无效” 方式禁用它。例如，若已以 “使其无效” 方式禁用它，则不必以这种方式第二次禁用它。

GSE1036E 当 Spatial Extender 尝试从包含层 “<layer-schema.layer-name.layer-column>” 的列的表卸下触发器时，发生 SQL 错误 (SQLSTATE “<sqlstate>”)。

解释: 创建了触发器来维护属性列（地理编码器的输入来自这些列）和 Spatial 列（地理编码器的输出进入该列）之间的数据完整数。当 DB2 尝试卸下这些触发器时，发生 SQL 错误。

用户响应: 查看与 SQLSTATE “<sqlstate>” 相关的信息。

GSE1037E Spatial Extender 无法对表层 “<layer-schema.layer-name.layer-column>” 的源数据进行地理编码，可能是因为给指定多少属性列将为此层提供源数据的自变量指定了不正确的值 “<number-of-attributes>”。

解释: 错误地指定了与此层相关的属性列数，或错误地指定了这些列中的一列或多列的名称。

用户响应: 确保给此层注册了正确数目和名称的相关属性列，或验证地理编码器的输入和输出数据的正确性。

GSE1038E 当 Spatial Extender 尝试以批处理方式对表层 “<layer-schema.layer-name.layer-column>” 的源数据进行地理编码时，发生 SQL 错误 (SQLSTATE “<sqlstate>”)。

用户响应:

- 查看与 SQLSTATE “<sqlstate>” 相关的信息。
- 确保正确定义了此层的内容和 primaryUDF 自变量。

GSE1050E 指定的网格大小 (“<grid-size>”) 对第一网格级无效。

解释: 指定了零或负数作为第一网格级的网格大小。

用户响应: 指定正数作为网格大小。

GSE1051E 指定的网格大小 (“<grid-size>”) 对第二或第三网格级无效。

解释: 指定了负数作为第二或第三网格级的网格大小。

用户响应: 指定零或正数作为网格大小。

GSE1052E 当 Spatial Extender 尝试为表层 “<layer-schema.layer-name.layer-column>” 创建 Spatial 索引 “<index-schema.index-column>” 时，发生 SQL 错误 (SQLSTATE “<sqlstate>”)。

用户响应:

- 确保正确指定了 Spatial 索引且 Spatial 列没有相关的索引。
- 查看与 SQLSTATE “<sqlstate>” 相关的信息。

GSE1500I 成功地对源记录 “<record-number>” 进行了地理编码。

解释: 成功地对包含属性数据的记录进行了地理编码。

GSE1501W 未对源记录 “<record-number>” 进行地理编码。

解释: 精度级别太高。

用户响应: 用较低的精度级别进行地理编码。

GSE1502W 找不到源记录 “<record-number>”。

用户响应: 确定数据库中是否存在该记录。

GSE2001E 指定的传送文件 (“<filename>”) 无效。

用户响应: 验证指定的文件是 SDE 传送文件且正确指定了路径名。

GSE2002E 提供的 SQL WHERE 子句 (“<SQL-where-clause>”) 无效。

用户响应: 检查 WHERE 子句的 SQL 语法是否正确、是否有拼写错误和无效的列名。

GSE2003E 提供的形状值不合法。

用户响应: 进行检查以确保提供的形状与指定的 Spatial 列类型匹配。若类型匹配或兼容，则几何图形的外形不合法。检查重叠的多边形、单个点、弧线等。

GSE2004E 传送文件模式与指定的层的模式不兼容。

用户响应: 进行检查以确保正确指定了模式和层名。若模式不匹配，则装入该数据作为新表并解决模式差异。

GSE2005E 传送文件几何图形类型与指定层的几何图形类型不兼容。

用户响应: 进行检查以确保正确指定了模式和层名。

GSE2006E 名为 “<filename>” 的文件发生了 I/O 错误。

用户响应: 验证该文件存在，您对该文件具有适当的存取权且该文件未被另一用户使用。

GSE2007E 发生了属性转换错误。

用户响应: 进行检查以确保表中的所有属性类型都是受支持的 - 例如，BLOB 数据在形状文件中不受支持。还要检查超出范围的数据值，或非法的数据值（比如错误的日期）。

GSE2008E 调入／调出函数已用完内存。

用户响应: 验证有足够的内存可用。

第11章 目录视图

DB2 Spatial Extender 的目录视图包含下列各项的元数据:

- 可使用的坐标系。有关这些系统的标识符和注释文本这样的信息，参见『DB2GSE.COORD_REF_SYS』。
- 已注册为层的 Spatial 列。有关这些列的名称、数据类型和相关 Spatial 参考系这样的信息，参见『DB2GSE.GEOMETRY_COLUMNS』。
- 可使用的地理编码器。有关这些地理编码器的标识符和包含这些地理编码器处理的位置的区域这样的信息，参见第98页的『DB2GSE.SPATIAL_GEOCODER』。
- 可使用的 Spatial 参考系。有关它们的标识符及其说明这样的信息，参见第99页的『DB2GSE.SPATIAL_REF_SYS』。

DB2GSE.COORD_REF_SYS

当对 Spatial 操作启用数据库时，DB2 Spatial Extender 注册可在目录表中使用的坐标系。从此表选择的列组成 DB2GSE.COORD_REF_SYS 目录视图， 表34中描述了该视图。

表 34. DB2GSE.COORD_REF_SYS 目录视图中的列

名称	数据类型	是否可为空?	内容
SCID	INTEGER	否	此坐标系的唯一数字标识符。
SC_NAME	VARCHAR(64)	否	此坐标系的名称。
AUTH_NAME	VARCHAR(256)	是	编制此坐标系的组织的名称; 例如，European Petroleum Survey Group (EPSG)。
AUTH_SRID	INTEGER	是	由在 AUTH_NAME 列中指定的组织给此坐标系指定的数字标识符。
DESC	VARCHAR(256)	是	此坐标系的说明。
SRTEXT	VARCHAR(2048)	否	此坐标系的注释文本。

DB2GSE.GEOMETRY_COLUMNS

当创建层时，DB2 Spatial Extender 通过将它的标识符和与它相关的信息记录在目录表中来注册它。从此表选择的列组成 DB2GSE.GEOMETRY_COLUMNS 目录视图， 第98页的表35中描述了该视图。

表 35. DB2GSE.GEOMETRY_COLUMNS 目录视图中的列

名称	数据类型	是否可为空?	内容
LAYER_CATALOG	VARCHAR(30)	是	此层的全限定名。
LAYER_SCHEMA	VARCHAR(30)	否	包含注册为此层的列的表或视图的模式。
LAYER_NAME	VARCHAR(128)	否	包含注册为此层的列的表或视图的名称。
LAYER_COLUMN	VARCHAR(30)	否	注册为此层的列的名称。
GEOMETRY_TYPE	INTEGER	否	注册为此层的列的数据类型。
SRID	INTEGER	否	对注册为此层的列中的值使用的 Spatial 参考系的标识符。
STORAGE_TYPE	INTEGER	是	关于 DB2 如何存储注册为此层的列中的值的信息。例如，STORAGE_TYPE 中的数据可能指示将这些值存储为大对象 (LOB)，或存储为抽象数据类型 (ADT) 的实例。

DB2GSE.SPATIAL_GEOCODER

在目录表中注册了可用的地理编码器。从此表选择的列组成 DB2GSE.SPATIAL_GEOCODER 目录视图， 表36中描述了该视图。

表 36. DB2GSE.SPATIAL_GEOCODER 目录视图中的列

名称	数据类型	是否可为空?	内容
GCID	INTEGER	否	此地理编码器的数字标识符。
GC_NAME	VARCHAR(64)	否	此地理编码器的简要说明。
VENDOR_NAME	VARCHAR(128)	否	提供此地理编码器的供应商的名称。
PRIMARY_UDF	VARCHAR(256)	否	此地理编码器的全限定名。
PRECISION_LEVEL	INTEGER	否	为了成功地通过地理编码器进行处理，源数据必须与对应的参考数据匹配的程度。
VENDOR_SPECIFIC	VARCHAR(256)	是	供应商可用来设置此地理编码器支持的任何特殊参数的文件的路径和名称。
GEO_AREA	VARCHAR(256)	是	包含将进行地理编码的位置的地理区域。
DESCRIPTION	VARCHAR(256)	是	供应商提供的注释。

DB2GSE.SPATIAL_REF_SYS

当创建 Spatial 参考系时，DB2 Spatial Extender 通过将它的标识符和与它相关的信息记录在目录表中来注册它。从此表选择的列组成 DB2GSE.SPATIAL_REF_SYS 目录视图， 表37中描述了该视图。

表 37. DB2GSE.SPATIAL_REF_SYS 目录视图中的列

名称	数据类型	是否可为空?	内容
SRID	INTEGER	否	此 Spatial 参考系的用户定义标识符。
SR_NAME	VARCHAR(64)	否	此 Spatial 参考系的名称。
SCID	INTEGER	否	作为此 Spatial 参考系的基础的坐标系的数字标识符。
SC_NAME	VARCHAR(64)	否	作为此 Spatial 参考系的基础的坐标系的名称。
AUTH_NAME	VARCHAR(256)	是	设置此 Spatial 参考系的标准的组织名称。
AUTH_SRID	INTEGER	是	AUTH_NAME 列中指定的组织给此 Spatial 参考系指定的标识符。
SRTEXT	VARCHAR(2048)	否	此 Spatial 参考系的注释文本。

第12章 Spatial 索引

因为 Spatial 列包含二维图形数据，所以查询那些列的应用程序需要一种索引策略来快速标识位于给定范围内的所有几何图形。由于这个原因，DB2 Spatial Extender 提供基于网格的三层 Spatial 索引。

本章描述这种索引并提供使用它的指南。包括如下主题:

样本程序段

考虑如何在 SQL 中创建和使用索引的如下示例。有关 CREATE INDEX 和 CREATE INDEX EXTENSION 命令的更多信息，可参考 *SQL Reference*。注意: 在创建索引之后，就可发出使用 Spatial 函数和谓词的标准 DDL 和 DML 语句。

```
create table customers (cid int, addr varchar(40), ..., loc db2gse.ST_Point)
create table stores (sid int, addr varchar(40), ..., loc db2gse.ST_Point,
 zone db2gse.ST_Polygon)

create index customersx1 on customers(loc) extend using spatial_index(10e0, 100e0,
 1000e0)
create index storesx1 on stores(loc) extend using spatial_index(10e0, 100e0,
 1000e0)
create index storesx2 on stores(zone) extend using spatial_index(10e0, 100e0,
 1000e0)

insert into customers (cid, addr, loc) values (:cid, :addr, sdeFromBinary(:loc))
insert into customers (cid, addr, loc) values (:cid, :addr, geocode(:addr))
insert into stores (sid, addr, loc) values (:sid, :addr, sdeFromBinary(:loc))

update stores set zone = db2gse.ST_Buffer (loc, 2)

select cid, loc from customers
 where db2gse.ST_Within(loc, :polygon) = 1

select cid, loc from customers
 where db2gse.ST_Within(loc, :circle1) = 1 OR
       db2gse.ST_Within(loc, :circle2) = 1

select c.cid, loc from customers c, stores s
```

```
 where db2gse.ST_Contains(s.zone, c.loc) = 1 selectivity 0.01

select avg(c.income) from customers c
 where not exist (select * from stores s
                  where db2gse.ST_Distance(c.loc, s.loc) < 10)
```

B 树索引

Spatial 索引技术基于传统的分层 B 树索引，但有显著的不同。 Spatial 索引利用*网格索引技术*，该技术设计成能为二维 Spatial 列建立索引。 B 树索引仅可处理一维数据且不能用于 GIS 信息。本节描述如何构造和使用 B 树索引。

B 树索引的顶级称作根节点，对于下一级的每个节点它都包含一个关键字。每个关键字的值是下一级的对应节点的最大现存关键字值。根据基表中的值的数目，可能需要几个中间节点。这些中间节点形成根节点和保持实际基表行 ID 的叶节点之间的桥梁。

数据库管理器从根节点开始搜索 B 树索引。然后它继续通过中间节点，直到到达具有基表的行 ID 的叶节点。

因为 Spatial 列的二维特性需要 Spatial 索引的结构，所以不能对 Spatial 列应用 B 树索引。由于相同的原因，不能对非 Spatial 列应用 Spatial 索引。另外，不能对任何种类的组合列应用 Spatial 索引。

创建 Spatial 索引的方法

创建 Spatial 索引有如下几种方法:

- 从 “创建 Spatial 索引” 窗口定义 Spatial 索引。有关指示，参见第43页的『第6章 创建 Spatial 索引』。
- 在应用程序中调用 db2gse.gse_enable_idx 存储过程。有关此存储过程的信息，参见第57页的『第9章 存储过程』。
- 在 USING 子句中用 **spatial_index** 函数发出 **db2 create index** 命令。例如:

  ```
  create index storesx1 on customers (loc) using spatial_index(10e0, 100e0, 1000e0)
  ```

Spatial 数据的性质要求数据库设计者理解它的相对大小分配。设计者必须确定用来创建 Spatial 索引的网格级的最优大小和数目。

通过增加单元大小输入网格级: <网格级 1>、<网格级 2> 和 <网格级 3>。因此，第二级必须具有比第一级更大的单元大小，第三级必须具有比第二级更大的单元大小。第一网格级是强制性的，但可用双精度零值 (0.0e0) 禁用第二和第三网格级。

如何生成 Spatial 索引

Spatial 索引是使用*包络*生成的。包络是几何图形自身且表示几何图形的最小和最大 X 和 Y 范围。对于大多数几何图形，包络是一个框，但对于水平和垂直线条，包络是一个两点线条。对于点，包络是点自身。有关包络的更多信息，参见第113页的『包络』。

通过为每个几何图形的包络与网格的交点生成一项或多项，在 Spatial 列上构造 Spatial 索引。将交点记录为几何图形的内部 ID 和相交的网格单元的最小 X 和 Y 坐标。例如，第104页的图7中的多边形与网格在坐标 (20,30)、(30,30)、(40,30)、(20,40)、(30,40)、(40,40)、(20,50)、(30,50) 和 (40,50) 处相交。参见第104页的表38以获取第104页的图7中所有几何图形的最小 X 和 Y 坐标。

若存在多个网格级，DB2 Spatial Extender 尝试使用可能的最低网格级。当某个几何图形在给定网格级与四个或四个以上网格单元相交时，会把该几何图形提升到下一个更高网格级。因此，对于具有 10.0e0、100.0e0 和 1000.0e0 三个网格级的 Spatial 索引， DB2 Spatial Extender 将首先使每个几何图形与 10.0e0 级网格相交。若某个几何图形与四个或四个以上 10.0e0 网格单元相交，则会提升它并使它与 100.0e0 级网格相交。若在 100.0e0 网格级产生了四个或四个以上的交点，则会把该几何图形提升到 1000.0e0 网格级。在 1000.0e0 网格级，必须将交点输入 Spatial 索引，因为这是可能的最高网格级。

第104页的图7说明四种不同类型的几何图形如何与 10.0e 相交。将四种几何图形的所有 23 个交点记录在 Spatial 索引中。

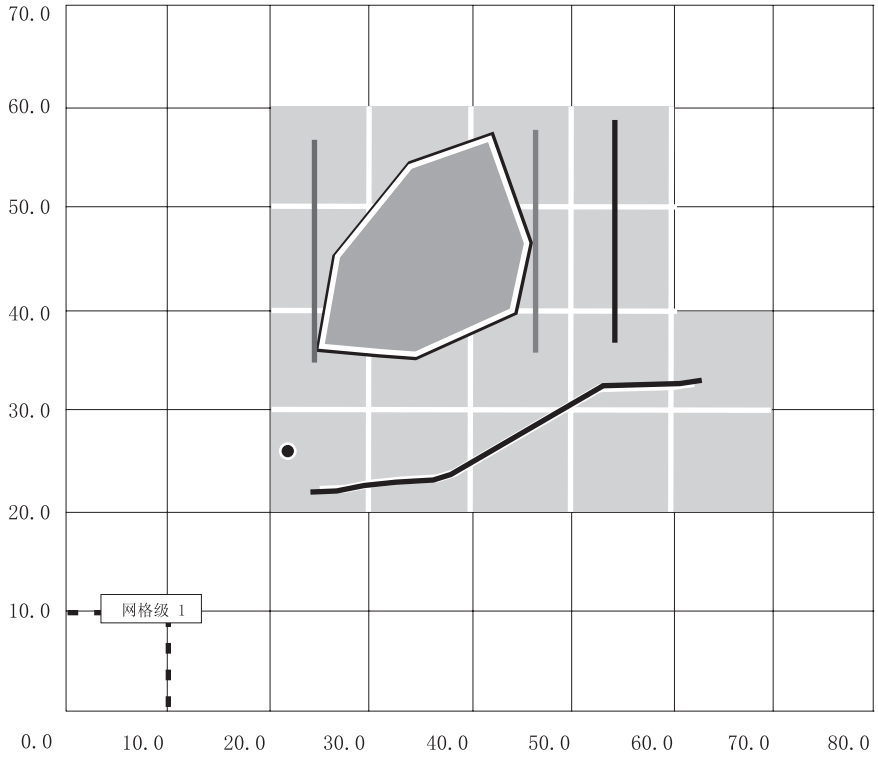


图 7. 10.0e0 网格级的应用

表38列示几何图形及其对应的网格交点。四种不同几何图形类型的包络与 10.0e 网格相交。将与每个网格单元相交的最小 X 和 Y 坐标输入 Spatial 索引中。

表 38. 示例几何图形的 10.0e0 网格单元项

几何图形	网格 X	网格 Y
多边形	20.0	30.0
多边形	30.0	30.0
多边形	40.0	30.0
多边形	20.0	40.0
多边形	30.0	40.0
多边形	40.0	40.0
多边形	20.0	50.0
多边形	30.0	50.0
多边形	40.0	50.0
垂直线条	50.0	30.0

表 38. 示例几何图形的 10.0e0 网格单元项 (续)

几何图形	网格 X	网格 Y
垂直线条	50.0	40.0
垂直线条	50.0	50.0
点	20.0	20.0
水平线条	20.0	20.0
水平线条	30.0	20.0
水平线条	40.0	20.0
水平线条	50.0	20.0
水平线条	60.0	20.0
水平线条	20.0	30.0
水平线条	30.0	30.0
水平线条	40.0	30.0
水平线条	50.0	30.0
水平线条	60.0	30.0

第106页的图8显示如何通过增加网格级 30.0e0 和 60.0e0 将交点数显著减少至八个。在此情况下，将标识为几何图形 1 的多边形提升到网格级 30.0e0，将标识为几何图形 4 的线条提升到网格级 60.0e0。这两个几何图形在 10.0e0 网格级分别有九个和十个交点，而在提升之后它们只有两个交点。

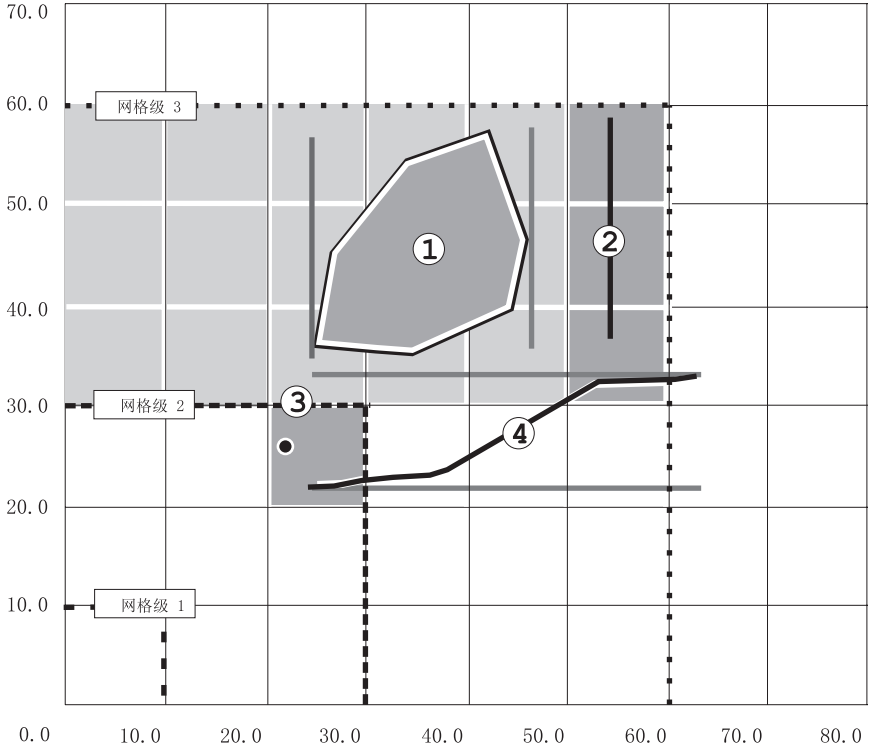


图 8. 添加网格级 30.0e0 和 60.0e0 的效果. 标识为几何图形 1 的多边形的包络与九个网格单元相交。标识为几何图形 2 的垂直线条的包络与三个网格单元相交。标识为几何图形 3 的点的包络只与一个网格单元相交。标识为几何图形 4 的线条的包络与十个网格单元相交。

DB2 Spatial Extender 采用 CREATE INDEX 语句中指定的网格级参数并检查每个 Spatial 对象，以确定对象所在的网格块的坐标和数目。在图8中，用递增的线宽和不同的灰度显示网格级 10.0e0、30.0e0 和 60.0e0。将垂直线条和点包络单元交点输入 10.0e0 网格级的索引，因为这二者生成的交点都少于四个。多边形与九个 10.0e0 网格单元相交，因此它被提升到 30.0e0 网格级。在此网格级，多边形与两个网格单元相交，这两个网格单元被输入索引。标识为几何图形 4 的线条与十个 10.0e0 网格单元相交，因此它被提升到 30.0e0 网格级。然而在此网格级，它与六个网格单元相交，因此它再次被提升到 60.0e0 网格级，在此网格级它生成两个交点。然后线条 60.0e0 网格交点被输入索引。若线条在此网格级生成四个或四个以上交点，仍会将这些交点输入索引，因为这是可将几何图形提升到的最高网格级。

表 39. 三层索引中的几何图形的交点

几何图形	网格 X	网格 Y
网格级 1（10.0e0 网格大小）中垂直线条和点之间的交点		

表 39. 三层索引中的几何图形的交点 (续)

几何图形	网格 X	网格 Y
2	50.0	30.0
2	50.0	40.0
2	50.0	50.0
3	20.0	20.0
网格级 2（30.0e0 网格大小）中多边形的交点		
1	0.0	30.0
1	30.0	30.0
网络级 3（60.0e0 网格大小）中线条的交点		
4	0.0	0.0
4	60.0	0.0

DB2 Spatial Extender 实际上不会创建任何种类的多边形网格结构。DB2 Spatial Extender 通过将原点定义在列的 Spatial 参考系的 X,Y 偏移处以参数方式表示每个网格级。它然后将网格扩展到正的坐标空间。 DB2 Spatial Extender 使用参数网格在数学上生成交点。

使用 Spatial 索引的指南

DB2 Spatial Extender 使用 Spatial 索引来提高 Spatial 查询的性能。考虑最基本和可能最流行的 Spatial 查询 – 框查询。此查询要求 DB2 Spatial Extender 返回完全在或部分在用户定义框内的所有几何图形。若索引不存在，DB2 Spatial Extender 必须将所有几何图形与框进行比较。然而，DB2 Spatial Extender 可通过索引找出满足下列条件的所有索引项：左下角坐标大于或等于框的左下角左坐标且右上角坐标小于或等于框的右上角坐标。因为索引是按此坐标系排序的，所以 DB2 Spatial Extender 能快速获取侯选几何图形的列表。刚才描述的过程称作*第一次处理*。

*第二次处理*确定每个侯选几何图形的包络是否与该框相交。因为其网格单元的包络与该框相交而符合第一次处理的几何图形可能自身具有不与该框相交的包络。

*第三次处理*将侯选几何图形的实际坐标与框比较，以确定该几何图形的任何部分是否实际上在框内。最后这个相当复杂的比较过程对由全体几何图形的子集组成的侯选几何图形的列表进行操作，该子集已通过前两遍处理显著减少。

除 EnvelopesIntersect 函数外，所有 Spatial 查询都会执行这三次处理。EnvelopesIntersect 函数只执行前两次处理。EnvelopesIntersect 函数是为下列显示操作设计的，这些显示操作经常利用它们自己的内置剪切例程因而不需要进行第三次处理。

选择网格单元大小

几何图形包络的不规则形状使选择网格单元大小变得复杂。因为这种不规则性，某些几何图形包络与几个网格相交，而其他包络则落在单个网格单元内。反之，根据数据的 Spatial 分布，某些网格单元与许多几何图形包络相交。

为使 Spatial 索引顺利工作，必须选择正确的网格数目和大小。考虑包含大小一致的几何图形的 Spatial 列。在此情况下，单个网格级就足够了。从包围平均几何图形包络的网格单元大小开始。当测试您的应用程序时，可能发现增加网格单元大小会提高查询的性能。这是因为每个网格单元包含多个几何图形，且第一遍处理能更快地废弃不合格的几何图形。然而，您将发现当继续增加单元大小时，性能将开始下降。这是因为最终第二遍处理将必须处理更多侯选者。

选择级数

若您想要建立索引的对象具有大约相同的相对大小，则可使用单个网格级。虽然这是正确的，但不是所有列都将包含具有相同相对大小的几何图形。通常可将 Spatial 列的几何图形分为几个大小间隔。例如，考虑一个道路网络，可将其中的几何图形分为街道、主要道路和公路。所有街道的长度大约相同，可分为一个大小间隔。主要道路和公路也是这样。因此，表示一个大小间隔的街道可分为第一网格级，道路网络可分为第二网格级，主要公路可分为第三网格级。另一示例包括一个县区列，该县区包含被更大的郊区包围的较小市区群。在此例中，有两个大小间隔和两个网格级，一个用于较小的市区，另一个用于较大的郊区。这些情况非常普通且需要使用多级网格。

要选择每个网格级的单元大小，选择比每个大小间隔稍大的网格单元大小。通过对 Spatial 列执行查询测试索引。

每个附加的网格级需要额外的索引扫描。尝试稍微将网格调大或调小，以确定是否可获得明显的性能提高。

第13章 几何图形和相关的 Spatial 函数

本章讨论由坐标组成并表示地形的信息单元，称作*几何图形*。本章也介绍 Spatial 函数，这些 Spatial 函数将几何图形作为输入，并返回结果以帮助您分析地形和在地理信息系统之间移动 Spatial 数据。涉及的主题如下:

- 几何图形的性质
- 几何图形的特性；返回与这些特性相关的信息的函数
- 可用具体例子说明的几何图形；处理这些几何图形的函数
- 具有以下功能的函数:
  - 显示地形之间的关系和比较
  - 生成几何图形
  - 将几何图形值转换为可调入和可调出的格式

关于几何图形

Oxford American Dictionary 将*几何学* 定义为 “处理线、角、表面和立体的特性和关系的数学分支”。1997 年 8 月 11 日， Open GIS Consortium Inc. (OGC) 在其出版物 *Open GIS Features for ODBC (SQL) Implementation Specification* 中，给出该术语的另一个定义。*几何图形*一词用来表示几何地形，几千年来，制图人员使用这些几何地形来映射现实世界。几何学的这一新含义是一种很抽象的定义，它可能是 “表示地面上的地形的一个点或点的集合”。

在 DB2 Spatial Extender 中，几何图形的*操作*定义可能是 “地形的模型”。该模型可以用地形的坐标表示，有时也可以用直观的符号表示。该模型可传达信息；例如，坐标标识地形相对于固定参考点的位置，符号描述其形状。另外，该模型可用来产生信息；例如，ST_Overlaps 函数可将两个接近的区域的坐标作为输入，并返回关于这两个区域是否重叠的信息。

几何图形所表示的地形的坐标被看作是该几何图形的*特性*。有几种几何图形还具有其他特性；例如:

- *内部*表示几何图形所表示的地形的内容。
- *外部*表示地形周围的空间。
- *边界*表示内容结束而周围空间开始的分界线。

这些特性及附加特性在第111页的『几何图形特性及相关的函数』中讨论。

DB2 Spatial Extender 支持的几何图形形成图9中所示的分层。该分层的六个成员均可用具体例子说明；它们可用直观符号表示，这些符号也在图中显示。

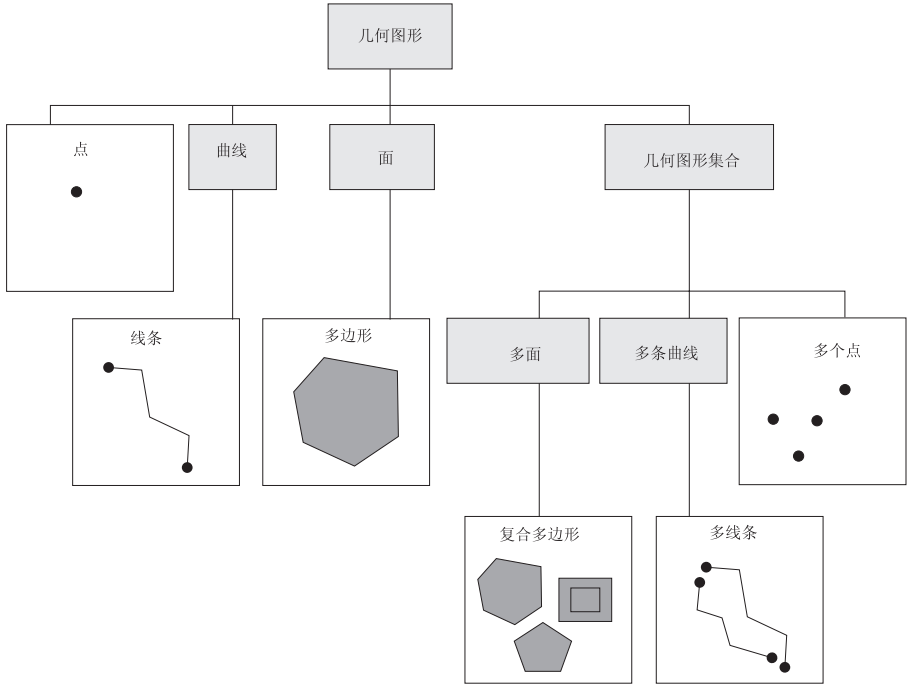


图 9. DB2 Spatial Extender 支持的几何图形分层. 可用具体例子说明的几何图形可用直观符号表示。这些符号显示在这些几何图形名称下面。

如图9所示，称作*几何图形*的超类是分层的根。子类分成两类：基本几何图形子类和同类集合子类。基本几何图形包括:

- *点*，它表示离散的地形，这些地形占据了东西坐标线（如纬线）与南北坐标线（如经线）的相交的位置。例如，假设大比例尺地图上的标志法表明地图上每个城市都位于一条纬线与一条经线的交点。按照此比例尺，每个城市可用一个点来表示。
- *线条*，它表示线性地形（例如，街道、渠道和管道线）。
- *多边形*，它表示多边地形（例如，福利区、森林和野生动物栖息地）。

同类集合包括:

- *多个点*，它表示多部分地形，其组分分别位于东西坐标线和南北坐标线的交点（例如一个群岛，其成员分别位于纬线和经线的交点）。

- *多线条*，它表示由线性单元或组分（例如，江河系统和公路系统）组成的多部分地形。
- *复合多边形*，它表示由多边单元或组分（例如，特定区域内的集中式耕地，或湖泊系统）组成的多部分地形。

正如其名称所暗示的，同类集合是基本几何图形的集合。除了具有基本几何图形的特性之外，同类几何图形还具有一些自己的特性。

DB2 Spatial Extender 支持的 Spatial 数据类型是第110页的图9中所示的几何图形的实现工具。有关这些数据类型的说明，参见第29页的『关于 Spatial 数据类型』。

几何图形特性及相关的函数

本节描述几何图形的特性以及与这些特性相关的 Spatial 函数。本节从核心特性开始:

- 几何图形所属的类
- X 和 Y 坐标

本节还说明:

- Z 坐标
- 度量单位
- 几何图形的内部、边界和外部
- 简单或非简单性质
- 空或非空性质
- 几何图形的包络
- 维数
- 几何图形的相关 Spatial 参考系的标识符

类

每个几何图形均属于第110页的图9中所示的分层中的一个类。如第109页的『关于几何图形』中所示，分层中的六个子类 — 点、线条、多边形、多个点、多线条以及复合多边形 — 均可用具体例子说明。超级类及其他子类不可用具体例子说明。

ST_GeometryType 函数接受几何图形，并以字符串的形式返回可用具体例子说明的子类。有关更多信息，参见第200页的『ST_GeometryType』。

ST_IsValid 函数接受已指定为 ST_Geometry 数据类型的几何图形。若几何图形有效，该函数返回 1 (TRUE)；若几何图形无效，该函数返回 0 (FALSE)。有关更多信息，参见第216页的『ST_IsValid』。

X 和 Y 坐标

*X 坐标值*表示相对于参考点以东或以西的位置。*Y 坐标值*表示相对于参考点以北或以南的位置。有关更多信息，参见第5页的『Spatial 数据的性质』和第23页的『关于坐标系和 Spatial 参考系』。

Z 坐标

有些几何图形具有相关的高度或深度。形成地形的几何图形的每个点均可包括可选的 Z 坐标，该坐标表示距地球表面的垂直高度或深度。

Is3d 谓词函数接受几何图形，并且当函数具有 Z 坐标时返回 1 (TRUE)，否则返回 0 (FALSE)。有关更多信息，参见第149页的『Is3d』。

度量单位

度量单位是一个值，它传达关于地形的信息，并与定义地形位置的坐标一起存储。例如，假设您要在 GIS 中表示运输系统。若想要应用程序处理那些表示线性距离或里程标的值，则可以将这些值与定义系统位置的坐标一起存储。度量单位存储为双精度数。

IsMeasured 谓词接受几何图形，并且当几何图形包含度量单位时返回 1 (TRUE)，否则返回 0 (FALSE)。有关更多信息，参见第150页的『IsMeasured』。

内部、边界和外部

所有几何图形均占据由其内部、边界和外部定义的空间内的某个位置。几何图形的外部是未被该几何图形占据的所有空间。几何图形的边界作为其内部和外部的交界。内部是几何图形占据的空间。子类直接继承内部和外部特性，但每个子类的边界特性不同。

ST_Boundary 函数接受几何图形，并返回表示源几何图形边界的几何图形。有关更多信息，参见第172页的『ST_Boundary』。

简单或非简单

有些几何图形子类（线条、多个点和多线条）是简单或非简单的。若某个子类遵守该子类上的所有拓扑规则，则它是简单的，否则是非简单的。若线不与其内部

相交，则它是简单的。若多个点的元素均不在同一坐标空间中，则它是简单的。若多线条的元素均不与它自己的内部相交，则它是简单的。

ST_IsSimple 谓词函数接受几何图形，并且当几何图形是简单的时返回 1 (TRUE)，否则返回 0 (FALSE)。有关更多信息，参见第215页的『ST_IsSimple』。

空或非空

若几何图形没有任何点，则为空。空几何图形的包络、边界、内部和外部均为空。 空几何图形总是简单的，并且可具有 Z 坐标或度量单位。空线条和空多线条的长度均为 0。空多边形和空复合多边形的面积均为 0。

ST_IsEmpty 谓词函数接受几何图形，并且当几何图形为空时返回 1 (TRUE)，否则返回 0 (FALSE)。 有关更多信息，参见第212页的『ST_IsEmpty』。

包络

几何图形的包络是由最小和最大 (X,Y) 坐标形成的边界几何图形。除下列情况之外，大多数几何图形的包络形成边界矩形:

- 点的包络是点本身，因为其最小和最大坐标相同。
- 水平或垂直线条的包络是由源线条边界（端点）表示的线条。

ST_Envelope 函数接受几何图形，并返回表示其包络的边界几何图形。有关更多信息，参见第191页的『ST_Envelope』。

维数

几何图形的维数可以是 0、1 或 2。这些维数列示如下:

0 既无长度，也无面积

1 具有长度

2 包含面积

点和多点子类的维数均为零。点表示可用单个坐标创建的维数地形，而多点子类表示必须用一簇分离坐标创建的数据。

线条和多线条子类的维数均为 1。它们存储路段、支流系统以及自然界中任何线性的其他地形。

多边形和复合多边形子类的维数均为 2。其周边包围一个可定义区域（如森林、小块陆地和水体）的地形可以用多边形或复合多边形数据类型来描绘。

维数不仅在作为子类的特性时很重要，而且在确定两个地形的 Spatial 关系方面也起重要作用。所产生的地形的维数确定操作是否成功。DB2 Spatial Extender检查地形的维数以确定应该如何比较这些地形。

ST_Dimension 函数接受几何图形，并以整数形式返回其维数。有关更多信息，参见第185页的『ST_Dimension』。

Spatial 参考系标识符

Spatial 参考系标识每个几何图形的坐标变换。

为数据库所知的所有 Spatial 参考系均可以通过DB2GSE.SPATIAL_REF_SYS 目录视图来存取。有关此视图的信息，参见第97页的『第11章 目录视图』。

ST_SRID 函数接受几何图形，并以整数形式返回其 Spatial 参考系标识符。有关更多信息，参见第249页的『ST_SRID』。

ST_Transform 函数将几何图形指定至一个 Spatial 参考系，该 Spatial 参考系不是该几何图形当前被指定至的 Spatial 参考系。有关更多信息，参见第254页的『ST_Transform』。

可用具体例子说明的几何图形及相关的函数

本节概述可用具体例子说明的几何图形的六个子类，并描述处理这些子类的函数。这些子类是:

- 点
- 线条
- 多边形
- 多个点
- 多线条
- 复合多边形

有关这些子类所属分层以及与这些子类相关的直观符号的举例说明，参见第110页的图9。

点

点是零维数几何图形，它占据坐标空间中的单个位置。点包括定义此位置的 X 坐标和 Y 坐标。它也可以包括 Z 坐标和度量单位。

点是简单的，并具有空边界。点通常用来定义油井、陆标和标高等地形。

专门处理点子类的函数:

ST_Point

接受 X 坐标、其相关的 Y 坐标以及这些坐标所属 Spatial 参考系的标识符，并返回这些坐标定义的点。有关更多信息，参见第240页的『ST_Point』。

ST_CoordDim

返回一个值，该值指示点包含什么坐标，以及该点是否还包含度量单位。此值称为*坐标维数*。可能的坐标维数为:

2 点由 X 坐标和 Y 坐标组成。

3 点由 X 坐标、Y 坐标和 Z 坐标组成。

4 点由 X 坐标、Y 坐标、Z 坐标和度量单位组成。

有关更多信息，参见第180页的『ST_CoordDim』。

ST_PointFromText

接受点的 OGC 公认文本 (WKT) 表示，并返回该点。有关更多信息，参见第237页的『ST_PointFromText』。

ST_X 返回 ST_Point 数据类型的 X 坐标值（双精度数）。有关更多信息，参见第260页的『ST_X』。

ST_Y 返回 ST_Point 数据类型的 Y 坐标值（双精度数）。有关更多信息，参见第261页的『ST_Y』。

Z 返回 ST_Point 数据类型的 Z 坐标值（双精度数）。有关更多信息，参见第262页的『Z』。

M 返回 ST_Point 数据类型的度量单位（双精度数）。有关更多信息，参见第157页的『M』。

线条

线条是存储为点序列（它定义一条线性插值路径）的一维对象。若线条不与其内部相交，则它是简单的。闭合线条的端点（边界）占据空间中同一个点。若线条闭合，并且不与其内部相交，则它是一个环。除从超类几何图形继承的其他特性之外，线条还具有长度。线条通常用来定义公路、江河和电源线等线性地形。

其起点和终点相同的简单线条称为*环*。

端点通常形成线条的边界，除非线条是闭合的（此时边界为空）。线的内部是端点之间的连接路径，除非它是闭合的（此时内部是连续的）。

处理线条的函数:

ST_StartPoint

接受线条并返回该线条的第一个点。有关更多信息，参见第250页的『ST_StartPoint』。

ST_EndPoint

接受线条并返回该线条的最后一个点。有关更多信息，参见第190页的『ST_Endpoint』。

ST_PointN

接受线条以及第 *n* 个点的索引，并返回该点。有关更多信息，参见第241页的『ST_PointN』。

ST_Length

接受线条并返回其长度（双精度数）。有关更多信息，参见第218页的『ST_Length』。

ST_NumPoints

接受线条并返回其序列中的点数（整数）。有关更多信息，参见第232页的『ST_NumPoints』。

ST_IsRing

接受线条，并且当线条是环时返回 1 (TRUE)，否则返回 0 (FALSE)。有关更多信息，参见第214页的『ST_IsRing』。

ST_IsClosed

接受线条，并且当线条闭合时返回 1 (TRUE)，否则返回 0 (FALSE)。有关更多信息，参见第210页的『ST_IsClosed』。

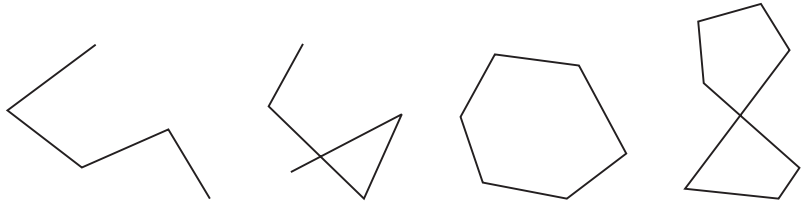


图 10. 线条对象.

1. 简单的非闭合线条。
2. 非简单的非闭合线条。
3. 闭合的简单线条，因此是一个环。
4. 闭合的非简单线条。它不是环。

多边形

多边形是存储为点序列的二维表面，该点序列定义该表面的外部边界环以及 0 个或多个内部环。多边形的环不能重叠。因此，根据定义，多边形总是简单的。 多边形通常定义小块陆地、水体和其他具有 Spatial 范围的地形。

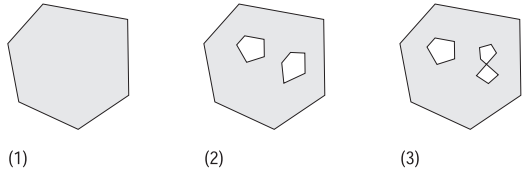


图 11. 多边形.

1. 其边界由一个外部环定义的多边形。
2. 其边界由一个外部环和两个内部环定义的多边形。内部环之内的区域是多边形外部的一部分。
3. 合法多边形，因为环在单个切点相交。

外部和任何内部环定义多边形的边界，环之间的包围的空间定义多边形的内部。多边形的环可以在切点相交，但从不交叉。除从超类几何图形继承的其他特性之外，多边形还具有面积。

处理多边形的函数:

ST_Area

接受多边形并返回其面积（双精度数）。有关更多信息，参见第168页的『ST_Area』。

ST_ExteriorRing

接受多边形并返回其外部环（线条）。有关更多信息，参见第194页的『ST_ExteriorRing』。

ST_NumInteriorRing

接受多边形并返回该多边形包含的内部环数。有关更多信息，参见第231页的『ST_NumInteriorRing』。

ST_InteriorRingN

接受多边形和索引，并返回第 *n* 个内部环（线条）。有关更多信息，参见第202页的『ST_InteriorRingN』。

ST_Centroid

接受多边形并返回多边形范围的中心点。有关更多信息，参见第175页的『ST_Centroid』。

ST_PointOnSurface

接受多边形并返回保证在多边形表面上的点。有关更多信息，参见第242页的『ST_PointOnSurface』。

ST_Perimeter

接受多边形并返回多边形表面的周长。有关更多信息，参见第235页的『ST_Perimeter』。

多个点

多个点是点的集合，并且象其元素一样，它的维数为 0。若多个点的元素均不在同一坐标空间中，则它是简单的。多个点的边界为空。多个点可用来定义架空广播辐射图和流行病蔓延事故等现象。

处理多个点的函数:

ST_NumGeometries

接受同类集合，并返回它包含的基本几何图形元素的数目。有关更多信息，参见第230页的『ST_NumGeometries』。

ST_GeometryN

接受同类集合和索引，并返回第 n 个基本几何图形。有关更多信息，参见第199页的『ST_GeometryN』。

多线条

多线条是线条的集合。若多线条仅在线条元素的端点相交，则多线条是简单的。若线条元素的内部相交，则多线条是非简单的。

多线条的边界是线条元素的非相交端点。若多线条的所有线条元素均闭合，则多线条闭合。若所有元素的端点均相交，则多线条的边界为空。除从超类几何图形继承的其他特性之外，多线条还具有长度。多线条用来定义河流或道路网络。

处理多线条的函数:

ST_Length

接受多线条，并返回其所有线条元素的累计长度（双精度数）。有关更多信息，参见第218页的『ST_Length』。

ST_IsClosed

接受多线条，并且当多线条闭合时返回 1 (TRUE)，否则返回 0 (FALSE)。有关更多信息，参见第210页的『ST_IsClosed』。

ST_NumGeometries

接受同类集合，并返回它包含的基本几何图形元素的数目。有关更多信息，参见第230页的『ST_NumGeometries』。

ST_GeometryN

接受同类集合和索引，并返回第 n 个基本几何图形。有关更多信息，参见第199页的『ST_GeometryN』。

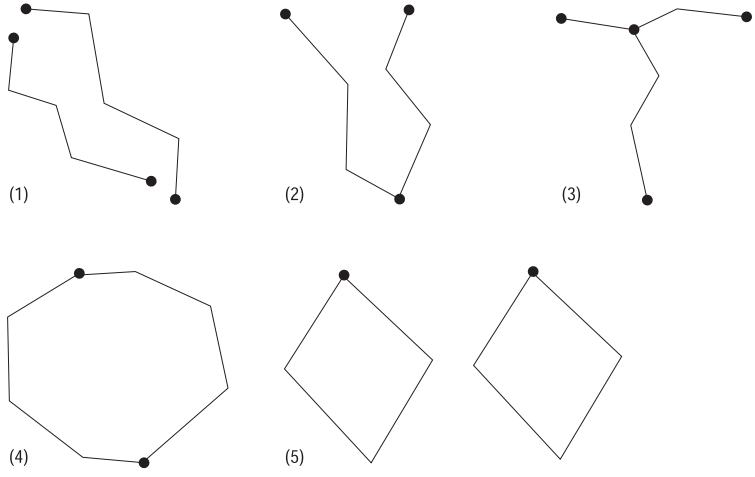
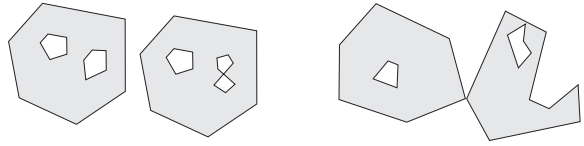


图 12. 多线条.

1. 简单多线条，其边界由其两个线条元素的四个端点定义。
2. 简单多线条，因为仅线条元素的端点相交。边界由两个不相交的端点定义。
3. 非简单多线条，因为其中一个线条元素的内部相交。此多线条的边界由四个端点定义，包括交点。
4. 简单的非闭合多线条。因为其元素线条不是闭合的，所以它不是闭合的。因为任何元素线条的内部均不相交，所以它是简单的。
5. 简单的闭合多线条。因为其元素都是闭合的，所以它是闭合的。因为其元素线均不在内部相交，所以它是简单的。

复合多边形

复合多边形的边界是其元素的内部环和外部环的累计长度。复合多边形的内部定义为其元素多边形的累计内部。复合多边形元素的边界只能在切点相交。除从超类几何图形继承的其他特性之外，复合多边形还具有面积。复合多边形定义森林层或不相邻的小块陆地（如群岛）等地形。



(1) (2)

图 13. 复合多边形.

1. 具有两个多边形元素的复合多边形。边界由两个外部环和三个内部环定义。
2. 具有两个多边形元素的复合多边形。边界由两个外部环和两个内部环定义。这两个多边形元素在切点相交。

处理复合多边形的函数:

ST_Area

接受复合多边形，并返回其多边形元素的累计面积（双精度数）。有关更多信息，参见第168页的『ST_Area』。

ST_Centroid

接受复合多边形，并返回其几何加权中心点。有关更多信息，参见第175页的『ST_Centroid』。

ST_NumGeometries

接受同类集合，并返回它包含的基本几何图形元素的数目。有关更多信息，参见第230页的『ST_NumGeometries』。

ST_GeometryN

接受同类集合和索引，并返回第 n 个基本几何图形。有关更多信息，参见第199页的『ST_GeometryN』。

显示关系和比较、生成几何图形和转换值的格式的函数

前面几节介绍了三种 Spatial 函数:

- 与几何图形的特性相关的函数
- 与特定几何图形相关的函数

本节介绍另外三种函数:

- 确定地形相关或比较方式的函数
- 生成新几何图形的函数
- 将几何图形的值转换为可调入或调出的格式的函数

显示地形之间的关系和比较的函数

几种 Spatial 函数返回关于地形互相相关或互相比较的方式的信息。这些函数（称作*谓词*）大多数是布尔函数。本节先概括地描述这些谓词，然后分别讨论每个函数。

谓词函数

若比较符合该谓词函数的标准，则该函数返回 1 (TRUE)；若比较失败，则该函数返回 0 (FALSE)。测试 Spatial 关系的谓词比较类型或维数可能不同的几何图形对。

谓词比较提交的几何图形的 X 和 Y 坐标。忽略 Z 坐标和度量单位（若有的话）。这样就能够将具有 Z 坐标或度量单位的几何图形与没有 Z 坐标或度量单位的几何图形进行比较。

维数扩展 9 相交模型 (DE-9IM)[1]是一种数学方法，它定义不同类型和维数的几何图形之间的成对 Spatial 关系。此模型将所有类型的几何图形之间的 Spatial 关系表示为其内部、边界和外部的成对相交，并考虑产生的交集的维数。

给定几何图形 *a* 和 *b*: I(*a*)、B(*a*) 和 E(*a*) 表示 *a* 的内部、边界和外部。I(b)、B(b) 和 E(b) 表示 *b* 的内部、边界和外部。 I(a)、B(a) 和 E(a) 与 I(b)、B(b) 和 E(b) 的交集产生 3 x 3 矩阵。每个交集可产生具有不同维数的几何图形。例如，两个多边形的边界交集由一个点和一条线条组成，此时 dim 函数返回的最大维数为 1。

dim 函数返回值 -1、0、1 或 2。-1 对应空集或 dim(null)，当找不到交集时，返回该值。

	内部	边界	外部
内部	dim(I(a) ∩ I(b))	dim(I(a) ∩ B(b))	dim(I(a) ∩ E(b))
边界	dim(B(a) ∩ I(b))	dim(B(a) ∩ B(b))	dim(B(a) ∩ E(b))
外部	dim(E(a) ∩ I(b))	dim(E(a) ∩ B(b))	dim(E(a) ∩ E(b))

通过将谓词的结果与表示 DE-9lM 的可接受值的模式矩阵进行比较，可以了解或验证 Spatial 关系谓词的结果。

---

1. DE-9IM 由 Clementini 和 Felice 开发，他们在维数上扩展了 Egenhofer 和 Herring 的 9 相交模型。DE-9IM 是以下四位作者合作的成果：Clementini、Eliseo、Di Felice 和 van Osstrom。他们在以下论文中发表了该模型：″A Small Set of Formal Topological Relationships Suitable for End-User Interaction″, D. Abel and B.C. Ooi (Ed.), *Advances in Spatial Database—Third International Symposium. SSD '93*. LNCS 692. Pp. 277-295。9 相交模型是 Springer-Verlag Singapore (1993) Egenhofer M.J. 和 Herring, J. 在以下论文中发表的：″Categorizing binary topological relationships between regions, lines, and points in geographic databases″, *Department of Surveying Engineering*, University of Maine, Orono, ME 1991。

模式矩阵包含每个交集矩阵单元的可接受值。可能的模式值是:

T 必须存在交集， dim = 0、1 或 2。

F 不得存在交集，dim = -1。

***** 是否存在交集并不重要，dim = -1、0、1 或 2。

0 必须存在交集，并且其最大维数必须为 0，dim = 0。

1 必须存在交集，并且其最大维数必须为 1，dim = 1。

2 必须存在交集，并且其最大维数必须为 2，dim = 2。

例如，ST_Within 谓词的如下模式矩阵包括值 T、F 和 *。

表 40. ST_Within 的矩阵. ST_Within 谓词的用于几何图形组合的模式矩阵。

		b		
		内部	**边界**	**外部**
a	**内部**	T	*	F
	边界	*	*	F
	外部	*	*	*

当两个几何图形的内部相交，且 *a* 的内部和边界与 *b* 的外部不相交时， ST_Within 谓词返回 TRUE。所有其他条件都无关紧要。

每个谓词至少有一个模式矩阵，但有些谓词需要多个模式矩阵，以便描述各种几何图形类型组合的关系。

ST_Equals

若同一类型的两个几何图形具有相同的 X,Y 坐标值， ST_Equals 返回 1 (TRUE)。

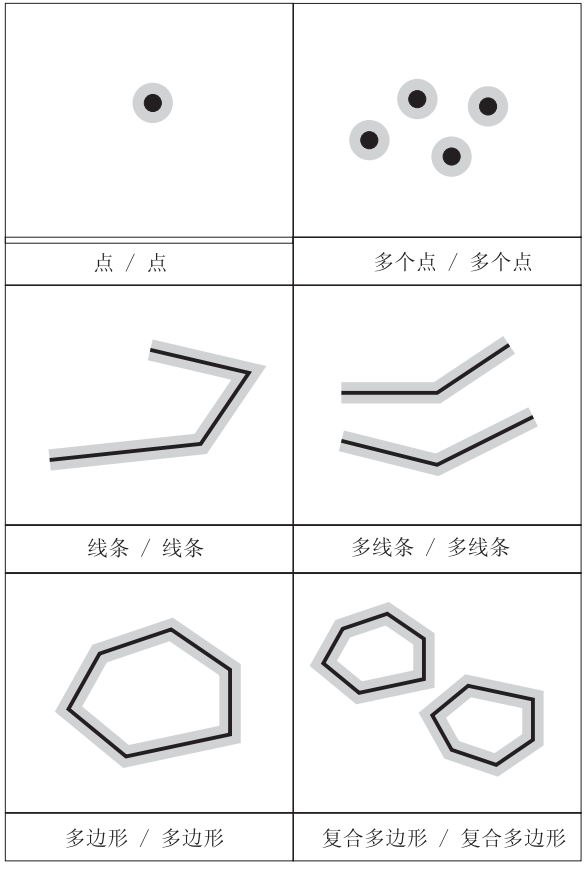


图 14. ST_Equals. 若几何图形具有匹配的 X,Y 坐标，则它们相等。

表 41. 相等矩阵. 相等 DE-9IM 模式矩阵确保内部相交，并且任一几何图形的内部或边界均不与对方的外部相交。

		b		
		内部	**边界**	**外部**
a	**内部**	T	*	F
	边界	*	*	F
	外部	F	F	*

有关更多信息，参见第193页的『ST_Equals』。

ST_OrderingEquals

ST_OrderingEquals 比较两个几何图形，若这两个几何图形相等且坐标次序相同，返回 1 (TRUE)；否则返回 0 (FALSE)。有关更多信息，参见第233页的『ST_OrderingEquals』。

ST_Disjoint

若两个几何图形的交集是空集，则 ST_Disjoint 返回 1 (TRUE)。

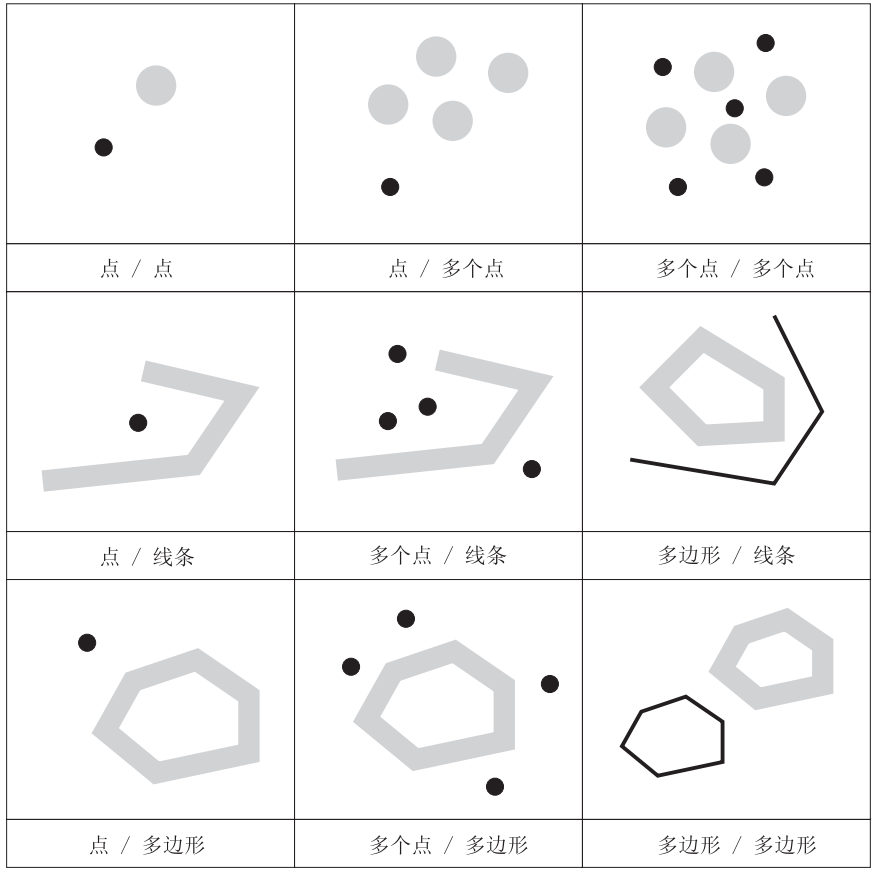


图 15. ST_Disjoint. 若几何图形不以任何方式互相相交，则称它们不相交。

表 42. ST_Disjoint 的矩阵. ST_Disjoint 谓词的模式矩阵只表明任一几何图形的内部或边界均不相交。

		b		
		内部	**边界**	**外部**
a	**内部**	F	F	*
	边界	F	F	*
	外部	*	*	*

有关更多信息，参见第187页的『ST_Disjoint』。

ST_Intersects

若相交不产生空集，则 ST_Intersects 返回 1 (TRUE)。Intersects 返回的结果与 ST_Disjoint 的结果正好相反。

若下列任何模式矩阵的条件返回 TRUE，则 ST_Intersects 谓词返回 TRUE。

表 43. ST_Intersects 的矩阵 (1). 若两个几何图形的内部相交，则 ST_Intersects 谓词返回 TRUE。

		b		
		内部	边界	外部
a	内部	T	*	*
	边界	*	*	*
	外部	*	*	*

表 44. ST_Intersects 的矩阵 (2). 若第一个几何图形的边界与第二个几何图形的边界相交，则 ST_Intersects 谓词返回 TRUE。

		b		
		内部	边界	外部
a	内部	*	T	*
	边界	*	*	*
	外部	*	*	*

表 45. ST_Intersects 的矩阵 (3). 若第一个几何图形的边界与第二个几何图形的内部相交，则 ST_Intersects 谓词返回 TRUE。

		b		
		内部	边界	外部
a	内部	*	*	*
	边界	T	*	*
	外部	*	*	*

表 46. ST_Intersects 的矩阵 (4). 若任一几何图形的边界相交，则 ST_Intersects 谓词返回 TRUE。

		b		
		内部	边界	外部
a	内部	*	*	*
	边界	*	T	*
	外部	*	*	*

有关更多信息，参见第208页的『ST_Intersects』。

EnvelopesIntersect

若两个几何图形的包络相交，则此函数返回 1 (TRUE)。它是一个便利函数，能够高效地实现 ST_Intersects (`ST_Envelope(g1),ST_Envelope(g2)`)。有关更多信息，参见第147页的『EnvelopesIntersect』。

ST_Touches

若两个几何图形的公共点均不与这两个几何图形的内部相交，则 ST_Touches 返回 1 (TRUE)。至少有一个几何图形必须为线条、多边形、多线条或复合多边形。

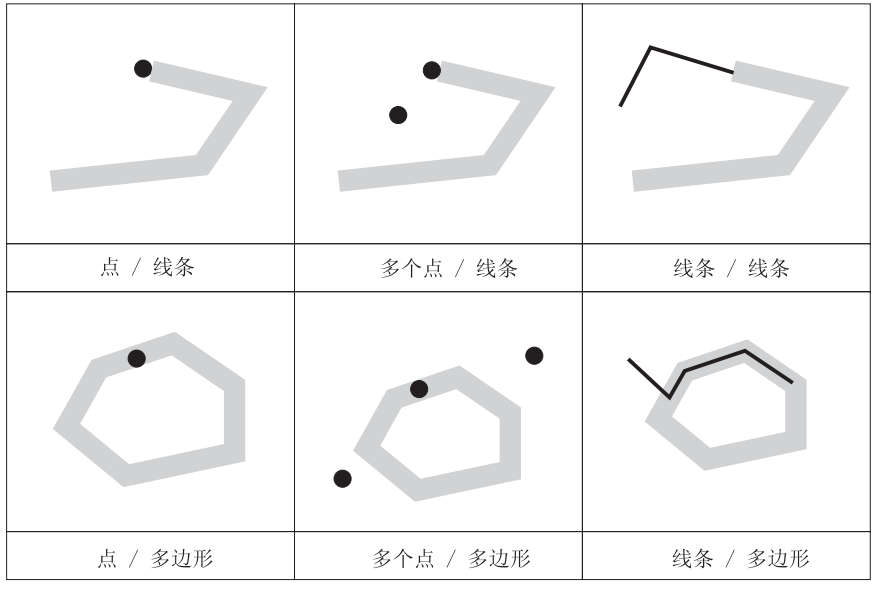


图 16. ST_Touches

这些模式矩阵显示，当几何图形的内部不相交，且任一几何图形的边界与另一几何图形的内部或边界相交时，则 ST_Touches 谓词返回 TRUE。

表 47. ST_Touches 的矩阵 (1)

		b		
		内部	**边界**	**外部**
a	**内部**	F	T	*
	边界	*	*	*
	外部	*	*	*

表 48. ST_Touches 的矩阵 (2)

		b		
		内部	**边界**	**外部**
a	**内部**	F	*	*
	边界	T	*	*
	外部	*	*	*

表 49. ST_Touches 的矩阵 (3)

		b		
		内部	边界	外部
a	内部	F	*	*
	边界	*	T	*
	外部	*	*	*

有关更多信息，参见第253页的『ST_Touches』。

ST_Overlaps

ST_Overlaps 比较具有相同维数的两个几何图形。若这两个几何图形的交集产生与这两个几何图形不同但具有相同维数的几何图形，则该函数返回 1 (TRUE)。

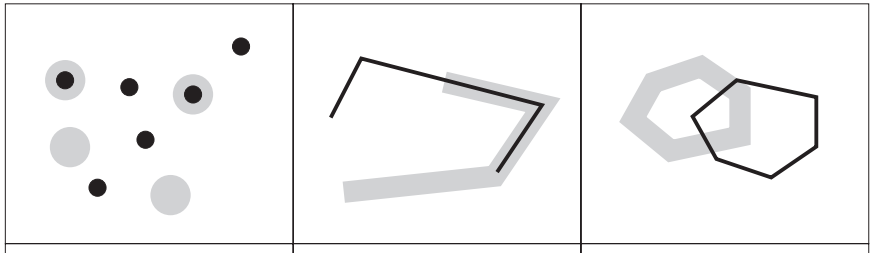


图 17. ST_Overlaps

表50 中的模式矩阵适用于多边形 / 多边形、多个点 / 多个点以及复合多边形 / 复合多边形重叠。对于这些组合，若两个几何图形与对方内部和外部相交，则重叠谓词返回 TRUE。

表 50. ST_Overlaps 的矩阵 (1)

		b		
		内部	边界	外部
a	内部	T	*	T
	边界	*	*	*
	外部	T	*	*

第129页的表51 中的模式矩阵适用于线条 / 线条和多线条 / 多线条重叠。在此情况下，几何图形的交集必须产生维数为 1 的几何图形（另一条线条）。若内部的交集的维数为 1，则 ST_Overlaps 谓词将返回 FALSE，而 ST_Crosses 谓词将返回 TRUE。

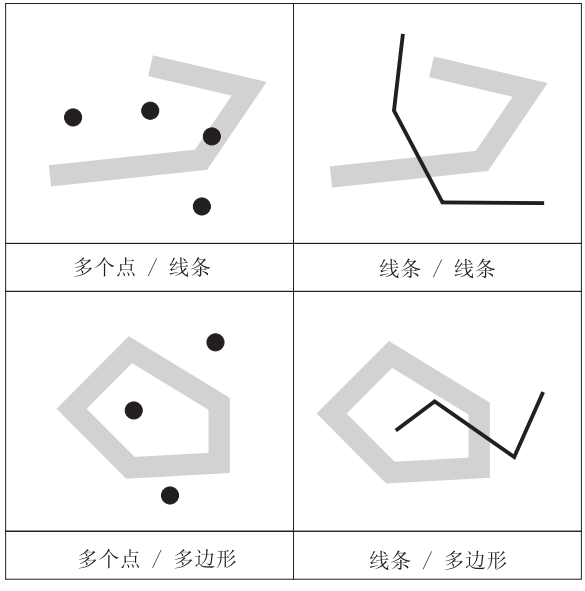
表 51. ST_Overlaps 的矩阵 (2)

		b		
		内部	边界	外部
a	内部	1	*	T
	边界	*	*	*
	外部	T	*	*

有关更多信息，参见第234页的『ST_Overlaps』。

ST_Crosses

若交集产生的几何图形的维数比两个源几何图形的最大维数小 1，且交集在两个源几何图形的内部，则 ST_Crosses 返回 1 (TRUE)。只对多个点 / 多边形、多个点 / 线条、线条 / 线条、线条 / 多边形和线条 / 复合多边形比较， ST_Crosses 返回 1 (TRUE)。



第130页的表52 中的模式矩阵适用于多个点 / 线条、多个点 / 多线条、多个点 / 多边形、多个点 / 复合多边形、线条 / 多边形以及线条 / 复合多边形。该矩阵表明内部必须相交，并且主几何图形 (*a*) 的内部必须与次几何图形 (*b*) 的外部相交。

表 52. ST_Crosses 的矩阵 (1)

		b		
		内部	边界	外部
a	内部	T	*	T
	边界	*	*	*
	外部	*	*	*

表53 中的模式矩阵适用于线条 / 线条、线条 / 多线条以及多线条 / 多线条。该矩阵表明内部交集的维数必须为 0 （相交于一个点）。若此交集的维数为 1（相交于一条线条），则 ST_Crosses 谓词返回 FALSE; 而 ST_Overlaps 谓词返回 TRUE。

表 53. ST_Crosses 的矩阵 (2)

		b		
		内部	边界	外部
a	内部	0	*	*
	边界	*	*	*
	外部	*	*	*

有关更多信息，参见第182页的『ST_Crosses』。

ST_Within

若第一个几何图形完全在第二个几何图形内，则 ST_Within 返回 1 (TRUE)。ST_Within 返回的结果与 ST_Contains 的结果正好相反。

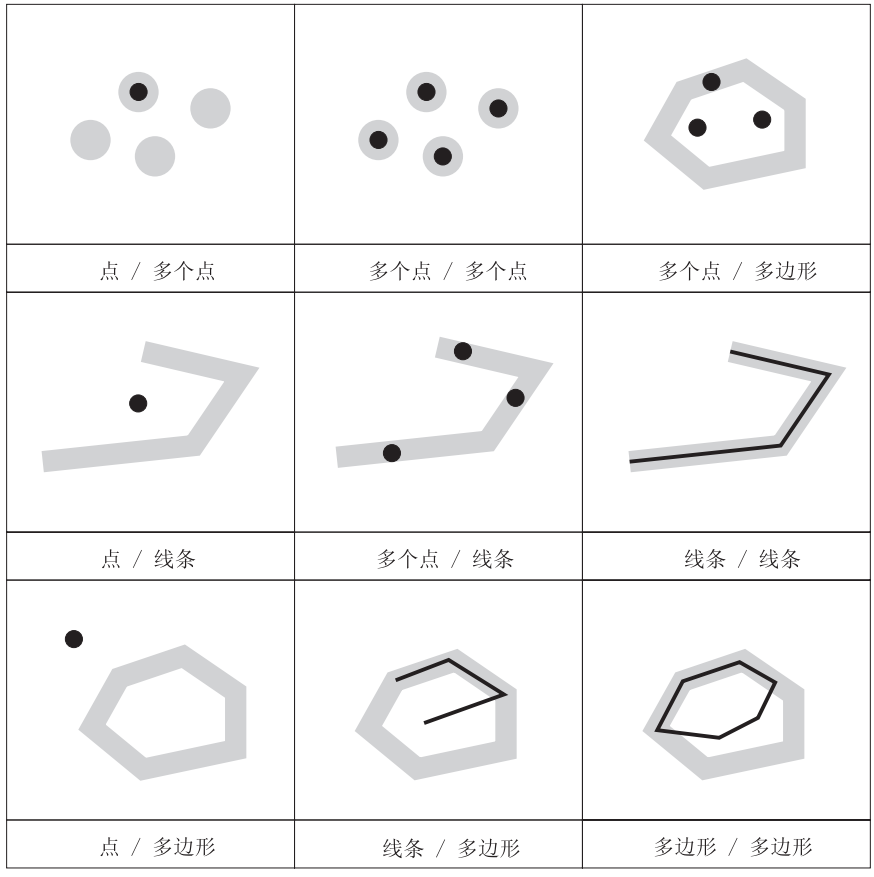


图 18. 之内

ST_Within 谓词模式矩阵表明两个几何图形的内部必须相交，且主几何图形 (*a*) 的内部不得与次几何图形 (*b*) 的外部相交。

表 54. ST_Within 的矩阵

		b		
		内部	**边界**	**外部**
a	**内部**	T	*	F
	边界	*	*	F
	外部	*	*	*

有关更多信息，参见第256页的『ST_Within』。

ST_Contains

若第二个几何图形完全被第一个几何图形包含，则 ST_Contains 返回 1 (TRUE)。ST_Contains 谓词返回的结果与 ST_Within 谓词的结果正好相反。

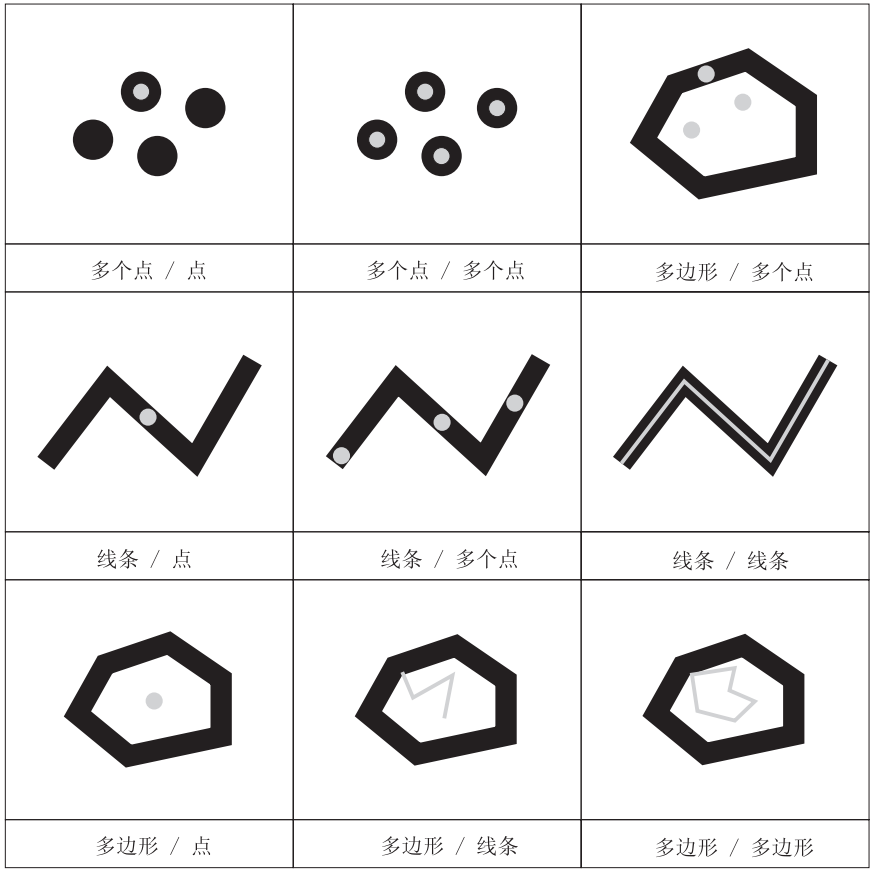


图 19. ST_Contains

ST_Contains 谓词的模式矩阵表明两个几何图形的内部必须相交，并且次几何图形 (*b*) 的内部和边界不得与主几何图形 (*a*) 的外部相交。

表 55. ST_Contains 的矩阵

		b		
		内部	**边界**	**外部**
a	**内部**	T	*	*
	边界	*	*	*
	外部	F	F	*

有关更多信息，参见第177页的『ST_Contains』。

ST_Relate

ST_Relate 比较两个几何图形，若这两个几何图形满足 DE-9IM 模式矩阵字符串指定的条件，则返回 1 (TRUE)；否则该函数返回 0 (FALSE)。有关更多信息，参见第247页的『ST_Relate』。

ST_Distance

ST_Distance 函数报告两个不相交地形之间的最小距离。若地形相交，该函数将报告 0 最小距离。

例如，ST_Distance 可报告飞行器在两个位置之间必须飞行的最短距离。 图20举例说明这一点。

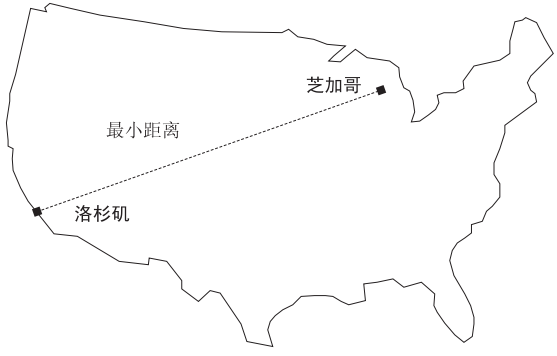


图 20. 两个城市之间的最小距离. ST_Distance 可将 Los Angeles 和 Chicago 两个位置的坐标作为输入，并返回一个表示这两个位置之间的最小距离的值。

有关更多信息，参见第188页的『ST_Distance』。

从现存几何图形生成新几何图形的函数

DB2 Spatial Extender 提供从现存几何图形生成新几何图形的谓词和转换函数。

ST_Intersection

ST_Intersection 函数返回两个几何图形的交集。交集总是以集合的形式返回，该集合的维数是源几何图形的最小维数。例如，对于与多边形相交的线，交集函数返回由该线条与多边形的内部和边界的公共部分构成的多线条。若源线条与具有两个或多个不连续段的多边形相交，则该多线条包含多个线条。若这些几何图形不相交，或者相交产生的维数小于两个源几何图形的维数，则返回空几何图形。

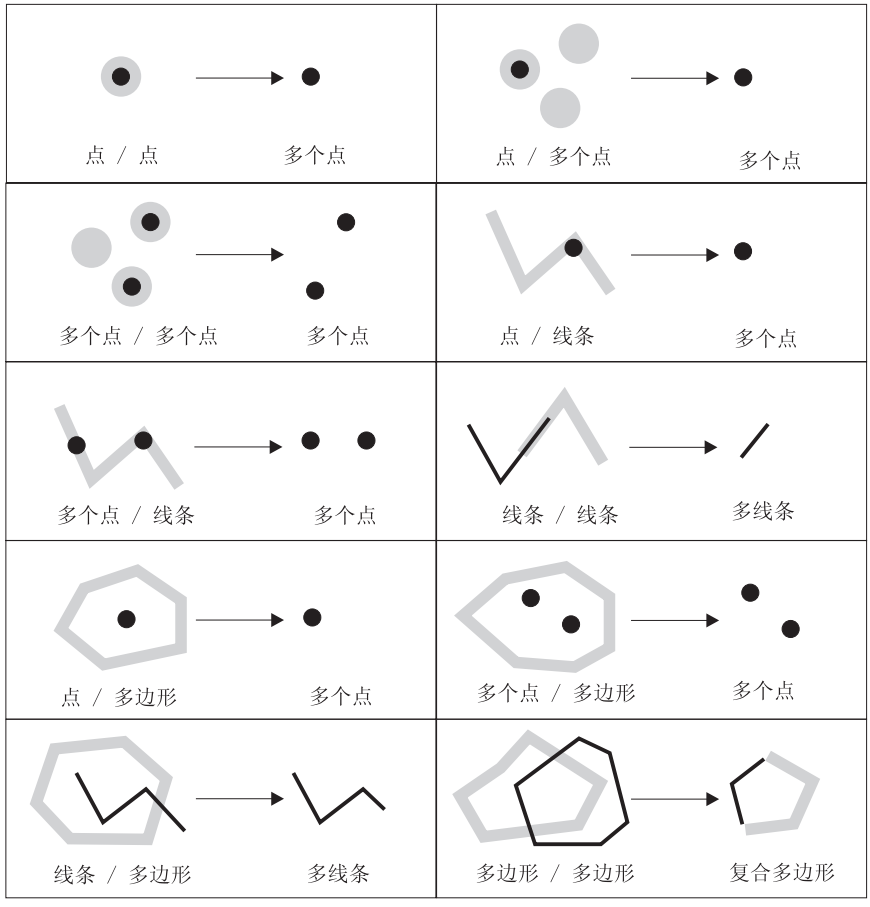


图 21. ST_Intersection. ST_Intersection 函数的示例。

有关更多信息，参见第207页的『ST_Intersection』。

ST_Difference

ST_Difference 函数返回主几何图形的与次几何图形不相交的部分。这是空间的逻辑“与非”。ST_Difference 函数只处理维数相同的几何图形，并返回一个与源几何图形具有相同维数的集合。在源几何图形相等时，返回空几何图形。

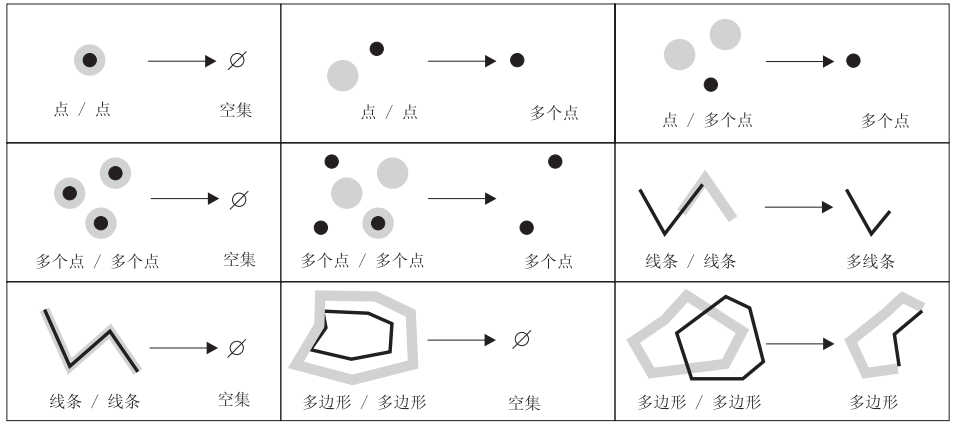


图 22. ST_Difference

有关更多信息，参见第183页的『ST_Difference』。

ST_Union

ST_Union 函数返回两个几何图形的并集。这是空间的逻辑“或”。源几何图形必须具有相同的维数。 ST_Union 总是以集合的形式返回结果。

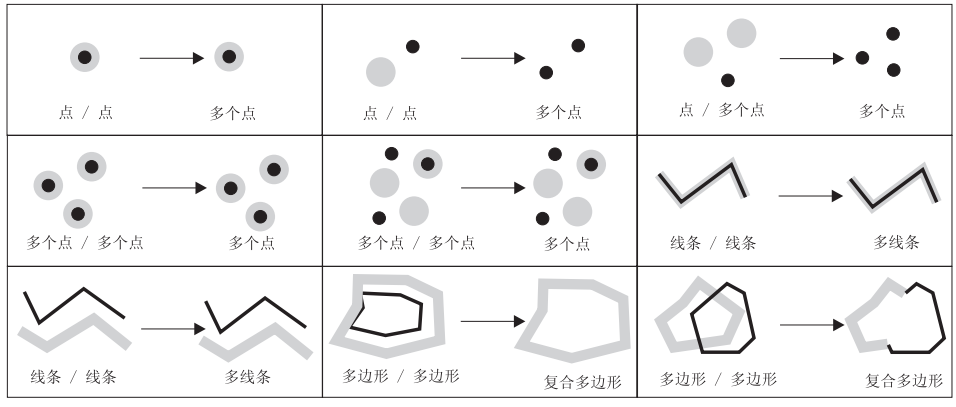


图 23. ST_Union

有关更多信息，参见第255页的『ST_Union』。

ST_Buffer

ST_Buffer 函数通过在指定的距离环绕一个几何图形生成一个几何图形。当缓冲主几何图形时，或者当集合的元素足够近，以致所有缓冲多边形重叠时，产生一个

多边形。然而，当被缓冲的集合的元素之间距离足够远时，将产生各自的缓冲多边形，此时 ST_Buffer 函数返回一个复合多边形。

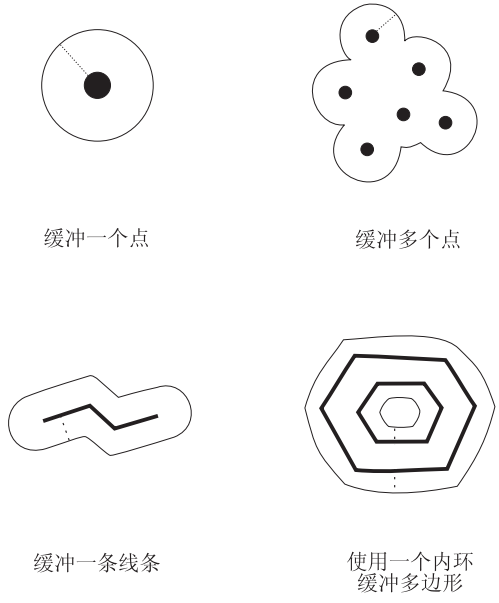


图 24. ST_Buffer

ST_Buffer 函数接受正距离和负距离，然而，仅二维几何图形（多边形和复合多边形）应用负缓冲距离。当源几何图形的维数小于 2 时（除多边形或复合多边形之外的所有几何图形），使用缓冲距离的绝对值。

一般来说，对于外部环，正缓冲距离生成总是远离源几何图形中心的多边形环；负缓冲距离生成趋向该中心的多边形或复合多边形。对于多边形或复合多边形的内部环，正缓冲距离生成趋向中心的缓冲环，负缓冲距离生成远离中心的缓冲环。

缓冲过程合并重叠的多边形。当负距离超过多边形最大内部宽度的一半时，将产生空几何图形。

有关更多信息，参见第174页的『ST_Buffer』。

LocateAlong

对于具有度量单位的几何图形，可通过 LocateAlong 函数找到特定度量单位的位置。LocateAlong 以多个点的形式返回该位置。若源几何图形的维数为 0（例如，点和多个点），则需要精确匹配，并以多个点的形式返回具有匹配的度量单位值

的那些点。然而，对于维数大于 0 的源几何图形，该位置是内插值。例如，若输入的度量单位值是 5.5，线条的顶点上的度量单位分别是 3、4、5、 6 和 7，则返回内插值点，该点在具有度量单位值 5 和 6 的两个顶点之间，且距这两个顶点距离相等。

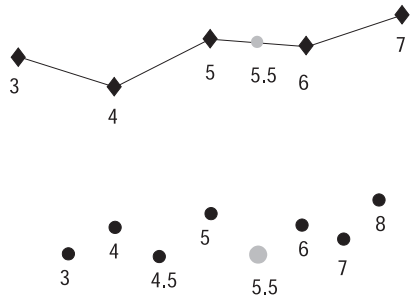


图 25. LocateAlong

有关更多信息，参见第153页的『LocateAlong』。

LocateBetween

LocateBetween 函数从具有度量单位的源几何图形返回位于两个度量单位值之间的路径或位置的集合。若源几何图形的维数为 0，则 LocateBetween 返回一个多点，该多点包含其度量单位在两个源度量单位之间的所有点。对于维数大于 0 的源几何图形，若可通过内插值得到一条路径，则 LocateBetween 返回多线条；否则 LocateBetween 返回包含点位置的多个点。当 LocateBetween 不能通过内插值得到一条路径或找到度量单位之间的位置时，则返回空点。LocateBetween 对几何图形执行包含搜索；因此，几何图形的度量单位必须大于或等于度量单位*下限*且小于或等于度量单位*上限*。

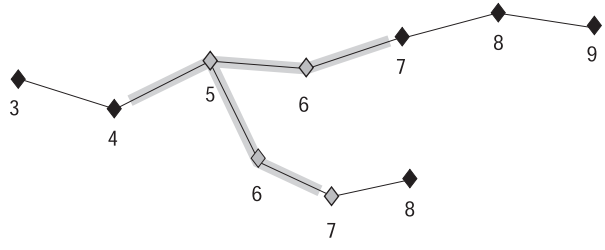


图 26. LocateBetween

有关更多信息，参见第155页的『LocateBetween』。

ST_ConvexHull

ST_ConvexHull 函数返回至少具有三个顶点（构成凸形）的任何几何图形的凸形多边形。若几何图形的顶点不构成凸形，则 ST_ConvexHull 返回空值。ST_ConvexHull 通常是棋盘形布置中用来由点集创建 TIN 网络的第一步。

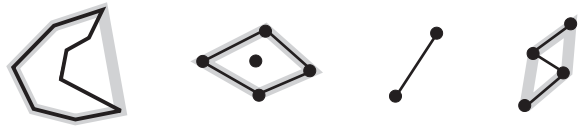


图 27. ST_ConvexHull

有关更多信息，参见第178页的『ST_ConvexHull』。

ST_Polygon

从线条生成多边形。有关更多信息，参见第246页的『ST_Polygon』。

转换几何图形值的格式的函数

DB2 Spatial Extender 支持三种 GIS 数据交换格式:

- 公认文本表示
- 公认二进制表示
- ESRI 二进制形状表示

公认文本表示

DB2 Spatial Extender 具有几个从文本说明生成几何图形的函数。

ST_WKTToSQL
使用任何几何图形类型的文本表示创建几何图形。不应指定任何 Spatial 参考系标识符。 有关更多信息，参见第259页的『ST_WKTToSQL』。

ST_GeomFromText
使用任何几何图形类型的文本表示创建几何图形。必须指定一个 Spatial 参考系标识符。 有关更多信息，参见第195页的『ST_GeometryFromText』。

ST_PointFromText
使用点文本表示创建点。有关更多信息，参见第237页的『ST_PointFromText』。

ST_LineFromText

使用线条文本表示创建线条。有关更多信息，参见第219页的『ST_LineFromText』。

ST_PolyFromText

使用多边形文本表示创建多边形。有关更多信息，参见第243页的『ST_PolyFromText』。

ST_MPointFromText

使用多点表示创建多个点。有关更多信息，参见第225页的『ST_MPointFromText』。

ST_MLineFromText

使用多线条表示创建多线条。有关更多信息，参见第222页的『ST_MLineFromText』。

ST_MPolyFromText

使用复合多边形表示创建复合多边形。有关更多信息，参见第228页的『ST_MPolyFromText』。

文本表示是 ASCII 字符串。它允许几何图形以 ASCII 文本形式进行交换。这些函数不需要定义任何特殊程序结构来映射二进制表示。因此，可在 3GL 或 4GL 程序中使用它们。

ST_AsText 函数将现存几何图形值转换为文本表示。有关更多信息，参见第171页的『ST_AsText』。

有关公认文本表示的更多信息，参见第273页的『OGC 公认文本表示』。

公认二进制表示

DB2 Spatial Extender 具有几个使用公认二进制 (WKB) 表示生成几何图形的函数。

ST_WKBToSQL

使用任何几何图形类型的公认二进制表示创建几何图形。 不应指定任何 Spatial 参考系标识符。 有关更多信息，参见第257页的『ST_WKBToSQL』。

ST_GeomFromWKB

使用任何几何图形类型的公认二进制表示创建几何图形。必须指定一个 Spatial 参考系标识符。 有关更多信息，参见第197页的『ST_GeomFromWKB』。

ST_PointFromWKB

使用点的公认二进制表示创建点。有关更多信息，参见第238页的『ST_PointFromWKB』。

ST_LineFromWKB

使用线条的公认二进制表示创建线条。有关更多信息，参见第220页的『ST_LineFromWKB』。

ST_PolyFromWKB

使用多边形的公认二进制表示创建多边形。有关更多信息，参见第244页的『ST_PolyFromWKB』。

ST_MPointFromWKB

使用多点的公认二进制表示创建多个点。有关更多信息，参见第226页的『ST_MPointFromWKB』。

ST_MLineFromWKB

使用多线条的公认二进制表示创建多线条。有关更多信息，参见第223页的『ST_MLineFromWKB』。

ST_MPolyFromWKB

使用复合多边形的公认二进制表示创建复合多边形。有关更多信息，参见第229页的『ST_MPolyFromWKB』。

公认二进制表示是连续的字节流。它允许在 ODBC 客户机和 SQL 数据库之间以二进制形式交换几何图形。这些几何图形函数需要定义 C 结构来映射二进制表示。因此，它们适用于 3GL 程序，而不适用于 4GL 环境。

ST_AsBinary 函数将现存几何图形值转换为公认二进制表示。有关更多信息，参见第169页的『ST_AsBinary』。

有关公认二进制表示的更多信息，参见第278页的『OGC 公认二进制 (WKB) 表示』。

ESRI 形状表示

DB2 Spatial Extender 具有几个使用 ESRI 形状表示生成几何图形的函数。ESRI 形状表示除支持文本和公认二进制表示支持的二维表示之外，还支持 Z 坐标和度量单位。

ShapeToSQL

使用任何几何图形类型的形状创建几何图形。不应指定任何 Spatial 参考系标识符。有关更多信息，参见第166页的『ShapeToSQL』。

GeometryFromShape

使用任何几何图形类型的形状创建几何图形。必须指定一个 Spatial 参考系标识符。有关更多信息，参见第145页的『GeometryFromShape』。

PointFromShape

使用点形状创建点。有关更多信息，参见第163页的『PointFromShape』。

LineFromShape

使用折线形状创建线条。有关更多信息，参见第151页的『LineFromShape』。

PolyFromShape

使用折线形状创建多边形。有关更多信息，参见第164页的『PolyFromShape』。

MPointFromShape

使用多点形状创建多个点。有关更多信息，参见第160页的『MPointFromShape』。

MLineFromShape

使用多部分折线形状创建多线条。有关更多信息，参见第158页的『MLine FromShape』。

MPolyFromShape

使用多部分多边形形状创建复合多边形。有关更多信息，参见第162页的『MPolyFromShape』。

这些函数的一般语法相同。第一个自变量是以 BLOB 数据类型形式输入的形状表示。第二个自变量是将给几何图形指定的 Spatial 参考系标识符。例如，GeometryFromShape 函数的语法如下:

```
GeometryFromShape(shapegeometry, SRID)
```

要映射二进制表示，这些形状函数需要定义 C 结构。因此，它们适用于 3GL 程序，而不适用于 4GL 环境。

AsBinaryShape 函数将几何图形值转换为 ESRI 形状表示。有关更多信息，参见第144页的『AsBinaryShape』。

有关形状表示的详细说明，参见第281页的『ESRI 形状表示』。

第14章 用于 SQL 查询的 Spatial 函数

本章列示查询 Spatial 数据时可调用的函数。在显示语法、返回类型和代码示例的节中描述每个函数。本章中的某些示例包括一个 CREATE TABLE 语句，在该语句中将一个或多个列定义为 Spatial 列。

以下考虑事项适用于 Spatial 函数:

- 本章中的示例用库名 db2gse 进行限定。代替用 db2gse 显式限定每个 Spatial 函数和类型，可以设置函数路径来包括 db2gse。
- 在可以将数据插入某个 Spatial 列之前:
  - 可能需要增大 udf_mem_sz 参数。建议的初始设置为 2048。若 2048 还不够，则以 256 为增量来增大 udf_mem_sz 参数。
  - 必须将该 Spatial 列注册为层。有关将 Spatial 列注册为层的更多信息，参见第29页的『第4章 定义 Spatial 列，将它们注册为层并启用地理编码器自动维护它们』。

AsBinaryShape

AsBinaryShape takes a geometry object and returns a BLOB.

语法

db2gse.AsBinaryShape(g db2gse.ST_Geometry)

返回类型

BLOB(1m)

示例

以下代码段说明 AsBinaryShape 函数如何将 SENSITIVE_AREAS 表的区域多边形转换为形状多边形。这些形状多边形被传递给应用程序的 draw_polygon 函数用于显示。

```
/* Create the SQL expression. */
strcpy(sqlstmt, "select db2gse.AsBinaryShape (zone) from SENSITIVE_AREAS
where db2gse.EnvelopesIntersect(zone, db2gse.PolyFromShape(cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");

/* Prepare the SQL statement. */
SQLPrepare(hstmt, (UCHAR *)sqlstmt, SQL_NTS);

/* Set the pcbvalue1 length of the shape. */
pcbvalue1 = blob_len;

/* Bind the shape parameter */
SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_BLOB, blob_len,
0, shape, blob_len, &pcbvalue1);

/* Execute the query */
rc = SQLExecute (hstmt);

/* Assign the results of the query (the Zone polygons) to the
   fetched_binary variable. */
SQLBindCol (hstmt, 1, SQL_C_Binary, fetched_binary, 100000, &ind_blob);

/* Fetch each polygon within the display window and display it. */

while(SQL_SUCCESS == (rc = SQLFetch(hstmt)))
  draw_polygon(fetched_binary);
```

GeometryFromShape

GeometryFromShape 接受形状和 Spatial 参考系标识以返回几何图形对象。

语法

db2gse.GeometryFromShape(ShapeGeometry Blob(1M), cr db2gse.coordref)

返回类型

db2gse.ST_Geometry

示例

以下 C 代码段包含 ODBC 函数，这些函数嵌有将数据插入 LOTS 表的 DB2 Spatial Extender SQL 函数。

已创建了 LOTS 表，该表具有两列: LOT_ID 列和 LOT 多边形列，前者唯一标识每个地块；后者包含每个地块的几何图形。

```
CREATE TABLE LOTS ( lot_id   integer,
                    lot      db2gse.ST_MultiPolygon);
```

GeometryFromShape 函数将形状转换为 DB2 Spatial Extender 几何图形。将整个 INSERT 语句复制到 shp_sql。 该 INSERT 语句包含参数标记以动态接受 LOT_ID 数据和 LOT 数据。

```
/* Create the SQL insert statement to populate the lot id and the
   lot multipolygon. The question marks are parameter markers that
   indicate the lot_id and lot values that will be retrieved at
   runtime. */
strcpy (shp_sql,"insert into LOTS (lot_id, lot) values (?, db2gse.GeometryFromShape
(cast(? as blob(1m)), db2gse.coordref()..srid(0)))");

/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);

/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)shp_sql, SQL_NTS);

/* Bind the integer key value to the first parameter. */
pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_SLONG,
     SQL_INTEGER, 0, 0, &lot_id, 0, &pcbvalue1);

/* Bind the shape to the second parameter. */
pcbvalue2 = blob_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
```

```
        SQL_BLOB, blob_len, 0, shape_blob, blob_len, &pcbvalue2);

    /* Execute the insert statement. */
    rc = SQLExecute (hstmt);
```

EnvelopesIntersect

若两个几何图形的包络相交， EnvelopesIntersect 返回 1 (TRUE)；否则，它返回 0 (FALSE)。

语法

db2gse.EnvelopesIntersect(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

返回类型

整数

示例

get_window 函数从应用程序检索显示窗口的坐标。窗口参数实际上是一种多边形形状结构，它包含一列代表显示多边形的坐标。 PolyFromShape 函数将显示窗口形状转换为 DB2 Spatial Extender 多边形， EnvelopesIntersect 函数将该多边形用作它的相交包络。返回与显示窗口的内部或边界相交的所有 SENSITIVE_AREAS 区域多边形。将每个多边形从结果集取出并传送给 draw_polygon 函数。

```
/* Get the display window coordinates as a polygon shape.
get_window(&window)

/* Create the SQL expression. The db2gse.EnvelopesIntersect function
  will be used to limit the result set to only those zone polygons
  that intersect the envelope of the display window. */
strcpy(sqlstmt, "select db2gse.AsBinaryShape(zone) from SENSITIVE_AREAS where
db2gse.EnvelopesIntersect (zone, db2gse.PolyFromShape(cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");

/* Set blob_len to the byte length of a 5 point shape polygon. */
blob_len = 128;

/* Prepare the SQL statement. */
SQLPrepare(hstmt, (UCHAR *)sqlstmt, SQL_NTS);

/* Set the pcbvalue1 to the window shape */
pcbvalue1 = blob_len;

/* Bind the shape parameter */
SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_BLOB, blob_len,
0, window, blob_len, &pcbvalue1);

/* Execute the query */
rc = SQLExecute (hstmt);

/* Assign the results of the query, (the Zone polygons) to the
   fetched_binary variable. */
SQLBindCol (hstmt, 1, SQL_C_Binary, fetched_binary, 100000, &ind_blob);
```

```
/* Fetch each polygon within the display window and display it. */
while(SQL_SUCCESS == (rc = SQLFetch(hstmt))
  draw_polygon(fetched_binary);
```

Is3d

Is3d 接受几何图形对象，若该对象具有三维坐标，则返回 1 (TRUE)；否则，它返回 0 (FALSE)。

语法

db2gse.Is3d(g db2gse.ST_Geometry)

返回类型

整数

示例

下列 CREATE TABLE 语句创建 THREED_TEST 表，该表具有两列：整数类型的 GID 列和 G1 几何图形列。

```
CREATE TABLE THREED_TEST (gid smallint, g1 db2gse.ST_Geometry)
```

下列 INSERT 语句将两个点插入 THREED_TEST 表。第一点不包含 Z 坐标，而第二点包含。

```
INSERT INTO THREED_TEST
VALUES(1, db2gse.ST_PointFromText('point (10 10)', db2gse.coordref()..srid(0)))

INSERT INTO THREED_TEST
VALUES (2, db2gse.ST_PointFromText('point z (10.92 10.12 5)',
db2gse.coordref()..srid(0)))
```

下列 SELECT 语句列示 GID 列的内容以及 Is3d 函数的结果。该函数为没有 Z 坐标的第一行返回 0，为具有 Z 坐标的第二行返回 1。

```
SELECT gid, db2gse.Is3d (g1) "Is it 3d?" FROM THREED_TEST
```

返回下列结果集。

```
gid      Is it 3d?
------   ----------
     1           0
     2           1
```

IsMeasured

IsMeasured 接受几何图形对象，若该对象具有度量单位，则返回 1 (TRUE)；否则，它返回 0 (FALSE)。

语法

db2gse.IsMeasured(g db2gse.ST_Geometry)

返回类型

整数

示例

下列 CREATE TABLE 语句创建 MEASURE_TEST 表，该表具有两列。 GID 列唯一地标识行，G1 列存储点几何图形。

```
CREATE TABLE MEASURE_TEST (gid smallint, g1 db2gse.ST_Geometry)
```

下列 INSERT 语句将两个记录插入 MEASURE_TEST 表。第一个记录存储没有度量单位的点。第二个记录的点具有度量单位。

```
INSERT INTO MEASURE_TEST
VALUES(1, db2gse.ST_PointFromText('point (10 10)', db2gse.coordref()..srid(0)))

INSERT INTO MEASURE_TEST
VALUES (2, db2gse.ST_PointFromText('point m (10.92 10.12 5)', db2gse.coordref()..srid(0)))
```

下列 SELECT 语句和对应的结果集显示 GID 列和 IsMeasured 函数的结果。IsMeasured 函数对第一行返回 0，因为该点没有度量单位。它对第二行返回 1，因为该点具有度量单位。

```
SELECT gid, db2gse.IsMeasured (g1) "Has measures?" FROM MEASURE_TEST
gid      Has measures
------   ----------
     1            0
     2            1
```

LineFromShape

LineFromShape 接受类型为点的形状和 Spatial 参考系标识并返回线条。

语法

db2gse.Line FromShape(ShapeLineString Blob(1M), cr db2gse.coordref)

返回类型

db2gse.ST_LineString

示例

下列代码段用每个排水管道的唯一 ID、大小类别和几何图形填充 SEWERLINES 表。

该 CREATE TABLE 语句创建 SEWERLINES 表，该表具有三列。第一列 SEWER_ID 唯一地标识每条排水管道。第二列 CLASS 的类型为整数，它标识排水管道的类型，排水管道的类型通常与管道的能力相关。 第三列 SEWER 为线条类型，存储排水管道的几何图形。

```
CREATE TABLE SEWERLINES (sewer_id   integer, class  integer,
                         sewer db2gse.ST_LineString);

/* Create the SQL insert statement to populate the sewer_id, size class and
   the sewer linestring. The question marks are parameter markers that
   indicate the sewer_id, class and sewer geometry values that will be
   retrieved at runtime. */
strcpy (shp_sql,"insert into sewerlines (sewer_id,class,sewer)
values (?,?, db2gse.Line FromShape (cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");

/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);

/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)shp_sql, SQL_NTS);

/* Bind the integer key value to the first parameter. */
pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_SLONG,
   SQL_INTEGER, 0, 0, &sewer_id, 0, &pcbvalue1);

/* Bind the integer class value to the second parameter. */
pcbvalue2 = 0;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_SLONG,
   SQL_INTEGER, 0, 0, &sewer_class, 0, &pcbvalue2);;

/* Bind the shape to the third parameter. */
pcbvalue3 = blob_len;
```

```
rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_BINARY,
     SQL_BLOB, blob_len, 0, sewer_shape, blob_len, &pcbvalue3);

/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

LocateAlong

LocateAlong 接受几何图形对象和度量单位，以多个点的形式返回在度量单位中找到的一组点。

语法

db2gse.LocateAlong(g db2gse.ST_Geometry, adistance Double)

返回类型

db2gse.ST_Geometry

示例

下列 CREATE TABLE 语句创建 LOCATEALONG_TEST 表。LOCATEALONG_TEST 具有两列: GID 列和 G1 几何图形列，前者唯一地标识每行，后者存储样本几何图形。

```
CREATE TABLE LOCATEALONG_TEST (gid integer, g1 db2gse.ST_Geometry)
```

下列 INSERT 语句插入两行。第一行为多线条；第二行为多个点。

```
INSERT INTO db2gse.LOCATEALONG_TEST VALUES(
1, db2gse.ST_MLineFromText('multilinestring m ((10.29 19.23 5,23.82 20.29 6,30.19 18.47
                                    7,45.98 20.74 8),
                                  (23.82 20.29 6,30.98 23.98 7,42.92 25.98 8))',
                                db2gse.coordref()..srid(0)))

INSERT INTO db2gse.LocateAlong_TEST VALUES(
2, db2gse.ST_MPointFromText('multipoint m (10.29 19.23 5,23.82 20.29 6,30.19 18.47 7,
45.98 20.74 8,23.82 20.29 6,30.98 23.98 7,42.92 25.98)',
                 db2gse.coordref()..srid(0)))
```

在下列 SELECT 语句和相应的结果集中，指示 LocateAlong 函数找出其度量单位为 6.5 的点。第一行返回包含两个点的多个点。而第二行则返回空点。对于线性地形（维数大于 0 的几何图形），LocateAlong 可内插该点；然而对于多个点，目标度量单位必须准确匹配。

```
SELECT gid, CAST(db2gse.ST_AsText(db2gse.LocateAlong (g1,6.5)) AS
varchar(96)) "Geometry"
FROM LOCATEALONG_TEST

GID         Geometry
----------- ----------------------------------------------------------------

          1 MULTIPOINT M ( 27.01000000 19.38000000 6.50000000, 27.40000000
22.14000000 6.50000000)
          2 POINT EMPTY

  2 record(s) selected.
```

在下列 SELECT 语句和相应的结果集中，LocateAlong 函数对两行返回多个点。目标度量单位 7 与多线条和多个点源数据中的度量单位都匹配。

```
SELECT gid,CAST(db2gse.ST_AsText(db2gse.LocateAlong (g1,7)) AS varchar(96)) "Geometry"
FROM LOCATEALONG_TEST

GID         Geometry

----------- ------------------------------------------------------------
          1 MULTIPOINT M ( 30.19000000 18.47000000 7.00000000, 30.98000000
 23.98000000 7.00000000)
          2 MULTIPOINT M ( 30.19000000 18.47000000 7.00000000, 30.98000000
 23.98000000 7.00000000)

  2 record(s) selected.
```

LocateBetween

LocateBetween 接受几何图形对象和两个测量位置，并返回表示两个测量位置之间的一组断开路径的几何图形。

语法

db2gse.LocateBetween(g db2gse.ST_Geometry, adistance Double, anotherdistance Double)

返回类型

db2gse.ST_Geometry

示例

下列 CREATE TABLE 语句创建 LOCATEBETWEEN_TEST 表。LOCATEBETWEEN_TEST 具有两列: GID 列和 G1 多线条列，前者唯一地标识每行，后者存储样本几何图形。

```
CREATE TABLE LOCATEBETWEEN_TEST (gid integer, g1 db2gse.ST_Geometry)
```

下列 INSERT 语句将两行插入 LOCATEBETWEEN_TEST 表。第一行为多线条，第二行为多个点。

```
INSERT INTO db2gse.LOCATEBETWEEN_TEST
VALUES(1,db2gse.ST_MLineFromText('multilinestring m ((10.29 19.23 5,23.82 20.29 6,
                                          30.19 18.47 7,45.98 20.74 8),
                                        (23.82 20.29 6,30.98 23.98 7,
                                    42.92 25.98 8))',
                        db2gse.coordref()..srid(0)))

INSERT INTO db2gse.LOCATEBETWEEN_TEST
VALUES(2, db2gse.ST_MPointFromText('multipoint m (10.29 19.23 5,23.82 20.29 6,
30.19 18.47 7,45.98 20.74 8,23.82 20.29 6,
30.98 23.98 7,42.92 25.98 8)',
                          db2gse.coordref()..srid(0)))
```

下列 SELECT 语句和相应结果集显示 LocateBetween 函数如何定位位于度量单位 6.5 和 7.5 （含二者）之间的度量单位。第一行返回包含几个线条的多线条。第二行返回多个点，因为源数据为多个点。当源数据的维数为 0（点或多个点）时，需要准确的匹配。

```
SELECT gid, CAST(db2gse.ST_AsText(db2gse.LocateBetween (g1,6.5,7.5))
       AS varchar(96)) "Geometry"
FROM LOCATEBETWEEN_TEST

GID         Geometry
----------- ------------------------------------------------------------------
          1 MULTILINESTRING M ( 27.01000000 19.38000000 6.50000000, 31.19000000
18.47000000 7.00000000,38.09000000 19.61000000 7.50000000),(27.40000000 22.1400
0000 6.50000000, 30.98000000 23.98000000 7.00000000,36.95000000 24.98000000 7.5
```

```
0000000)
          2 MULTIPOINT M ( 30.19000000 18.47000000 7.00000000, 30.98000000 23.9
8000000 7.00000000)

  2 record(s) selected.
```

M

M 接受点并返回它的度量单位。

语法

db2gse.M(p db2gse.ST_Point)

返回类型

双精度

示例

下列 CREATE TABLE 语句创建 M_TEST 表。M_TEST 具有两列: GID 整数列和 PT1 点列，前者唯一地标识每行，后者存储样本几何图形。

```
CREATE TABLE M_TEST (gid integer, pt1 db2gse.ST_Point)
```

下列 INSERT 语句插入包含具有度量单位的点的行和包含没有度量单位的点的行。

```
INSERT INTO db2gse.M_TEST
VALUES(1, db2gse.ST_PointFromText('point (10.02 20.01)', db2gse.coordref()..srid(0)))

INSERT INTO db2gse.M_TEST
VALUES(2, db2gse.ST_PointFromText('point  zm(10.02 20.01 5.0 7.0)',
db2gse.coordref()..srid(0)))
```

在下列 SELECT 语句和相应的结果集中，M 函数列示点的度量单位值。因为第一点没有度量单位，所以 M 函数返回空值。

```
SELECT gid, db2gse.M (pt1) "The measure" FROM M_TEST

GID         The measure
----------- ------------------------
          1                         -
          2   +7.00000000000000E+000

  2 record(s) selected.
```

MLine FromShape

MLineFromShape 接受类型为多线条的形状和 Spatial 参考系标识并返回多线条。

语法

db2gse.MLineFromShape(ShapeMultiLineString Blob(1M), cr db2gse.coordref)

返回类型

db2gse.ST_MultiLineString

示例

下列代码段用唯一的 ID、名称和水多线条填充 WATERWAYS 表。

创建具有 ID 列和 NAME 列的 WATERWAYS 表，这些列标识存储在该表中的每个溪流和江河系统。WATER 列为多线条，因为江河和溪流系统通常是几个线条的集合。

```
CREATE TABLE WATERWAYS (id        integer,
                        name      varchar(128),
                       water      db2gse.ST_MultiLineString);

/* Create the SQL insert statement to populate the id, name and
   multilinestring. The question marks are parameter markers that
   indicate the id, name and water values that will be retrieved at
   runtime. */
strcpy (shp_sql,"insert into WATERWAYS (id,name,water)
values (?,?, db2gse.MLineFromShape (cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");

/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);

/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)shp_sql, SQL_NTS);

/* Bind the integer id value to the first parameter. */

pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_SLONG,
   SQL_INTEGER, 0, 0, &id, 0, &pcbvalue1);
/* Bind the varchar name value to the second parameter. */

pcbvalue2 = name_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
     SQL_CHAR, name_len, 0, &name, name_len, &pcbvalue2);

/* Bind the shape to the third parameter. */
pcbvalue3 = blob_len;
rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_BINARY,
```

```
     SQL_BLOB, blob_len, 0, water_shape, blob_len, &pcbvalue3);

/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

MPointFromShape

MPointFromShape 接受类型为多个点的形状和 Spatial 参考系标识以返回多个点。

语法

db2gse.MPointFromShape(ShapeMultiPoint (1M), srs db2gse.coordref)

返回类型

db2gse.ST_MultiPoint

示例

此代码段填充生物学家的 SPECIES_SITINGS 表。

创建具有三列的 SPECIES_SITINGS 表。species 和 genus 列唯一地标识每行，而 sitings 多个点则存储物种所在的位置。

```
CREATE TABLE SPECIES_SITINGS (species  varchar(32),
                              genus varchar(32),
                              sitings  db2gse.ST_MultiPoint);

/* Create the SQL insert statement to populate the species, genus and
   sitings. The question marks are parameter markers that indicate the
   name and water values that will be retrieved at runtime. */
strcpy (shp_sql,"insert into SPECIES_SITINGS (species,genus,sitings)
values (?,?, db2gse.MPointFromShape (cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");

/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);

/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)shp_sql, SQL_NTS);

/* Bind the varchar species value to the first parameter. */
pcbvalue1 = species_len;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
     SQL_CHAR, species_len, 0, species, species_len, &pcbvalue1);

/* Bind the varchar genus value to the second parameter. */
pcbvalue2 = genus_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
   SQL_CHAR, genus_len, 0, name, genus_len, &pcbvalue2);

/* Bind the shape to the third parameter. */
pcbvalue3 = blob_len;
rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_BINARY,
```

```
     SQL_BLOB, sitings_len, 0, sitings_shape, sitings_len, &pcbvalue3);

/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

MPolyFromShape

MPolyFromShape 接受类型为复合多边形的形状和 Spatial 参考系标识以返回复合多边形。

语法

db2gse.MPolyFromShape(ShapeMultiPolygon Blob(1m), srs db2gse.coordref)

返回类型

db2gse.ST_MultiPolygon

示例

此代码段填充 LOTS 表。

LOTS 表存储唯一地标识每个地块的 lot_id 和包含地块线几何图形的地块复合多边形。

```
CREATE TABLE LOTS (  lot_id integer, lot db2gse.ST_MultiPolygon );

/* Create the SQL insert statement to populate the lot_id and lot. The
   question marks are parameter markers that indicate the lot_id and lot
   values that will be retrieved at runtime. */
strcpy (shp_sql,"insert into LOTS (lot_id,lot)
values (?, db2gse.MPolyFromShape (cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");

/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);

/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)shp_sql, SQL_NTS);

/* Bind the lot_id integer value to the first parameter. */
pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_INTEGER,
     SQL_INTEGER, 0, 0, &lot_id, 0, &pcbvalue1);

/* Bind the lot shape to the second parameter. */
pcbvalue2 = lot_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
     SQL_BLOB, lot_len, 0, lot_shape, lot_len, &pcbvalue2);

/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

PointFromShape

PointFromShape 接受类型为点的形状和 Spatial 参考系标识以返回点。

语法

db2gse.PointFromShape(db2gse.ShapePoint blob(1M), srs db2gse.coordref)

返回类型

db2gse.ST_Point

示例

该程序段填充 HAZARDOUS_SITES 表。

危险地点存储在用以下 CREATE TABLE 语句创建的 HAZARDOUS_SITES 表中。定义为点的 location 列存储每个危险地点的地理中心的位置。

```
CREATE TABLE HAZARDOUS_SITES (site_id   integer,
                              name      varchar(128),
                              location  db2gse.ST_Point);

/* Create the SQL insert statement to populate the site_id, name and
   location. The question marks are parameter markers that indicate the
   site_id, name and location values that will be retrieved at runtime. */
strcpy (shp_sql,"insert into HAZARDOUS_SITES (site_id, name, location)
values (?,?, db2gse.PointFromShape (cast(? as blob(1m)), db2gse.coordref()..srid(0)))");

/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);

/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)shp_sql, SQL_NTS);

/* Bind the site_id integer value to the first parameter. */
pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_INTEGER,
     SQL_INTEGER, 0, 0, &site_id, 0, &pcbvalue1);

/* Bind the name varchar value to the second parameter. */
pcbvalue2 = name_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
   SQL_CHAR, 0, 0, name, 0, &pcbvalue2);

/* Bind the location shape to the third parameter. */
pcbvalue3 = location_len;
rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_BINARY,
     SQL_BLOB, location_len, 0, location_shape, location_len, &pcbvalue3);

/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

PolyFromShape

PolyFromShape 接受类型为多边形的形状和 Spatial 参考系标识以返回多边形。

语法

db2gse.PolyFromShape (ShapePolygon Blob(1M), srs db2gse.coordref)

返回类型

db2gse.ST_Polygon

示例

该程序段填充 SENSITIVE_AREAS 表。问号表示将在运行期检索的 ID、名称、大小、类型和区域值的参数标记。

除了包含 zone 列（用于存储公共设施的多边形几何图形）之外，SENSITIVE_AREAS 表还包含描述受威胁的公共设施的几列。

```
CREATE TABLE SENSITIVE_AREAS (id        integer,
                              name      varchar(128),
                              size      float,
                              type      varchar(10),
                              zone      db2gse.ST_Polygon);

/* Create the SQL insert statement to populate the id, name, size, type and
   zone. The question marks are parameter markers that indicate the
   id, name, size, type and zone values that will be retrieved at runtime. */
strcpy (shp_sql,"insert into SENSITIVE_AREAS (id, name, size, type, zone)
values (?,?,?,?, db2gse.PolyFromShape (cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");

/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);

/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)shp_sql, SQL_NTS);

/* Bind the id integer value to the first parameter. */
pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_INTEGER,
     SQL_INTEGER, 0, 0, &site_id, 0, &pcbvalue1);
/* Bind the name varchar value to the second parameter. */
pcbvalue2 = name_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
   SQL_CHAR, 0, 0, name, 0, &pcbvalue2);

/* Bind the size float to the third parameter. */
pcbvalue3 = 0;
rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_FLOAT,
     SQL_REAL, 0, 0, &size, 0, &pcbvalue3);
```

```
/* Bind the type varchar to the fourth parameter. */
pcbvalue4 = type_len;
rc = SQLBindParameter (hstmt, 4, SQL_PARAM_INPUT, SQL_C_CHAR,
     SQL_VARCHAR, type_len, 0, type, type_len, &pcbvalue4);

/* Bind the zone polygon to the fifth parameter. */
pcbvalue5 = zone_len;
rc = SQLBindParameter (hstmt, 5, SQL_PARAM_INPUT, SQL_C_BINARY,
     SQL_BLOB, zone_len, 0, zone_shp, zone_len, &pcbvalue5);

/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

ShapeToSQL

ShapeToSQL 构造一个 db2gse.ST_Geometry 值，给出它的公认二进制表示。自动使用值为 0 的 SRID。

语法

db2gse.ST_ShapeToSQL(ShapeGeometry blob(1M))

返回类型

db2gse.ST_Geometry

示例

以下 C 代码段包含 ODBC 函数，这些函数嵌有将数据插入 LOTS 表的 DB2 Spatial Extender SQL 函数。 已创建了 LOTS 表，该表具有两列: lot_id 列和地块复合多边形列，前者唯一标识每个地块，后者包含每个地块的几何图形。

```
CREATE TABLE lots (lot_id  integer,
                   lot     db2gse.ST_MultiPolygon);
```

ShapeToSQL 函数将形状转换为 DB2 Spatial Extender 几何图形。将整个 INSERT 语句复制到 shp_sql。该 INSERT 语句包含一些参数标记，以动态接受 LOT_id 和地块数据。

```
/* Create the SQL insert statement to populate the lot id and the
   lot multipolygon. The question marks are parameter markers that
   indicate the lot_id and lot values that will be retrieved at
   run time. */

strcpy (shp_sql,"insert into lots (lot_id, lot) values(?,
db2gse.ShapeToSQL(cast(? as blob(1m))))");

/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */

rc = SQLAllocStmt (handle, &hstmt);

/* Prepare the SQL statement for execution. */

rc = SQLPrepare (hstmt, (unsigned char *)shp_sql, SQL_NTS);

/* Bind the integer key value to the first parameter. */

pcbvalue1 = 0;

rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_SLONG,
     SQL_INTEGER, 0, 0, &lot_id, 0, &pcbvalue1);

/* Bind the shape to the second parameter. */
```

```
pcbvalue2 = blob_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
     SQL_BLOB, blob_len, 0, shape_blob, blob_len, &pcbvalue2);

/* Execute the insert statement. */

rc = SQLExecute (hstmt);
```

ST_Area

ST_Area 接受多边形或复合多边形，并返回它的面积。

语法

db2gse.ST_Area(s db2gse.ST_Surface)

返回类型

双精度

示例

城市工程师需要建筑物面积的列表。为获取该列表， GIS 技术人员选择建筑物 ID 和每个建筑物的占地面积。

建筑物占地形状存储在用以下 CREATE TABLE 语句创建的 BUILDINGFOOTPRINTS 表中:

```
CREATE TABLE BUILDINGFOOTPRINTS (   building_id integer,
                                    lot_id      integer,
                                    footprint   db2gse.ST_MultiPolygon);
```

为满足城市工程师的请求，技术人员使用以下 SELECT 语句从 BUILDINGFOOTPRINTS 表中选择唯一的关键字 building_id 和每个建筑物的占地面积:

```
SELECT building_id, db2gse.ST_Area (footprint) "Area"
FROM BUILDINGFOOTPRINTS;
```

SELECT 语句返回以下结果集:

```
building_id   Area
------------  ------------------------
         506    +1.40768000000000E+003
        1208    +2.55759000000000E+003
         543    +1.80786000000000E+003
         178    +2.08671000000000E+003
         .
         .
         .
```

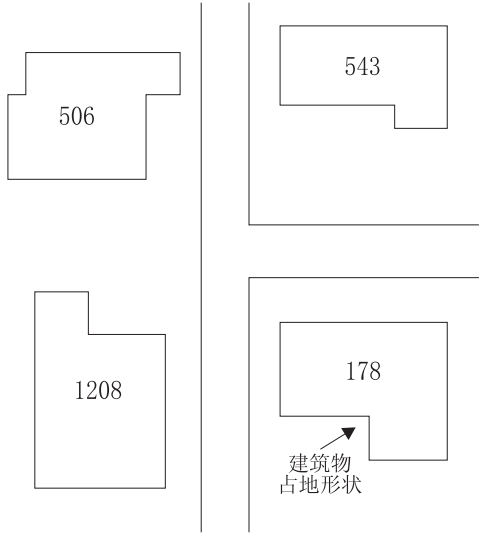


图 28. 使用面积来查找建筑物占地形状. 标有建筑物 ID 号的四个建筑物占地形状沿它们相邻的街道边显示。

ST_AsBinary

ST_AsBinary 接受几何图形对象并返回它的公认二进制表示。

语法

db2gse.ST_AsBinary(g db2gse.ST_Geometry)

返回类型

BLOB(1m)

示例

以下代码段说明 ST_AsBinary 函数如何将 BUILDINGFOOTPRINTS 表的占地形状复合多边形转换为 WKB 复合多边形。这些复合多边形被传递给应用程序的 draw_polygon 函数进行显示。

```
/* Create the SQL expression. */
strcpy(sqlstmt, "select db2gse.ST_AsBinary (footprint) from BUILDINGFOOTPRINTS
where db2gse.EnvelopesIntersect(footprint, db2gse.ST_PolyFromWKB(cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");

/* Prepare the SQL statement. */
SQLPrepare(hstmt, (UCHAR *)sqlstmt, SQL_NTS);

/* Set the pcbvalue1 length of the shape. */
pcbvalue1 = blob_len;
```

```
/* Bind the shape parameter */
SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_BLOB, blob_len,
0, shape, blob_len, &pcbvalue1);

/* Execute the query */
rc = SQLExecute (hstmt);

/* Assign the results of the query (the Zone polygons) to the
  fetched_binary variable.
 */
SQLBindCol (hstmt, 1, SQL_C_Binary, fetched_binary, 100000, &ind_blob);

/* Fetch each polygon within the display window and display it. */
while(SQL_SUCCESS == (rc = SQLFetch(hstmt)))
  draw_polygon(fetched_binary);
```

ST_AsText

db2gse.ST_AsText takes a geometry object and returns its well-known text representation.

语法

db2gse.ST_AsText(g db2gse.ST_Geometry)

返回类型

Varchar(4000)

示例

在以下方案中，db2gse.ST_AsText 函数将 HAZARDOUS_SITES 位置点转换成它的文本说明:

```
CREATE TABLE HAZARDOUS_SITES (site_id   integer,
                              name      varchar(40),
                              location  db2gse.ST_Point);
INSERT INTO HAZARDOUS_SITES
VALUES (102,
        'W. H. Kleenare Chemical Repository',
        db2gse.ST_PointFromText('point (1020.12 324.02)', db2gse.coordref()..srid(0)));

SELECT site_id, name, cast(db2gse.ST_AsText(location) as varchar(40)) "Location"
FROM HAZARDOUS_SITES;
```

SELECT 语句返回以下结果集:

```
SITE_ID Name                                Location
------- ----------------------------------- ----------------------------------------
    102 W. H. Kleenare Chemical Repository  POINT (1020.00000000 324.00000000)
```

ST_Boundary

ST_Boundary 接受几何图形对象并返回它的组合边界作为几何图形对象。

语法

db2gse.ST_Boundary(g db2gse.ST_Geometry)

返回类型

db2gse.ST_Geometry

示例

在以下代码段中，创建名为 BOUNDARY_TEST 的表。 BOUNDARY_TEST 具有两列: GEOTYPE 和 G1，前者定义为变量字符；后者定义为超类几何图形。跟随的 INSERT 语句插入每个子类几何图形。 ST_Boundary 函数检索 G1 几何图形列中存储的每个子类的边界。注意: 产生的几何图形的维数总是比输入的几何图形少一。点和多个点总是产生空几何图形的边界，维数为 1。线条和多线条返回多个点边界，维数为 0。多边形或复合多边形总是返回多线条边界，维数为 1。

```
CREATE TABLE BOUNDARY_TEST (GEOTYPE varchar(20), G1 db2gse.ST_Geometry)

INSERT INTO BOUNDARY_TEST
VALUES('Point',
       db2gse.ST_PointFromText('point (10.02 20.01)',
                     db2gse.coordref()..srid(0)))

INSERT INTO BOUNDARY_TEST
VALUES ('Linestring',
       db2gse.ST_LineFromText('linestring  (10.02 20.01,10.32 23.98,11.92 25.64)',
                    db2gse.coordref()..srid(0)))

INSERT INTO BOUNDARY_TEST
VALUES('Polygon',
       db2gse.ST_PolyFromText('polygon ((10.02 20.01,11.92 35.64,25.02 34.15,
                                  19.15 33.94, 10.02 20.01))',
                    db2gse.coordref()..srid(0)))

INSERT INTO BOUNDARY_TEST
VALUES('Multipoint',
       db2gse.ST_MPointFromText('multipoint (10.02 20.01,10.32 23.98,11.92 25.64)',
                      db2gse.coordref()..srid(0)))

INSERT INTO BOUNDARY_TEST
VALUES('Multilinestring',
       db2gse.ST_MLineFromText('multilinestring ((10.02 20.01,10.32 23.98,11.92 25.64),
                                     (9.55 23.75,15.36 30.11))',
                     db2gse.coordref()..srid(0)))

INSERT INTO BOUNDARY_TEST
VALUES('Multipolygon',
```

```
db2gse.ST_MPolyFromText('multipolygon (((10.02 20.01,11.92 35.64,25.02 34.15,
                                         19.15 33.94,10.02 20.01)),
                                        ((51.71 21.73,73.36 27.04,71.52 32.87,
                                         52.43 31.90,51.71 21.73)))',
                    db2gse.coordref()..srid(0)))

SELECT GEOTYPE,
       CAST(db2gse.ST_AsText(db2gse.ST_Boundary (G1)) as varchar(280)) "The boundary"
FROM BOUNDARY_TEST


GEOTYPE              The boundary
-------------------- ------------------------------------------------------------
Point                POINT EMPTY
Linestring           MULTIPOINT ( 10.02000000 20.01000000, 11.92000000
 25.64000000)
Polygon              MULTILINESTRING (( 10.02000000 20.01000000, 19.15000000
 33.94000000, 25.02000000 34.15000000, 11.92000000 35.64000000, 10.02000000
 20.01000000))
Multipoint           POINT EMPTY
Multilinestring      MULTIPOINT ( 9.55000000 23.75000000, 10.02000000
 20.01000000, 11.92000000 25.64000000, 15.36000000 30.11000000)
Multipolygon         MULTILINESTRING (( 51.71000000 21.73000000, 73.36000000
 27.04000000, 71.52000000 32.87000000, 52.43000000 31.90000000, 51.71000000
 21.73000000),( 10.02000000 20.01000000, 19.15000000 33.94000000, 25.02000000
 34.15000000, 11.92000000 35.64000000, 10.02000000 20.01000000))

6 record(s) selected.
```

ST_Buffer

ST_Buffer 接受几何图形对象和距离并返回围绕源对象的几何图形对象。

语法

db2gse.ST_Buffer(g db2gse.ST_Geometry , adistance Double)

返回类型

db2gse.ST_Geometry

示例

县级主管人员需要有一张危险地点的列表，这些地点周围半径为五英里的范围内包括了敏感区域（比如学校、医院和私人疗养院）。敏感区域存储在用以下 CREATE TABLE 语句创建的表 SENSITIVE_AREAS 中。 ZONE 列定义为多边形，存储为每个敏感区域的轮廓。

```
CREATE TABLE SENSITIVE_AREAS (id        integer,
                              name      varchar(128),
                              size      float,
                              type      varchar(10),
                              zone      db2gse.ST_Polygon);
```

危险地点存储在用以下 CREATE TABLE 语句创建的 HAZARDOUS_SITES 表中。定义为点的 LOCATION 列存储每个危险地点的地理中心的位置。

```
CREATE TABLE HAZARDOUS_SITES (site_id   integer,
                              name      varchar(128),
                              location  db2gse.ST_Point);
```

db2gse.ST_Overlaps 函数将 SENSITIVE_AREAS 和 HAZARDOUS_SITES 表连接在一起。对其区域多边形覆盖 HAZARDOUS_SITES 位置点的缓冲五英里半径的所有 SENSITIVE_AREAS 行，该函数返回 1 (TRUE)。

```
SELECT sa.name "Sensitive Areas", hs.name "Hazardous Sites"
FROM SENSITIVE_AREAS sa, HAZARDOUS_SITES hs
WHERE db2gse.ST_Overlaps(sa.zone, db2gse.ST_Buffer (hs.location,(5 * 5280))) = 1;
```

在第175页的图29中，此管理区的某些敏感区域位于危险地点的五英里缓冲区内。两个五英里缓冲区都与医院相交，它们中的一个与学校相交。而私人疗养院安全地位于两个半径之外。

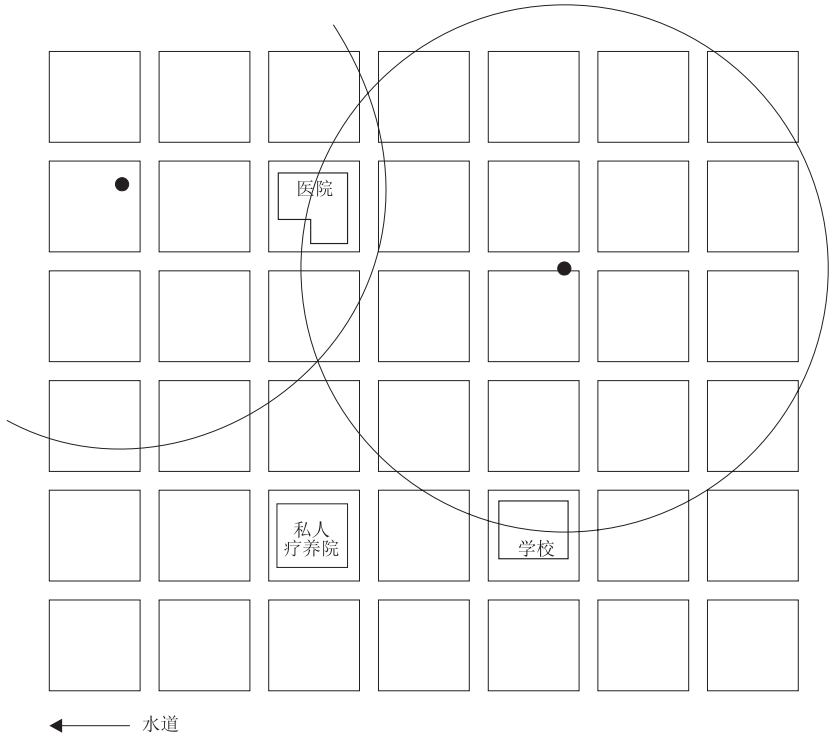


图 29. 对某个点应用五英里半径的缓冲区

ST_Centroid

ST_Centroid 接受多边形或复合多边形并以点的形式返回它的地理中心。

语法

db2gse.ST_Centroid(s db2gse.ST_Surface)
db2gse.ST_Centroid(ms db2gse.ST_MultiSurface)

返回类型

对于表面: db2gse.ST_Point

示例

城市 GIS 技术人员想要在建筑物密度图形上将建筑物占地形状的复合多边形显示为单个点。

建筑物占地形状存储在用以下 CREATE TABLE 语句创建的 BUILDINGFOOTPRINTS 表中。

```
CREATE TABLE BUILDINGFOOTPRINTS (   building_id integer,
                                    lot_id      integer,
                                    footprint   db2gse.ST_MultiPolygon);
```

ST_Centroid 函数返回每个建筑物占地形状复合多边形的质心。 AsBinaryShape 函数将质心点转换为形状（应用程序可识别的外部表示）。

```
SELECT building_id,
       CAST(db2gse.AsBinaryShape(db2gse.ST_Centroid (footprint)) as blob(1m))
"Centroid"
FROM BUILDINGFOOTPRINTS;
```

ST_Contains

ST_Contains 接受两个几何图形对象，若第一个对象完全包含第二个对象，则返回 1 (TRUE)；否则，它返回 0 (FALSE)。

语法

db2gse.ST_Contains(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

返回类型

整数

示例

在下面的示例中创建两个表。一个表包含城市的建筑物占地形状，另一个表包含该城市的地块。城市工程师想要确保所有建筑物占地形状完全在它们的地块内。

在两个表中，复合多边形数据类型存储建筑物占地形状和地块的几何图形。数据库设计者为两个地形选择了复合多边形。设计者认识到地块可能被自然地形（比如江河）分开，建筑物占地形状可能经常由几个建筑物组成。

```
CREATE TABLE BUILDINGFOOTPRINTS (   building_id integer,
                                    lot_id      integer,
                                    footprint   db2gse.ST_MultiPolygon);

CREATE TABLE LOTS (  lot_id integer, lot db2gse.ST_MultiPolygon );
```

城市工程师首先选择不完全包含在一个地块内的建筑物。

```
SELECT building_id
  FROM BUILDINGFOOTPRINTS, LOTS
WHERE db2gse.ST_Contains(lot,footprint) = 0;
```

城市工程师认识到第一次查询将返回占地形状在地块多边形之外的所有建筑物 ID 的列表。但城市规划工程师也知道此信息不能表明是否给其他建筑物指定了正确的地块 ID。第二次查询对 BUILDINGFOOTPRINTS 表的 lot_id 列执行数据完整性检查。

```
SELECT bf.building_id "building id", bf.lot_id "buildings lot_id",
       LOTS.lot_id "LOTS lot_id"
  FROM BUILDINGFOOTPRINTS bf, LOTS
WHERE db2gse.ST_Contains(lot,footprint) = 1 AND LOTS.lot_id <> bf.lot_id;
```

在第178页的图30中，标有其建筑物 ID 的建筑物占地形状位于它们的地块内。用虚线表示地块线。虽然未显示，这些虚线延伸至街道中心线并完全包围它们之内的地块和建筑物占地形状。

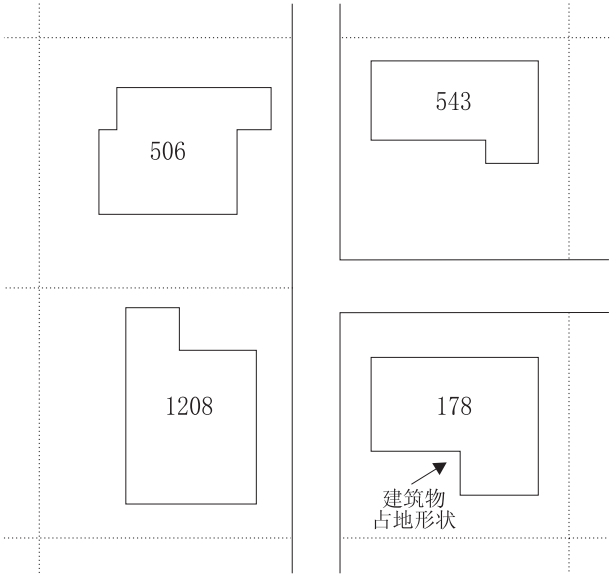


图 30. 使用 ST_Contains 确保所有建筑物包含在它们的地块内

ST_ConvexHull

ST_ConvexHull 接受几何图形对象并返回凸壳。

语法

db2gse.ST_ConvexHull(g db2gse.ST_Geometry)

返回类型

db2gse.ST_Geometry

示例

该示例创建 CONVEXHULL_TEST 表，该表具有两列: GEOTYPE 和 G1。GEOTYPE 列（varchar(20)）将存储几何图形的子类的名称，这些图形存储在定义为几何图形的 G1 中。

```
CREATE TABLE CONVEXHULL_TEST (geotype varchar(20), g1 db2gse.ST_Geometry)
```

每个 INSERT 语句将每种子类的几何图形插入 CONVEXHULL_TEST 表。

```
INSERT INTO CONVEXHULL_TEST
VALUES('Point',
       db2gse.ST_PointFromText('point (10.02 20.01)',db2gse.coordref()..srid(0)))

INSERT INTO CONVEXHULL_TEST
VALUES ('Linestring',
```

```
       db2gse.ST_LineFromText('linestring  (10.02 20.01,10.32 23.98,11.92 25.64)',
                    db2gse.coordref()..srid(0)))

INSERT INTO CONVEXHULL_TEST
VALUES('Polygon',
       db2gse.ST_PolyFromText('polygon ((10.02 20.01,11.92 35.64,25.02 34.15,
                                         19.15 33.94,10.02 20.01))',
                    db2gse.coordref()..srid(0)))

INSERT INTO CONVEXHULL_TEST
VALUES('Multipoint',
       db2gse.ST_MPointFromText('multipoint (10.02 20.01,10.32 23.98,11.92 25.64)',
                     db2gse.coordref()..srid(0)))

INSERT INTO CONVEXHULL_TEST
VALUES('Multilinestring',
       db2gse.ST_MLineFromText('multilinestring ((10.02 20.01,10.32 23.98,11.92 25.64),
                                       (9.55 23.75,15.36 30.11))',
                     db2gse.coordref()..srid(0)))

INSERT INTO CONVEXHULL_TEST
VALUES('Multipolygon',
db2gse.ST_MPolyFromText('multipolygon (((10.02 20.01,11.92 35.64,25.02 34.15,
                                         19.15 33.94,10.02 20.01)),
                                        ((51.71 21.73,73.36 27.04,71.52 32.87,
                                         52.43 31.90,51.71 21.73)))',
                     db2gse.coordref()..srid(0)))
```

以下 SELECT 语句列示 GEOTYPE 列中存储的子类名和凸壳。由 ST_AsText 函数将 ST_ConvexHull 函数生成的凸壳转换为文本。然后将它转换为 varchar(256), 因为 ST_AsText 的缺省输出为 varchar(4000)。

```
SELECT GEOTYPE, CAST(db2gse.ST_AsText(db2gse.ST_ConvexHull(G1))) as varchar(256)
"The convexhull"
FROM CONVEXHULL_TEST
```

ST_CoordDim

ST_CoordDim 返回 ST_Geometry 值的坐标维数。有关坐标维数的解释，参见第114页的『点』。

语法

db2gse.ST_CoordDim(g1 db2gse.ST_Geometry)

返回类型

整数

示例

创建具有 geotype 和 g1 列的 coorddim_test 表。 geotype 列存储 g1 几何图形列中存储的几何图形子类的名称。

```
CREATE TABLE coorddim_test (geotype varchar(20), g1 db2gse.ST_Geometry)
```

下列 INSERT 语句将样本子类插入 coorddim_test 表。

```
INSERT INTO coorddim_test VALUES(
    'Point', db2gse.ST_PointFromText('point (10.02 20.01)', db2gse.coordref()..srid(0))
)

INSERT INTO coorddim_test VALUES(
'Linestring',
       db2gse.ST_LineFromText('linestring  (10.02 20.01,10.32 23.98,11.92 25.64)',
                        db2gse.coordref()..srid(0))
)

INSERT INTO coorddim_test VALUES(
'Polygon', db2gse.ST_PolyFromText('polygon ((10.02 20.01,11.92 35.64,25.02 34.15,
19.15 33.94,10.02 20.01))', db2gse.coordref()..srid(0))
)

INSERT INTO coorddim_test VALUES(
'Multipoint', db2gse.ST_MPointFromText('multipoint (10.02 20.01,10.32 23.98,
11.92 25.64)', db2gse.coordref()..srid(0))
)

INSERT INTO coorddim_test VALUES(
'Multilinestring', db2gse.ST_MLineFromText('multilinestring ((10.02 20.01,
10.32 23.98,11.92 25.64),(9.55 23.75,15.36 30.11))', db2gse.coordref()..srid(0))
)

INSERT INTO coorddim_test VALUES(
'Multipolygon',
       MPolyFromText('multipolygon (((10.02 20.01,11.92 35.64,25.02 34.15,
19.15 33.94,10.02 20.01)),((51.71 21.73,73.36 27.04,71.52 32.87,
52.43 31.90,51.71 21.73)))', db2gse.coordref()..srid(0))
)
```

该 SELECT 语句列示 geotype 列中存储的子类名和该 geotype 的坐标维数。

```
SELECT geotype, db2gse.ST_coordDim(g1)' coordinate_dimension'
FROM coorddim_test

GEOTYPE        coordinate_dimension
-------------------- -------------------------
ST_Point                                  2
ST_Linestring                             2
ST_Polygon                  2
ST_Multipoint                             2
ST_Multilinestring                        2
ST_Multipolygon             2

6 record(s) selected.
```

ST_Crosses

ST_Crosses 接受两个几何图形对象，若它们的交集产生维数比源对象的最大维数少一的几何图形对象，则返回 1 (TRUE)。交集对象包含两个源几何图形内部的点且不等于任一个源对象。否则，它返回 0 (FALSE)。

语法

db2gse.ST_Crosses(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

返回类型

整数

示例

县政府正考虑一个新规章，规定县内所有有害废物存储设施不能在任何水道的五英里范围内。县 GIS 管理员具有江河和溪流的准确表示，它们在 WATERWAYS 表中存储为多线条。但 GIS 管理员只有每个有害废物存储设施的单个点位置。

```
CREATE TABLE WATERWAYS (id        integer,
                        name      varchar(128),
                        water     db2gse.ST_MultiLineString);

CREATE TABLE HAZARDOUS_SITES (site_id   integer,
                              name      varchar(128),
                              location  db2gse.ST_Point);
```

要确定是否需要提醒县级管理人员注意违反建议的规章的设施，GIS 管理员应给危险地点位置设置缓冲区并检查是否有任何江河或溪流穿过缓冲多边形。 ST_Crosses 谓词将缓冲的 HAZARDOUS_SITES 与 WATERWAYS 进行比较。因此，仅返回水道穿越该县建议的规定半径的那些记录。

```
SELECT ww.name "River or stream", hs.name "Hazardous site"
FROM WATERWAYS ww, HAZARDOUS_SITES hs
WHERE db2gse.ST_Crosses(db2gse.ST_Buffer(hs.location,(5 * 5280)),ww.water) = 1;
```

在第183页的图31中，有害废物地点的五英里缓冲区与流过该县的管理区的溪流网络交叉。该溪流网络已定义为多线条。因此，结果集包括与该半径交叉的所有线条段。

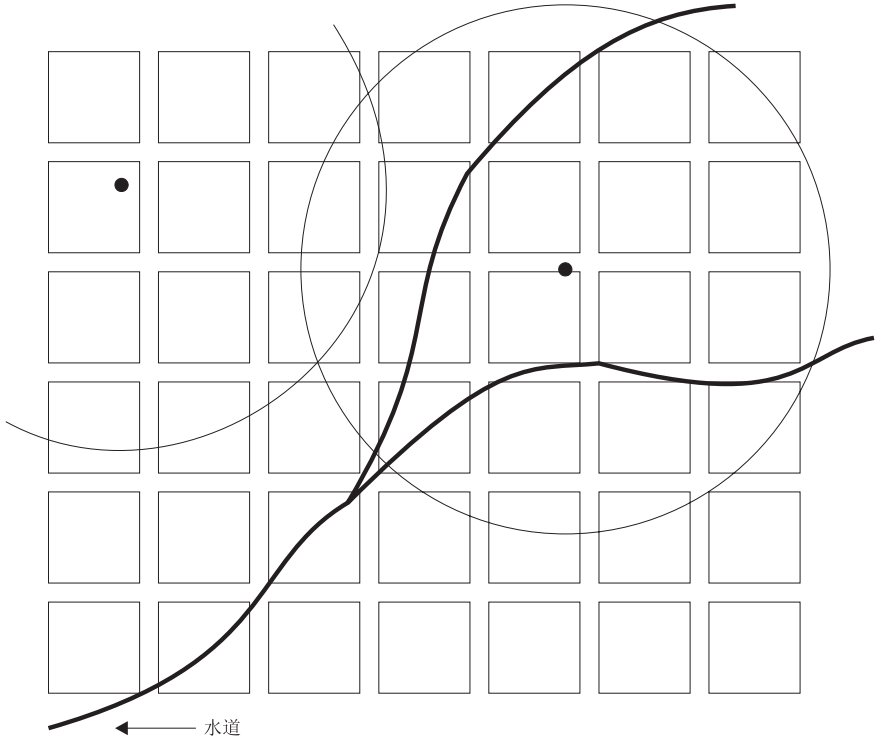


图 31. 使用 ST_Crosses 查找穿过有害废物区域的水道

ST_Difference

ST_Difference 接受两个几何图形对象，并返回是源对象之差的几何图形对象。

语法

db2gse.ST_Difference(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

返回类型

db2gse.ST_Geometry

示例

城市工程师需要知道该城市未被建筑物覆盖的地块面积的总面积。即，城市工程师想要知道在除去建筑物面积之后的地块面积的和。

```
CREATE TABLE BUILDINGFOOTPRINTS (   building_id integer,
                                    lot_id      integer,
                                    footprint   db2gse.ST_MultiPolygon);
```

```
CREATE TABLE LOTS ( lot_id   integer,
                    lot      db2gse.ST_MultiPolygon);
```

城市工程师通过 lot_id 对等地连接 BUILDINGFOOTPRINTS 和 LOTS 表。该工程师然后求出地块减去建筑物占地形状的差的面积之和。

```
SELECT SUM(db2gse.ST_Area(db2gse.ST_Difference(lot,footprint)))
  FROM BUILDINGFOOTPRINTS bf, LOTS
WHERE bf.lot_id = LOTS.lot_id;
```

ST_Dimension

ST_Dimension 接受几何图形对象并返回它的维数。

语法

db2gse.ST_Dimension(g1 db2gse.ST_Geometry)

返回类型

整数

示例

创建具有 GEOTYPE 和 G1 列的 DIMENSION_TEST 表。 GEOTYPE 列存储 G1 几何图形列中存储的几何图形子类的名称。

```
CREATE TABLE DIMENSION_TEST (geotype varchar(20), g1 db2gse.ST_Geometry)
```

下列 INSERT 语句将样本子类插入 DIMENSION_TEST 表。

```
INSERT INTO DIMENSION_TEST
VALUES('Point',
db2gse.ST_PointFromText('point (10.02 20.01)', db2gse.coordref()..srid(0)))

INSERT INTO DIMENSION_TEST
VALUES ('Linestring',
       db2gse.ST_LineFromText('linestring  (10.02 20.01,10.32 23.98,11.92 25.64)',
                    db2gse.coordref()..srid(0)))

INSERT INTO DIMENSION_TEST
VALUES('Polygon',
       db2gse.ST_PolyFromText('polygon ((10.02 20.01,11.92 35.64,25.02 34.15,
                                         19.15 33.94,10.02 20.01))',
                    db2gse.coordref()..srid(0)))

INSERT INTO DIMENSION_TEST
VALUES('Multipoint',
       db2gse.ST_MPointFromText('multipoint (10.02 20.01,10.32 23.98,11.92 25.64)',
                     db2gse.coordref()..srid(0)))

INSERT INTO DIMENSION_TEST
VALUES('Multilinestring',
       db2gse.ST_MLineFromText('multilinestring ((10.02 20.01,10.32 23.98,11.92 25.64),
                                       (9.55 23.75,15.36 30.11))',
                     db2gse.coordref()..srid(0)))

INSERT INTO DIMENSION_TEST
VALUES('Multipolygon',
db2gse.ST_MPolyFromText('multipolygon (((10.02 20.01,11.92 35.64,25.02 34.15,
                                         19.15 33.94,10.02 20.01)),
                                        ((51.71 21.73,73.36 27.04,71.52 32.87,
                                         52.43 31.90,51.71 21.73)))',
                     db2gse.coordref)..srid(0)))
```

下列 SELECT 语句列示 GEOTYPE 列中存储的子类名以及该 geotype 的维数。

```
SELECT geotype, db2gse.ST_Dimension(g1) "The dimension"
FROM DIMENSION_TEST
```

返回下列结果集。

```
GEOTYPE              The dimension
-------------------- -------------
ST_Point                       0
ST_Linestring                  1
ST_Polygon                     2
ST_Multipoint                  0
ST_Multilinestring             1
ST_Multipolygon                2

6 record(s) selected.
```

ST_Disjoint

ST_Disjoint 接受两个几何图形，若两个几何图形的交集为空，则返回 1 (TRUE)；否则，它返回 0 (FALSE)。

语法

db2gse.ST_Disjoint(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

返回类型

整数

示例

保险公司需要估计一个镇的医院、私人疗养院和学校的保险总额。此过程的一部分包括确定有害废物地点对每个公共设施的威胁。保险公司只想考虑没有污染危险的那些公共设施。该保险公司雇用的 GIS 顾问已受命找出未在有害废物地点五英里半径内的所有公共设施。

除了包含 ZONE 列（用于存储公共设施的多边形几何图形）之外，SENSITIVE_AREAS 表还包含描述受威胁的公共设施的几列。

```
CREATE TABLE SENSITIVE_AREAS (id        integer,
                              name      varchar(128),
                              size      float,
                              type      varchar(10),
                              zone      db2gse.ST_Polygon);
```

HAZARDOUS_SITES 表将地点的标识存储在 SITE_ID 和 NAME 列中，而将每个地点的实际地理位置存储在 LOCATION 列中。

```
CREATE TABLE HAZARDOUS_SITES (site_id   integer,
                              name      varchar(128),
                              location  db2gse.ST_Point);
```

下列 SELECT 语句列示未在有害废物地点五英里半径内的所有敏感区域的名称。若 ST_Disjoint 函数的结果设置为等于 0 而不是 1，则 ST_Intersects 函数可在此查询中替换 ST_Disjoint 函数。这是因为 ST_Intersects 和 ST_Disjoint 返回正好相反的结果。

```
SELECT sa.name
FROM SENSITIVE_AREAS sa, HAZARDOUS_SITES hs
WHERE db2gse.ST_Disjoint(db2gse.ST_Buffer(hs.location,(5 * 5280)),sa.zone) = 1;
```

在第188页的图32中，将敏感区域地点与有害废物地点的五英里半径比较。私人疗养院是 ST_Disjoint 函数将返回 1 (TRUE) 的唯一敏感区域。无论何时两个几何图形不以任何方式相交，ST_Disjoint 函数就返回 1。

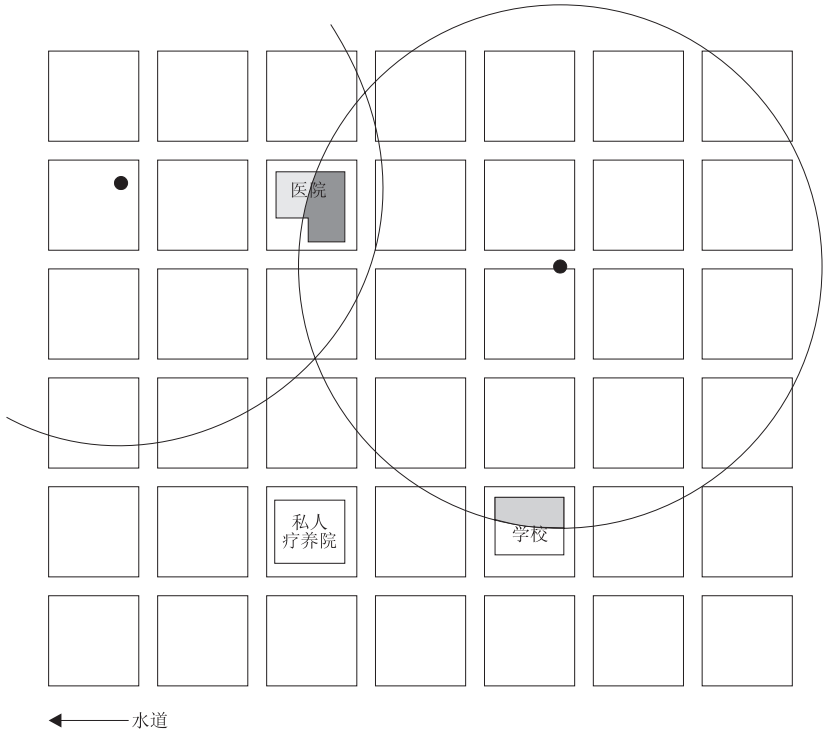


图 32. 使用 ST_Disjoint 查找不位于（相交）任何有害废物区域内的建筑物

ST_Distance

ST_Distance 接受两个几何图形并返回它们之间的最近距离。

语法

db2gse.ST_Distance(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

返回类型

双精度

示例

城市工程师需要任何地块线的一英尺内的所有建筑物的列表。

BUILDINGFOOTPRINTS 表的 BUILDING_ID 列唯一地标识每个建筑物。 LOT_ID 列标识每个建筑物所属的地块。占地形状多边形存储每个建筑物的占地形状的几何图形。

```
CREATE TABLE BUILDINGFOOTPRINTS (   building_id integer,
                                    lot_id      integer,
                                    footprint   db2gse.ST_MultiPolygon);
```

LOTS 表存储唯一地标识每个地块的地块 ID 和包含地块线几何图形的地块复合多边形。

```
CREATE TABLE LOTS ( lot_id   integer,
                    lot      db2gse.ST_MultiPolygon);
```

该查询返回建筑物 ID 的列表，这些建筑物在其地块线的一英尺之内。 ST_Distance 函数在占地形状和地块复合多边形的边界之间执行 Spatial 连接。然而，bf.lot_id 和 LOTS.lot_id 之间的对等连接确保 ST_Distance 函数仅比较属于同一地块的复合多边形。

```
SELECT bf.building_id
  FROM BUILDINGFOOTPRINTS bf, LOTS
 WHERE bf.lot_id = LOTS.lot_id AND
       db2gse.ST_Distance(bf.footprint, db2gse.ST_Boundary(LOTS.lot)) <= 1.0;
```

ST_Endpoint

ST_Endpoint 接受线条并返回该线条的最后一点。

语法

db2gse.ST_Endpoint(c db2gse.ST_Curve)

返回类型

db2gse.ST_Point

示例

ENDPOINT_TEST 表存储 GID 整数列和 LN1 线条列，前者唯一地标识每一行，后者存储线条。

```
CREATE TABLE ENDPOINT_TEST (gid integer, ln1 db2gse.ST_LineString)
```

下列 INSERT 语句将线条插入 ENDPOINT_TEST 表。第一条线条没有 Z 坐标或度量单位；第二个线条则有。

```
INSERT INTO ENDPOINT_TEST
VALUES( 1,
db2gse.ST_LineFromText('linestring (10.02 20.01,23.73 21.92,30.10 40.23)',
                    db2gse.coordref()..srid(0)))

INSERT INTO ENDPOINT_TEST
VALUES (2,
       db2gse.ST_LineFromText('linestring zm (10.02 20.01 5.0 7.0,23.73 21.92 6.5 7.1,
                                  30.10 40.23 6.9 7.2)',
                   db2gse.coordref()..srid(0)))
```

以下 SELECT 语句列示 GID 列以及 ST_Endpoint 函数的输出。 ST_Endpoint 函数生成点几何图形，ST_AsText 函数将该几何图形转换为文本。 CAST 函数用于将 ST_AsText 函数的缺省 varchar(4000) 值缩短至 varchar(60)。

```
SELECT gid, CAST(db2gse.ST_AsText(db2gse.ST_Endpoint(ln1)) AS varchar(60)) "Endpoint"
FROM ENDPOINT_TEST
```

返回下列结果集。

```
GID         Endpoint
----------- ------------------------------------------------------------
          1 POINT ( 30.10000000 40.23000000)
          2 POINT ZM ( 30.10000000 40.23000000 7.00000000 7.20000000)

  2 record(s) selected.
```

ST_Envelope

ST_Envelope 接受几何图形对象并返回它的边框作为几何图形。

语法

db2gse.ST_Envelope(g db2gse.ST_Geometry)

返回类型

db2gse.ST_Geometry

示例

ENVELOPE_TEST 表中的 GEOTYPE 列存储 G1 几何图形列中存储的几何图形子类的名称。

```
CREATE TABLE ENVELOPE_TEST (geotype varchar(20), g1 db2gse.ST_Geometry)
```

下列 INSERT 语句将每个几何图形子类插入 ENVELOPE_TEST 表。

```
INSERT INTO ENVELOPE_TEST
VALUES('Point',
db2gse.ST_PointFromText('point (10.02 20.01)', db2gse.coordref()..srid(0)))

INSERT INTO ENVELOPE_TEST
VALUES ('Linestring',
        db2gse.ST_LineFromText('linestring (10.01 20.01, 10.01 30.01, 10.01 40.01)',
        db2gse.coordref()..srid(0)))

INSERT INTO ENVELOPE_TEST
VALUES ('Linestring',
       db2gse.ST_LineFromText('linestring  (10.02 20.01,10.32 23.98,11.92 25.64)',
       db2gse.coordref()..srid(0)))

INSERT INTO ENVELOPE_TEST
VALUES('Polygon',
       db2gse.ST_PolyFromText('polygon ((10.02 20.01,11.92 35.64,25.02 34.15,
                                         19.15 33.94,10.02 20.01))',
                    db2gse.coordref()..srid(0)))

INSERT INTO ENVELOPE_TEST
VALUES('Multipoint',
       db2gse.ST_MPointFromText('multipoint (10.02 20.01,10.32 23.98,11.92 25.64)',
       db2gse.coordref()..srid(0)))

INSERT INTO ENVELOPE_TEST
VALUES('Multilinestring',
       db2gse.ST_MLineFromText('multilinestring ((10.01 20.01,20.01 20.01,30.01 20.01),
                                  (30.01 20.01,40.01 20.01,50.01 20.01))',
                     db2gse.coordref()..srid(0)))

INSERT INTO ENVELOPE_TEST
VALUES('Multilinestring',
```

```
      db2gse.ST_MLineFromText('multilinestring ((10.02 20.01,10.32 23.98,11.92 25.64),
                                    (9.55 23.75,15.36 30.11))',
                    db2gse.coordref()..srid(0)))

INSERT INTO ENVELOPE_TEST
VALUES('Multipolygon',
db2gse.ST_MPolyFromText('multipolygon (((10.02 20.01,11.92 35.64,25.02 34.15,
                                      19.15 33.94,10.02 20.01)),
                                    ((51.71 21.73,73.36 27.04,71.52 32.87,
                                      52.43 31.90,51.71 21.73)))',
                    db2gse.coordref()..srid(0)))
```

下列 SELECT 语句将子类名列示在它的包络的旁边。因为 ST_Envelope 函数返回点、线条或多边形，所以用 ST_AsText 函数将它的输出转换为文本。CAST 函数将 ST_AsText 函数的缺省 varchar(4000) 结果转换为 varchar(280)。

```
SELECT GEOTYPE, CAST(db2gse.ST_AsText(db2gse.ST_Envelope(g1)) AS varchar(280))
"The envelope"
FROM ENVELOPE_TEST
```

返回下列结果集。

```
GEOTYPE              The envelope
-------------------- ------------------------------------------------------------
Point                POINT ( 10.02000000 20.01000000)
Linestring           LINESTRING ( 10.01000000 20.01000000, 10.01000000
 40.01000000)
Linestring           POLYGON (( 10.02000000 20.01000000, 11.92000000
 20.01000000, 11.92000000 25.64000000, 10.02000000 25.64000000, 10.02000000
 20.01000000))
Polygon              POLYGON (( 10.02000000 20.01000000, 25.02000000
 20.01000000, 25.02000000 35.64000000, 10.02000000 35.64000000, 10.02000000
 20.01000000))
Multipoint           POLYGON (( 10.02000000 20.01000000, 11.92000000
 20.01000000, 11.92000000 25.64000000, 10.02000000 25.64000000, 10.02000000
 20.01000000))
Multilinestring      LINESTRING ( 10.01000000 20.01000000, 50.01000000
 20.01000000)
Multilinestring      POLYGON (( 9.55000000 20.01000000, 15.36000000
 20.01000000, 15.36000000 30.11000000, 9.55000000 30.11000000, 9.55000000
 20.01000000))
Multipolygon         POLYGON (( 10.02000000 20.01000000, 73.36000000
 20.01000000, 73.36000000 35.64000000, 10.02000000 35.64000000, 10.02000000
 20.01000000))

  8 record(s) selected.
```

ST_Equals

ST_Equals 比较两个几何图形，若这两个几何图形相同，则返回 1 (TRUE)；否则，它返回 0 (FALSE)。

语法

db2gse.ST_Equals(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

返回类型

整数

示例

城市 GIS 技术人员怀疑 BUILDINGFOOTPRINTS 表中的某些数据由于某种原因重复了。该技术人员查询该表以确定任何占地形状的复合多边形是否是相等的。

BUILDINGFOOTPRINTS 表是用以下语句创建的。 BUILDING_ID 列唯一地标识建筑物；LOT_ID 列标识该建筑物的地块； FOOTPRINT 列存储该建筑物的几何图形。

```
CREATE TABLE BUILDINGFOOTPRINTS (   building_id integer,
                                    lot_id      integer,
                                    footprint   db2gse.ST_MultiPolygon);
```

ST_Equals 谓词将 BUILDINGFOOTPRINTS 表与它自身进行 Spatial 连接，无论何时发现两个复合多边形相等，就会返回 1。需要 bf1.building_id <> bf2.building_id 条件来消除同一几何图形的比较。

```
SELECT bf1.building_id, bf2.building_id
FROM BUILDINGFOOTPRINTS bf1, BUILDINGFOOTPRINTS bf2
WHERE db2gse.ST_Equals(bf1.footprint,bf2.footprint) = 1
      and bf1.building_id <> bf2.building_id;
```

ST_ExteriorRing

ST_ExteriorRing 接受多边形并以线条的形式返回它的外环。

语法

db2gse.ST_ExteriorRing(s db2gse.ST_Polygon)

返回类型

db2gse.ST_LineString

示例

正在几个南海岛屿上研究鸟群的某个鸟类学家知道特定鸟类的进食区限于海岸线。要计算海岛的承载能力，该鸟类学家需要知道海岛的周长。虽然某些海岛上有几个池塘，但是这些池塘的池岸线被另一种更具攻击性的鸟类独占。因此，该鸟类学家需要知道海岛外环的周长。

ISLANDS 表的 ID 和 NAME 列标识每个海岛，类型为 ST_Polygon 的 LAND 列存储每个海岛的几何图形。

```
CREATE TABLE ISLANDS (id    integer,
                      name  varchar(32),
                      land  db2gse.ST_Polygon);
```

ST_ExteriorRing 函数从每个海岛多边形抽出外环作为线条。通过 length 函数计算线条的长度。通过 SUM 函数求出线条长度之和。

```
SELECT SUM(db2gse.ST_length(db2gse.ST_ExteriorRing (land))) FROM ISLANDS;
```

在第195页的图33中，海岛的外环表示每个海岛与海共享的生态分界面。某些海岛有湖泊，用多边形的内环表示这些湖泊。

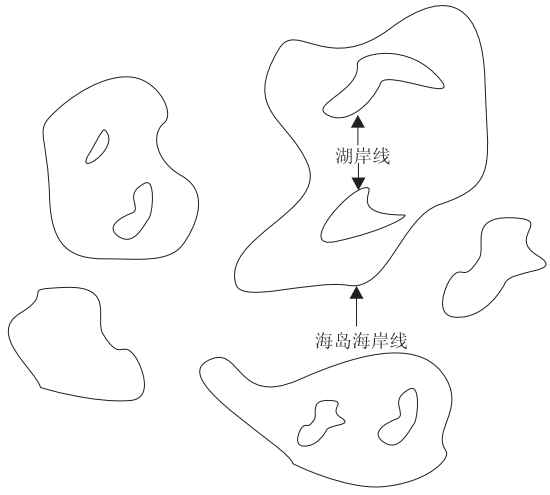


图 33. 使用 ST_ExteriorRing 确定海岛海岸线的长度

ST_GeometryFromText

ST_GeometryFromText 接受公认文本表示和 Spatial 参考系标识，并返回几何图形对象。

语法

db2gse.ST_GeometryFromText(geometryTaggedText Varchar(4000), cr db2gse.coordref)

返回类型

db2gse.ST_Geometry

示例

GEOMETRY_TEST 表包含整数 GID 列和 G1 列，前者唯一地标识每行，后者存储几何图形。

```
CREATE TABLE GEOMETRY_TEST (gid smallint, g1 db2gse.ST_Geometry)
```

下列 INSERT 语句将数据插入 GEOMETRY_TEST 表的 GID 列和 G1 列。ST_GeometryFromText 函数将每个几何图形的文本表示转换为它对应的 DB2 Spatial Extender 可例示子类。

```
INSERT INTO GEOMETRY_TEST
VALUES(1, db2gse.ST_GeometryFromText('point (10.02 20.01)', db2gse.coordref()..srid(0)))

INSERT INTO GEOMETRY_TEST
VALUES (2,
        db2gse.ST_GeometryFromText('linestring (10.01 20.01, 10.01 30.01, 10.01 40.01)',
                        db2gse.coordref()..srid(0)))
```

```
INSERT INTO GEOMETRY_TEST
VALUES(3,
       db2gse.ST_GeometryFromText('polygon ((10.02 20.01,11.92 35.64,25.02 34.15,
                                             19.15 33.94,10.02 20.01))',
                          db2gse.coordref()..srid(0)))

INSERT INTO GEOMETRY_TEST
VALUES(4,
       db2gse.ST_GeometryFromText('multipoint (10.02 20.01,10.32 23.98,11.92 25.64)',
                          db2gse.coordref()..srid(0)))

INSERT INTO GEOMETRY_TEST
VALUES(5,
       db2gse.ST_GeometryFromText('multilinestring ((10.02 20.01,10.32 23.98,
                                                    11.92 25.64),
                                           (9.55 23.75,15.36 30.11))',
                          db2gse.coordref()..srid(0)))

INSERT INTO GEOMETRY_TEST
VALUES(6,
       db2gse.ST_GeometryFromText('multipolygon (((10.02 20.01,11.92 35.64,25.02 34.15,
                                             19.15 33.94,10.02 20.01)),
                                           ((51.71 21.73,73.36 27.04,71.52 32.87,
                                             52.43 31.90,51.71 21.73)))',
                          db2gse.coordref()..srid(0)))
```

ST_GeomFromWKB

ST_GeomFromWKB 接受公认二进制表示和 Spatial 参考系标识，并返回几何图形对象。

语法

db2gse.ST_GeomFromWKB(WKBGeometry Blob(1M), cr db2gse.coordref)

返回类型

db2gse.ST_Geometry

示例

以下 C 代码段包含 ODBC 函数，这些函数嵌有将数据插入 LOTS 表的 DB2 Spatial Extender SQL 函数。

已创建了 LOTS 表，该表具有两列：LOT_ID 列和 LOT 复合多边形列，前者唯一标识每个地块；后者包含每个地块的几何图形。

```
CREATE TABLE LOTS ( lot_id   integer,
                    lot      db2gse.ST_MultiPolygon);
```

ST_GeomFromWKB 函数将 WKB 表示转换为 DB2 Spatial Extender 几何图形。整个 INSERT 语句被复制到 wkb_sql 字符串。该 INSERT 语句包含参数标记以动态接受 LOT_ID 数据和 LOT 数据。

```
/* Create the SQL insert statement to populate the lot id and the
   lot multipolygon. The question marks are parameter markers that
   indicate the lot_id and lot values that will be retrieved at
   runtime. */
strcpy (wkb_sql,"insert into LOTS (lot_id, lot) values (?, db2gse.ST_GeomFromWKB

(cast(? as blob(1m)), db2gse.coordref()..srid(0)))");

/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);

/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)wkb_sql, SQL_NTS);

/* Bind the integer key value to the first parameter. */
pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_SLONG,
     SQL_INTEGER, 0, 0, &lot_id, 0, &pcbvalue1);

/* Bind the shape to the second parameter. */
pcbvalue2 = blob_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
```

```
        SQL_BLOB, blob_len, 0, shape_blob, blob_len, &pcbvalue2);

    /* Execute the insert statement. */
    rc = SQLExecute (hstmt);
```

ST_GeometryN

ST_GeometryN 接受集合和整数索引并返回该集合中的第 *n* 个几何图形对象。

语法

db2gse.ST GeometryN(g db2gse.ST_GeomCollection, n Integer)

返回类型

db2gse.ST_Geometry

示例

城市工程师需要知道建筑物占地形状是否全在地块的复合多边形的第一个多边形内。

BUILDING_ID 列唯一地标识 BUILDINGFOOTPRINTS 表的每行。 LOT_ID 列标识建筑物的地块。FOOTPRINT 列存储建筑物的几何图形。

```
CREATE TABLE BUILDINGFOOTPRINTS (   building_id integer,
                                    lot_id      integer,
                                    footprint   db2gse.ST_MultiPolygon);

CREATE TABLE LOTS ( lot_id   integer,
                    lot   db2gse.ST_MultiPolygon);
```

该查询列示全在第一个地块多边形内的所有建筑物占地形状的 BUILDINGFOOTPRINTS building_id 和 lot_id。ST_GeometryN 函数返回复合多边形数组中的第一个地块多边形。

```
SELECT bf.building_id,bf.lot_id
FROM BUILDINGFOOTPRINTS bf,LOTS
WHERE db2gse.ST_Within(footprint, db2gse.ST_GeometryN (lot,1)) = 1
      AND bf.lot_id = LOTS.lot_id;
```

ST_GeometryType

ST_GeometryType 接受 ST_Geometry 对象并以字符串形式返回它的几何图形类型。

语法

db2gse.ST_GeometryType (g db2gse.ST_Geometry)

返回类型

Varchar(4000)

示例

GEOMETRYTYPE_TEST 表包含 G1 几何图形列。

```
CREATE TABLE GEOMETRYTYPE_TEST(g1 db2gse.ST_Geometry)
```

下列 INSERT 语句将每个几何图形子类插入 G1 列。

```
INSERT INTO GEOMETRYTYPE_TEST
VALUES(db2gse.ST_GeometryFromText('point (10.02 20.01)', db2gse.coordref()..srid(0)))

INSERT INTO GEOMETRYTYPE_TEST
VALUES (db2gse.ST_GeometryFromText('linestring (10.01 20.01, 10.01 30.01, 10.01 40.01)',
                         db2gse.coordref()..srid(0)))

INSERT INTO GEOMETRYTYPE_TEST
VALUES(db2gse.ST_Geometrytype_test values(db2gse.ST_GeomFromText('polygon ((10.02
20.01,11.92 35.64,25.02 34.15,19.15 33.94, 10.02 20.01))',
                        db2gse.coordref()..srid(0))))

INSERT INTO GEOMETRYTYPE_TEST
VALUES(db2gse.ST_GeometryFromText('multipoint (10.02
 20.01,10.32 23.98,11.92 25.64)',
                        db2gse.coordref()..srid(0)))

INSERT INTO GEOMETRYTYPE_TEST
VALUES(db2gse.ST_GeometryFromText('multilinestring ((10.02 20.01,10.32 23.98,
                                          11.92 25.64),
                                       (9.55 23.75,15.36 30.11))',
                        db2gse.coordref()..srid(0)))

INSERT INTO GEOMETRYTYPE_TEST
VALUES(db2gse.ST_GeometryFromText('multipolygon (((10.02 20.01,11.92 35.64,25.02 34.15,
                                            19.15 33.94,10.02 20.01)),
                                          ((51.71 21.73,73.36 27.04,71.52 32.87,
                                            52.43 31.90,51.71 21.73)))',
                        db2gse.coordref()..srid(0)))
```

下列 SELECT 语句列示 G1 几何图形列中存储的每个子类的几何图形类型。

```
SELECT db2gse.ST_GeometryType(g1) "Geometry type" FROM GEOMETRYTYPE_TEST
```

返回下列结果集。

```
Geometry type
--------------------------------
ST_Point
ST_LineString
ST_Polygon
ST_MultiPoint
ST_MultiLineString
ST_MultiPolygon

6 record(s) selected.
```

ST_InteriorRingN

以线条形式返回多边形的第 *n* 个内环。这些环不是按几何方向组织的。它们是根据内部几何图形验证例程定义的规则组织的。因此，不能预定义环的次序。

语法

ST_InteriorRingN(p ST_Polygon, n Integer)

返回类型

db2gse.ST_LineString

示例

正在几个南海岛屿上研究鸟群的鸟类学家知道特定温顺鸟类的进食区限于海岸线。某些海岛上有几个湖泊。这些湖泊的湖岸线被另一种更具攻击性的鸟类独占。该鸟类学家知道，对于每个海岛，若湖泊的周长超过某个阈值，攻击性鸟类会变得太多，以致威胁温顺的海岸鸟类。因此，鸟类学家需要这些海岛的内环的周长总和。

在图34中，海岛的外环表示每个海岛与海共享的生态分界面。某些海岛有湖泊，用多边形的内环表示这些湖泊。

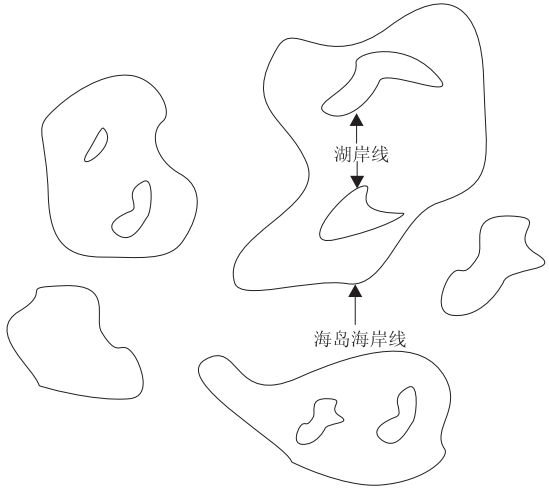


图 34. 使用 ST_InteriorRingN 确定每个海岛内的湖岸的长度

ISLANDS 表的 ID 和 name 列标识每个海岛，而 land 多边形列存储海岛的几何图形。

```
CREATE TABLE ISLANDS (id    integer,
                      name  varchar(32),
                      land  db2gse.ST_Polygon);
```

以下 ODBC 程序使用 ST_InteriorRingN 函数从每个海岛多边形抽出内环（湖泊）作为线条。对 length 函数返回的线条的周长求和并将其与海岛的 ID 显示在一起。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include "sg.h"
#include "sgerr.h"
#include "sqlcli1.h"

/***                         ***
 *** Change these constants ***
 ***                         ***/

#define USER_NAME   "sdetest"   /* your user name */
#define USER_PASS   "acid.rain" /* your user password */
#define DB_NAME     "mydb"      /* database to connect to */

static void check_sql_err (SQLHDBC  handle,
                           SQLHSTMT hstmt,
  LONG          rc,
                           CHAR     *str);

void main( argc, argv )
int argc;
char *argv[];
{
  SQLHDBC       handle;
  SQLHENV       henv;
  CHAR          sql_stmt[256];
  LONG          rc,
                total_perimeter,
                num_lakes,
                lake_number,
                island_id,
                lake_perimeter;
  SQLHSTMT      island_cursor,
                lake_cursor;
  SDWORD        pcbvalue,
                id_ind,
                lake_ind,
                length_ind;

/* Allocate memory for the ODBC environment handle henv and initialize the application. */

  rc = SQLAllocEnv (&henv);
    if (rc != SQL_SUCCESS)
  {
    printf ("SQLAllocEnv failed with %d\n", rc);
    exit(0);
  }

/* Allocate memory for a connection handle within the henv environment. */

  rc = SQLAllocConnect (henv, &handle);
    if (rc != SQL_SUCCESS)
  {
    printf ("SQLAllocConnect failed with %d\n", rc);
    exit(0);
  }
```

```
/* Load the ODBC driver and connect to the data source identified by the database,
   user, and password.*/

  rc = SQLConnect (handle,
                   (UCHAR *)DB_NAME,
                   SQL_NTS,
                   (UCHAR *)USER_NAME,
                   SQL_NTS,
                   (UCHAR *)USER_PASS,
                   SQL_NTS);

  check_sql_err (handle, NULL, rc, "SQLConnect");

/* Allocate memory to the SQL statement handle island_cursor. */

  rc = SQLAllocStmt (handle, &island_cursor);
  check_sql_err (handle, NULL, rc, "SQLAllocStmt");

/* Prepare and execute the query to get the island IDs and number of
   lakes (interior rings) */

  strcpy (sql_stmt, "select id, db2gse.ST_NumInteriorRings(land) from ISLANDS");

  rc = SQLExecDirect (island_cursor, (UCHAR *)sql_stmt, SQL_NTS);
  check_sql_err (NULL, island_cursor, rc, "SQLExecDirect");

/* Bind the island table's ID column to the variable island_id */

  rc = SQLBindCol (island_cursor, 1, SQL_C_SLONG, &island_id, 0, &id_ind);
  check_sql_err (NULL, island_cursor, rc, "SQLBindCol");

/* Bind the result of numinteriorrings(land) to the num_lakes variable. */

  rc = SQLBindCol (island_cursor, 2, SQL_C_SLONG, &num_lakes, 0, &lake_ind);
  check_sql_err (NULL, island_cursor, rc, "SQLBindCol");

/* Allocate memory to the SQL statement handle lake_cursor. */

rc = SQLAllocStmt (handle, &lake_cursor);
  check_sql_err (handle, NULL, rc, "SQLAllocStmt");

/* Prepare the query to get the length of an interior ring. */

  strcpy (sql_stmt,
          "select Length(db2gse.ST_InteriorRingN(land, cast (? as
           integer))) from ISLANDS where id = ?");

  rc = SQLPrepare (lake_cursor, (UCHAR *)sql_stmt, SQL_NTS);
  check_sql_err (NULL, lake_cursor, rc, "SQLPrepare");

/* Bind the lake_number variable as the first input parameter */

  pcbvalue = 0;
  rc = SQLBindParameter (lake_cursor, 1, SQL_PARAM_INPUT, SQL_C_LONG,
       SQL_INTEGER, 0, 0, &lake_number, 0, &pcbvalue);
  check_sql_err (NULL, lake_cursor, rc, "SQLBindParameter");

/* Bind the island_id as the second input parameter */

  pcbvalue = 0;
  rc = SQLBindParameter (lake_cursor, 2, SQL_PARAM_INPUT, SQL_C_LONG,
       SQL_INTEGER, 0, 0, &island_id, 0, &pcbvalue);
  check_sql_err (NULL, lake_cursor, rc, "SQLBindParameter");

/* Bind the result of the Length(db2gse.ST_InteriorRingN(land, cast (? as integer)))
   to the variable lake_perimeter */

  rc = SQLBindCol (lake_cursor, 1, SQL_C_SLONG, &lake_perimeter, 0,
                   &length_ind);
  check_sql_err (NULL, island_cursor, rc, "SQLBindCol");
```

```
  /* Outer loop, get the island ids and the number of lakes (interior rings) */

  while (SQL_SUCCESS == rc)
  {
    /* Fetch an island */

    rc = SQLFetch (island_cursor);

    if (rc != SQL_NO_DATA)
    {
      check_sql_err (NULL, island_cursor, rc, "SQLFetch");

      /* Inner loop, for this island, get the perimeter of all of
         its lakes (interior rings) */

      for (total_perimeter = 0,lake_number = 1;
           lake_number <= num_lakes;
           lake_number++)
      {
        rc = SQLExecute (lake_cursor);
        check_sql_err (NULL, lake_cursor, rc, "SQLExecute");

        rc = SQLFetch (lake_cursor);
        check_sql_err (NULL, lake_cursor, rc, "SQLFetch");

        total_perimeter += lake_perimeter;

        SQLFreeStmt (lake_cursor, SQL_CLOSE);
      }
    }

/* Display the Island id and the total perimeter of its lakes. */

    printf ("Island ID = %d,  Total lake perimeter = %d\n",
             island_id,total_perimeter);

  }

  SQLFreeStmt (lake_cursor, SQL_DROP);
  SQLFreeStmt (island_cursor, SQL_DROP);
        SQLDisconnect (handle);
        SQLFreeConnect (handle);
  SQLFreeEnv (henv);

  printf( "\nTest Complete ...\n" );

}

static void check_sql_err (SQLHDBC handle, SQLHSTMT  hstmt, LONG rc,
                           CHAR *str)
{

    SDWORD dbms_err = 0;
    SWORD  length;
    UCHAR   err_msg[SQL_MAX_MESSAGE_LENGTH], state[6];

    if (rc != SQL_SUCCESS)
    {
      SQLError (SQL_NULL_HENV, handle, hstmt, state, &dbms_err,
                err_msg, SQL_MAX_MESSAGE_LENGTH - 1, &length);
      printf ("%s ERROR (%d): DBMS code:%d, SQL state: %s, message: \n %s\n",
               str, rc, dbms_err, state, err_msg);

      if (handle)
      {
        SQLDisconnect (handle);
        SQLFreeConnect (handle);
      }
```

```
        exit(1);
      }
  }
```

ST_Intersection

ST_Intersection 接受两个几何图形对象并以几何图形对象形式返回交集。

语法

db2gse.ST_Intersection(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

返回类型

db2gse.ST_Geometry

示例

消防队长必须获取与可能的有害废物污染的半径相交的医院、学校和私人疗养院区域。

敏感区域存储在用以下 CREATE TABLE 语句创建的表 SENSITIVE_AREAS 中。ZONE 列定义为存储每个敏感区域的轮廓的多边形。

```
CREATE TABLE SENSITIVE_AREAS (id        integer,
                              name      varchar(128),
                              size      float,
                              type      varchar(10),
                              zone      db2gse.ST_Polygon);
```

危险地点存储在用以下 CREATE TABLE 语句创建的 HAZARDOUS_SITES 表中。定义为点的 LOCATION 列存储每个危险地点的地理中心的位置。

```
CREATE TABLE HAZARDOUS_SITES (site_id   integer,
                              name      varchar(128),
                              location  db2gse.ST_Point);
```

buffer 函数生成环绕有害废物地点位置的五英里缓冲区。 ST_Intersection 函数从缓冲的有害废物地点多边形和敏感区域的交集生成多边形。 ST_Area 函数返回交集多边形的面积，由 SUM 函数对每个危险地点求出该面积。 GROUP BY 子句指示查询按有害废物 site_ID 汇总相交的面积。

```
SELECT hs.name,SUM(db2gse.ST_Area(db2gse.ST_Intersection (sa.zone,
db2gse.ST_buffer hs.location,(5 * 5280))))
FROM SENSITIVE_AREAS sa, HAZARDOUS_SITES hs
GROUP BY hs.site_id;
```

在第208页的图35中，圆表示有害废物地点周围的缓冲多边形。这些缓冲多边形与敏感区域的交集产生其他三个多边形。左上角的医院相交了两次，而右下角的学校只相交了一次。

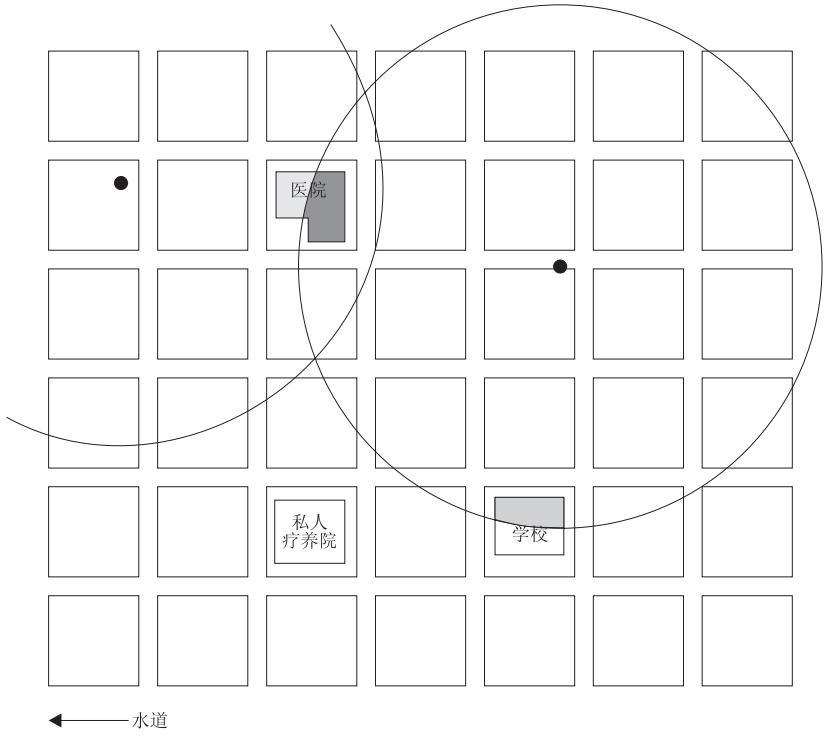


图 35. 使用 ST_Intersection 确定每个建筑物中可能有多大面积受有害废物影响

ST_Intersects

ST_Intersects 接受两个几何图形，若两个几何图形的交集不产生空集，则返回 1 (TRUE)。否则，它返回 0 (FALSE)。

语法

db2gse.ST_Intersects(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

返回类型

整数

示例

消防队长需要在有害废物地点的五英里半径内的所有敏感区域的列表。

敏感区域存储在用以下 CREATE TABLE 语句创建的表 SENSITIVE_AREAS 中。ZONE 列定义为存储每个敏感区域的轮廓的多边形。

```
CREATE TABLE SENSITIVE_AREAS (id        integer,
                              name      varchar(128),
                              size      float,
                              type      varchar(10),
                              zone      db2gse.ST_Polygon);
```

危险地点存储在用以下 CREATE TABLE 语句创建的 HAZARDOUS_SITES 表中。定义为点的 LOCATION 列存储每个危险地点的地理中心的位置。

```
CREATE TABLE HAZARDOUS_SITES (site_id   integer,
                              name      varchar(128),
                              location  db2gse.ST_Point);
```

该查询对与危险地点的五英里缓冲区相交的敏感区域返回敏感区域和危险地点名称的列表。

```
SELECT sa.name, hs.name
FROM SENSITIVE_AREAS sa, HAZARDOUS_SITES hs
WHERE db2gse.ST_Intersects(db2gse.ST_Buffer(hs.location,(5 * 5280)),sa.zone) = 1;
```

ST_IsClosed

ST_IsClosed 接受线条或多线条，若该线条是封闭的，则返回 1 (TRUE)；否则，它返回 0 (FALSE)。

语法

db2gse.ST_IsClosed(c db2gse.ST_Curve)
db2gse.ST_IsClosed(mc db2gse.ST_MultiCurve)

返回类型

整数

示例

以下 CREATE TABLE 语句创建 CLOSED_LINESTRING 表，它有单个线条列。

```
CREATE TABLE CLOSED_LINESTRING (ln1 db2gse.ST_LineString)
```

以下 INSERT 语句将两个记录插入 CLOSED_LINESTRING 表。第一个记录不是封闭的线条，而第二个记录则是。

```
INSERT INTO CLOSED_LINESTRING
VALUES(db2gse.ST_LineFromText('linestring  (10.02 20.01,10.32 23.98,11.92 25.64)',
                    db2gse.coordref()..srid(0)))

INSERT INTO CLOSED_LINESTRING
VALUES(db2gse.ST_LineFromText('linestring (10.02 20.01,11.92 35.64,25.02 34.15,
                                  19.15 33.94,10.02 20.01)',
                    db2gse.coordref()..srid(0)))
```

以下 SELECT 语句和相应结果集显示 ST_IsClosed 函数的结果。第一行返回 0，因为该线条不是封闭的；而第二行返回 1，因为该线条是封闭的。

```
SELECT db2gse.ST_IsClosed(ln1) "Is it closed" FROM CLOSED_LINESTRING

Is it closed
------------
0
1

  2 record(s) selected.
```

以下 CREATE TABLE 语句创建 CLOSED_MULTILINESTRING 表，它具有单个多线条列。

```
CREATE TABLE CLOSED_MULTILINESTRING (mln1 db2gse.ST_MultiLineString)
```

以下 INSERT 语句将两个记录插入 CLOSED_MULTILINESTRING，一个不封闭的多线条记录和一个封闭的多线条记录。

```
INSERT INTO CLOSED_MULTILINESTRING
VALUES(db2gse.ST_MLineFromText('multilinestring ((10.02 20.01,10.32 23.98,11.92 25.64),
                                      (9.55 23.75,15.36 30.11))',
                     db2gse.coordref()..srid(0)))

INSERT INTO CLOSED_MULTILINESTRING
VALUES(db2gse.ST_MLineFromText('multilinestring ((10.02 20.01,11.92 35.64,25.02 34.15,
                                       19.15 33.94,10.02 20.01),
                                      (51.71 21.73,73.36 27.04,71.52 32.87,
                                       52.43 31.90,51.71 21.73))',
                     db2gse.coordref()..srid(0)))
```

以下 SELECT 语句和相应结果集显示 ST_IsClosed 函数的结果。第一行返回 0，因为该多线条不是封闭的；而第二行返回 1，因为该多线条是封闭的。若多线条的所有线条元素是封闭的，则该多线条是封闭的。

```
SELECT db2gse.ST_IsClosed(mln1) "Is it closed" FROM CLOSED_MULTILINESTRING

Is it closed
------------
0
1

  2 record(s) selected.
```

ST_IsEmpty

ST_IsEmpty 接受几何图形对象，若该几何图形对象是空的，则返回 1 (TRUE)；否则，它返回 0 (FALSE)。

语法

db2gse.ST_IsEmpty(g db2gse.ST_Geometry)

返回类型

整数

示例

以下 CREATE TABLE 语句创建具有两列的 EMPTY_TEST 表。 GEOTYPE 列存储 G1 几何图形列中存储的子类的数据类型。

```
CREATE TABLE EMPTY_TEST (geotype varchar(20), g1 db2gse.ST_Geometry)
```

下列 INSERT 语句为几何图形子类点、线条和多边形插入两个记录。一个记录是空的，另一个则不是。

```
INSERT INTO EMPTY_TEST
VALUES('Point', db2gse.ST_PointFromText('point (10.02 20.01)',
db2gse.coordref()..srid(0)))

INSERT INTO EMPTY_TEST
VALUES('Point', db2gse.ST_PointFromText('point empty', db2gse.coordref()..srid(0)))

INSERT INTO EMPTY_TEST
VALUES('Linestring', db2gse.ST_LineFromText('linestring (10.02 20.01,10.32 23.98,
                                             11.92 25.64)',
                                  db2gse.coordref()..srid(0)))

INSERT INTO EMPTY_TEST
VALUES('Linestring', db2gse.ST_LineFromText('linestring  empty',
db2gse.coordref()..srid(0)))

INSERT INTO EMPTY_TEST
VALUES('Polygon', db2gse.ST_PolyFromText('polygon ((10.02 20.01,11.92 35.64,
25.02 34.15,19.15 33.94,10.02 20.01))',
                               db2gse.coordref()..srid(0)))

INSERT INTO EMPTY_TEST
VALUES('Polygon', db2gse.ST_PolyFromText('polygon  empty', db2gse.coordref()..srid(0)))
```

以下 SELECT 语句和相应结果集显示来自 GEOTYPE 列的几何图形类型和 ST_IsEmpty 函数的结果。

```
SELECT geotype, db2gse.ST_IsEmpty(g1) "It is empty" FROM EMPTY_TEST

GEOTYPE                  It is empty
```

```
--------------------      -----------
ST_Point                          0
ST_Point                          1
ST_Linestring                     0
ST_Linestring                     1
ST_Polygon                        0
ST_Polygon                        1

6 record(s) selected.
```

ST_IsRing

ST_IsRing 接受线条，若该线条为环（即该线条是封闭且简单的），则返回 1 (TRUE)；否则，它返回 0 (FALSE)。

语法

db2gse.ST_IsRing(c db2gse.ST_Curve)

返回类型

整数

示例

以下 CREATE TABLE 语句创建 RING_LINESTRING 表，该表具有称作 LN1 的单个线条列。

```
CREATE TABLE RING_LINESTRING (ln1 db2gse.ST_LineString)
```

下列 INSERT 语句将三个线条插入 LN1 列。第一行包含不封闭因此不是环的线条。第二行包含封闭且简单因此是环的线条。第三行包含封闭但不简单的线条，因为它与它自己的内部相交，因此不是环。

```
INSERT INTO RING_LINESTRING
VALUES(db2gse.ST_LineFromText('linestring  (10.02 20.01,10.32 23.98,11.92 25.64)',
                    db2gse.coordref()..srid(0)))

INSERT INTO RING_LINESTRING
VALUES(db2gse.ST_LineFromText('linestring (10.02 20.01,11.92 35.64,25.02 34.15,
                                  19.15 33.94, 10.02 20.01)',
                    db2gse.coordref()..srid(0)))

INSERT INTO RING_LINESTRING
VALUES(db2gse.ST_LineFromText('linestring (15.47 30.12,20.73 22.12,10.83 14.13,
                                  16.45 17.24,21.56 13.37,11.23 22.56,
                                  19.11 26.78,15.47 30.12)',
                    db2gse.coordref()..srid(0)))
```

以下 SELECT 语句和相应结果集显示 ST_IsRing 函数的结果。第一和第三行返回 0。这是因为这两个线条不是环，而第二行返回 1，因为它是环。

```
SELECT db2gse.ST_IsRing(ln1) "Is it ring" FROM RING_LINESTRING

Is it ring
-----------
0
1
0

  3 record(s) selected.
```

ST_IsSimple

ST_IsSimple 接受几何图形对象，若该对象是简单的，则返回 1 (TRUE)；否则，它返回 0 (FALSE)。

语法

db2gse.ST_IsSimple(g db2gse.ST_Geometry)

返回类型

整数

示例

以下 CREATE TABLE 语句创建 ISSIMPLE_TEST 表，该表具有两列。 PID 列为小整数，包含每行的唯一标识符。 G1 几何图形列存储简单和不简单的几何图形样本。

```
CREATE TABLE ISSIMPLE_TEST (pid smallint, g1 db2gse.ST_Geometry)
```

下列 INSERT 语句将两个记录插入 ISSIMPLE_TEST 表。第一个记录是简单的，因为它是不与自己内部相交的线条。第二个记录不是简单的，因为它与自己的内部相交。

```
INSERT INTO ISSIMPLE_TEST
VALUES (1, db2gse.ST_LineFromText('linestring (10 10, 20 20, 30 30)',
db2gse.coordref()..srid(0)))

INSERT INTO ISSIMPLE_TEST
VALUES (2, db2gse.ST_LineFromText('linestring (10 10,20 20,20 30,10 30,10 20,20 10)',
db2gse.coordref()..srid(0)))
```

以下 SELECT 语句和相应结果集显示 ST_IsSimple 函数的结果。第一行返回 1，因为该线条是简单的；而第二行返回 1，因为该线串不是简单的。

```
SELECT ST_IsSimple(g1)
FROM ISSIMPLE_TEST

g1
--------------
1
0
```

ST_IsValid

ST_IsValid 接受 ST_Geometry，若 ST_Geometry 有效，则返回 1 (TRUE)；否则，它返回 0 (FALSE)。 插入 DB2 数据库的几何图形将总是有效的，因为 DB2 Spatial Extender 在接受它的 Spatial 数据之前总是验证它。然而，其他 DBMS 供应商可能不验证输入，而是要求应用程序执行进行验证。

语法

db2gse.ST_IsValid(g db2gse.ST_Geometry)

返回类型

整数

示例

创建具有 geotype 和 g1 列的 valid_test 表。 geotype 列存储 g1 几何图形列中存储的几何图形子类的名称。

```
CREATE TABLE valid_test (geotype varchar(20), g1 db2gse.ST_Geometry)
```

下列 INSERT 语句将样本子类插入 valid_test 表。

```
INSERT INTO valid_test VALUES(
    'Point', db2gse.ST_PointFromText('point (10.02 20.01)', db2gse.coordref()..srid(0))
)

INSERT INTO valid_test VALUES(
'Linestring',
       db2gse.ST_LineFromText('linestring  (10.02 20.01,10.32 23.98,11.92 25.64)',
                         db2gse.coordref()..srid(0))
)

INSERT INTO valid_test VALUES(
'Polygon', db2gse.ST_PolyFromText('polygon ((10.02 20.01,11.92 35.64,25.02 34.15,
19.15 33.94,10.02 20.01))', db2gse.coordref()..srid(0))
)

INSERT INTO valid_test VALUES(
'Multipoint', db2gse.ST_MPointFromText('multipoint (10.02 20.01,10.32 23.98,
11.92 25.64)', db2gse.coordref()..srid(0))
)

INSERT INTO valid_test VALUES(
'Multilinestring', db2gse.ST_MLineFromText('multilinestring ((10.02 20.01,
10.32 23.98,11.92 25.64),(9.55 23.75,15.36 30.11))', db2gse.coordref()..srid(0))
)

INSERT INTO valid_test VALUES(
'Multipolygon',
```

```
db2gse.ST_MPolyFromText('multipolygon (((10.02 20.01,11.92 35.64,25.02 34.15,
19.15 33.94,10.02 20.01)),((51.71 21.73,73.36 27.04,71.52 32.87,
52.43 31.90,51.71 21.73)))', db2gse.coordref()..srid(0))
)
```

该 SELECT 语句列示 geotype 列中存储的子类名和该 geotype 的维数。

```
SELECT geotype, db2gse.ST_IsValid(g1) Valid FROM valid_test

GEOTYPE              Valid
-------------------- -------------
ST_Point                       1
ST_Linestring                  1
ST_Polygon                     1
ST_Multipoint                  1
ST_Multilinestring             1
ST_Multipolygon                1

6 record(s) selected.
```

ST_Length

ST_Length 接受线条或多线条，并返回它的长度。

语法

db2gse.ST_Length(c db2gse.ST_Curve)
db2gse.ST_Length(mc db2gse.ST_MultiCurve)

返回类型

双精度

示例

某个本地生态学家正研究该县水道中的鲑鱼群的迁移模式。该生态学家想要获取流过该县的所有溪流和江河系统的长度。

以下 CREATE TABLE 语句创建 WATERWAYS 表。 ID 和 NAME 列标识表中存储的每个溪流和江河系统。 WATER 列为多线条，因为江河和溪流系统通常是几个线条的集合。

```
CREATE TABLE WATERWAYS (id integer, name varchar(128),
                            water      db2gse.ST_MultiLineString);
```

以下 SELECT 语句使用 ST_Length 函数返回该县内的每个水道的名称和长度。

```
SELECT name, db2gse.ST_Length(water) "Length"
FROM WATERWAYS;
```

第219页的图36显示位于该县边界内的江河和溪流系统。

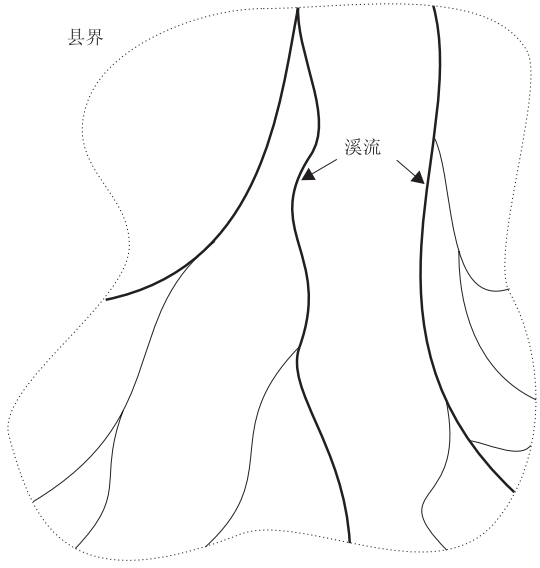


图 36. 使用 ST_Length 确定县内水道的总长

ST_LineFromText

ST_LineFromText 接受线条类型的公认文本表示和 Spatial 参考系标识，并返回线条。

语法

db2gse.ST_LineFromText(lineStringTaggedText Varchar(4000), cr db2gse.coordref)

返回类型

db2gse.ST_LineString

示例

以下 CREATE TABLE 语句创建 LINESTRING_TEST 表，该表具有单个 LN1 线条列。

```
CREATE TABLE LINESTRING_TEST (ln1 db2gse.ST_LineString)
```

下列 INSERT 语句通过使用 ST_LineFromText 函数将线条插入 LN1 列。

```
INSERT INTO LINESTRING_TEST
VALUES (db2gse.ST_LineFromText('linestring(10.01 20.03,20.94 21.34,35.93 19.04)',
db2gse.coordref()..srid(0)))
```

ST_LineFromWKB

ST_LineFromWKB 接受线条类型的公认二进制表示和 Spatial 参考系标识，并返回线条。

语法

db2gse.ST_LineFromWKB(WKBLineString Blob(1M), cr db2gse.coordref)

返回类型

db2gse.ST_LineString

示例

下列代码段用每个排水管道的唯一 ID、大小类别和几何图形填充 SEWERLINES 表。

创建具有三列的 SEWERLINES 表。第一列 SEWER_ID 唯一地标识每条排水管道。 第二列 CLASS 为整数类型，标识排水管道的类型，它通常与管道的能力相关。 第三列 SEWER 为线条类型，存储排水管道的几何图形。

```
CREATE TABLE SEWERLINES (sewer_id   integer,
                         class  integer,
                         sewer  db2gse.ST_LineString);

/* Create the SQL insert statement to populate the sewer_id, size class
   and the sewer linestring. The question marks are parameter markers that
   indicate the sewer_id, class and sewer geometry values that will be
   retrieved at runtime. */
strcpy (wkb_sql,"insert into sewerlines (sewer_id,class,sewer)
values (?,?, db2gse.ST_LineFromWKB (cast(? as blob(1m)), db2gse.coordref()..srid(0)))");

/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);

/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)wkb_sql, SQL_NTS);

/* Bind the integer sewer_id value to the first parameter. */
pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_SLONG,
   SQL_INTEGER, 0, 0, &sewer_id, 0, &pcbvalue1);

/* Bind the integer class value to the second parameter. */
pcbvalue2 = 0;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_SLONG,
   SQL_INTEGER, 0, 0, &sewer_class, 0, &pcbvalue2);

/* Bind the shape to the third parameter. */
pcbvalue3 = blob_len;
rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_BINARY,
```

```
     SQL_BLOB, blob_len, 0, sewer_wkb, blob_len, &pcbvalue3);

/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

ST_MLineFromText

ST_MLineFromText 接受多线条类型的公认文本表示和 Spatial 参考系标识，并返回多线条。

语法

db2gse.ST_MLineFromText(multiLineStringTaggedText String, cr db2gse.coordref)

返回类型

db2gse.ST_MultiLineString

示例

以下 CREATE TABLE 语句创建 MLINESTRING_TEST 表。MLINESTRING_TEST 具有两列: GID 小整数列（唯一地标识行）和 ML1 多线条列。

```
CREATE TABLE ST_MLINESTRING_TEST (gid smallint, ml1 db2gse.ST_MultiLineString)
```

以下 INSERT 语句用 ST_MLineFromText 函数插入多线条。

```
INSERT INTO MLINESTRING_TEST
VALUES (1, db2gse.ST_MLineFromText('multilinestring((10.01 20.03,10.52 40.11,30.29 41.56,
                                        31.78 10.74),
                                       (20.93 20.81, 21.52 40.10))',
                          db2gse.coordref()..srid(0)))
```

ST_MLineFromWKB

ST_MLineFromWKB 接受多线条类型的公认二进制表示和 Spatial 参考系标识，并返回多线条。

语法

db2gse.ST_MLineFromWKB(WKBMultiLineString Blob(1M), cr db2gse.coordref)

返回类型

db2gse.ST_MultiLineString

示例

下列代码段用唯一的 ID、名称和水多线条填充 WATERWAYS 表。

创建 WATERWAYS 表，该表具有 ID 和 NAME 列，它们标识存储在该表中的每个溪流和江河系统。WATER 列为多线条，因为江河和溪流系统通常是几个线条的集合。

```
CREATE TABLE WATERWAYS (id        integer,
                        name      varchar(128),
                        water     db2gse.ST_MultiLineString);

/* Create the SQL insert statement to populate the id, name and
   multilinestring. The question marks are parameter markers that
   indicate the id, name and water values that will be retrieved at
   runtime. */
strcpy (shp_sql,"insert into WATERWAYS (id,name,water)
values (?,?, db2gse.ST_MLineFromWKB (cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");

/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);

/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)shp_sql, SQL_NTS);

/* Bind the integer id value to the first parameter. */
pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_SLONG,
   SQL_INTEGER, 0, 0, &id, 0, &pcbvalue1);

/* Bind the varchar name value to the second parameter. */
pcbvalue2 = name_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
     SQL_CHAR, name_len, 0, &name, name_len, &pcbvalue2);

/* Bind the shape to the third parameter. */
pcbvalue3 = blob_len;
rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_BINARY,
```

```
        SQL_BLOB, blob_len, 0, water_shape, blob_len, &pcbvalue3);

    /* Execute the insert statement. */
    rc = SQLExecute (hstmt);
```

ST_MPointFromText

ST_MPointFromText 接受多个点类型的公认文本表示和 Spatial 参考系标识，并返回多个点。

语法

db2gse.ST_MPointFromText(multiPointTaggedText Varchar(4000), cr db2gse.coordref)

返回类型

db2gse.ST_MultiPoint

示例

以下 CREATE TABLE 语句创建具有一个多点列 MPT1 的 MULTIPOINT_TEST 表。

```
CREATE TABLE MULTIPOINT_TEST (mpt1 db2gse.ST_MultiPoint)
```

以下 INSERT 语句通过使用 ST_MPointFromText 列将多个点插入 MPT1 列。

```
INSERT INTO MULTIPOINT_TEST
VALUES (1, db2gse.ST_MPointFromText('multipoint(10.01 20.03,10.52 40.11,
                                    30.29 41.56,31.78 10.74)',
                          db2gse.coordref()..srid(0)))
```

ST_MPointFromWKB

ST_MPointFromWKB 接受多个点类型的公认二进制表示和 Spatial 参考系标识，以返回多个点。

语法

db2gse.ST_MPointFromWKB(WKBMultiPoint Blob(1M), cr db2gse.coordref)

返回类型

db2gse.ST_MultiPoint

示例

以下代码段填充 SPECIES_SITINGS 表。

创建具有三列的 SPECIES_SITINGS 表。SPECIES 和 GENUS 列唯一地标识每行，而 SITINGS 多点列则存储物种所在的位置。

```
CREATE TABLE SPECIES_SITINGS (species  varchar(32),
                              genus varchar(32),
                              sitings  db2gse.ST_MultiPoint);

/* Create the SQL insert statement to populate the species, genus and
   sitings. The question marks are parameter markers that
   indicate the species, genus and sitings values that will be retrieved at
   runtime. */
strcpy (wkb_sql,"insert into SPECIES_SITINGS (species,genus,sitings)
values (?,?, db2gse.ST_MPointFromWKB (cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");

/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);

/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)wkb_sql, SQL_NTS);

/* Bind the varchar species value to the first parameter. */
pcbvalue1 = species_len;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
   SQL_CHAR, species_len, 0, &species, species_len, &pcbvalue1);
/* Bind the varchar genus value to the second parameter. */
pcbvalue2 = genus_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
   SQL_CHAR, genus_len, 0, &name, genus_len, &pcbvalue2);

/* Bind the shape to the third parameter. */
pcbvalue3 = sitings_len;
rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_BINARY,
```

```
     SQL_BLOB, sitings_len, 0, sitings_wkb, sitings_len, &pcbvalue3);

/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

ST_MPolyFromText

ST_MPolyFromText 接受复合多边形类型的公认文本表示和 Spatial 参考系标识，并返回复合多边形。

语法

db2gse.ST_MPolyFromText(multiPolygonTaggedText Varchar(4000), cr db2gse.coordref)

返回类型

db2gse.ST_MultiPolygon

示例

以下 CREATE TABLE 语句创建 MULTIPOLYGON_TEST 表，它具有单个复合多边形列 MPL1。

```
CREATE TABLE MULTIPOLYGON_TEST (mpl1 db2gse.ST_MultiPolygon)
```

以下 INSERT 语句通过使用 ST_MPolyFromText 函数将复合多边形插入 MPL1 列。

```
INSERT INTO MULTIPOLYGON_TEST VALUES (
db2gse.ST_MPolyFromText('multipolygon(((10.01 20.03,10.52 40.11,30.29 41.56,31.78
10.74,10.01 20.03),(21.23 15.74,21.34 35.21,28.94 35.35,29.02 16.83,21.23
15.74)),((40.91 10.92,40.56 20.19,50.01 21.12,51.34 9.81,40.91 10.92)))',
db2gse.coordref()..srid(0)))
```

ST_MPolyFromWKB

ST_MPolyFromWKB 接受复合多边形类型的公认二进制表示和 Spatial 参考系标识，并返回复合多边形。

语法

db2gse.ST_MPolyFromWKB(WKBMultiPolygon Blob(1M), cr db2gse.coordref)

返回类型

db2gse.ST_MultiPolygon

示例

以下代码段填充 LOTS 表。

LOTS 表存储 LOT_ID（它唯一地标识每个地块）和 LOT 复合多边形（它包含地块线几何图形）。

```
CREATE TABLE LOTS (  lot_id integer, lot db2gse.ST_MultiPolygon );

/* Create the SQL insert statement to populate the lot_id, and lot. The
   question marks are parameter markers that indicate the lot_id, and lot
   values that will be retrieved at runtime. */
strcpy (wkb_sql,"insert into LOTS (lot_id,lot)
values (?, db2gse.ST_MPolyFromWKB (cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");

/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);

/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)wkb_sql, SQL_NTS);

/* Bind the lot_id integer value to the first parameter. */
pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_INTEGER,
     SQL_INTEGER, 0, 0, &lot_id, 0, &pcbvalue1);

/* Bind the lot shape to the second parameter. */
pcbvalue2 = lot_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
     SQL_BLOB, lot_len, 0, lot_wkb, lot_len, &pcbvalue2);

/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

ST_NumGeometries

ST_NumGeometries 接受集合并返回集合中的几何图形数。

语法

db2gse.ST_NumGeometries(g db2gse.ST_GeomCollection)

返回类型

整数

示例

城市工程师需要知道与每个建筑物占地形状相关的不同建筑物的实际数目。

建筑物占地形状存储在用以下 CREATE TABLE 语句创建的 BUILDINGFOOTPRINTS 表中。

```
CREATE TABLE BUILDINGFOOTPRINTS (   building_id integer,
                                    lot_id      integer,
                                    footprint   db2gse.ST_MultiPolygon);
```

以下 SELECT 语句使用 ST_NumGeometries 函数列示唯一地标识每个建筑物的 BUILDING_ID 和每个占地形状中的建筑物数。

```
SELECT building_id, db2gse.ST_NumGeometries (footprint) "Number of buildings"
FROM BUILDINGFOOTPRINTS;
```

ST_NumInteriorRing

ST_NumInteriorRing 接受多边形并返回该多边形的内环数。

语法

db2gse.NumInteriorRing(p db2gse.ST_Polygon)

返回类型

整数

示例

有一个鸟类学家，希望研究几个南海岛屿上的鸟群。她知道特定鸟类的进食区域限于包含淡水湖的海岛。因此，她想要知道哪些海岛包含一个或多个湖泊。

以下 CREATE TABLE 语句创建 ISLANDS 表。ISLANDS 表的 ID 和 NAME 列标识每个海岛，而 LAND 多边形列存储海岛的几何图形。

```
CREATE TABLE ISLANDS (id integer, name varchar(32), land db2gse.ST_Polygon);
```

因为内环表示湖泊，所以使用 ST_NumInteriorRing 函数只列示至少具有一个内环的那些海岛。

```
SELECT name FROM ISLANDS WHERE db2gse.ST_NumInteriorRing(land) > 0;
```

ST_NumPoints

ST_NumPoints 接受线条并返回它的点数。

语法

db2gse.ST_NumPoints(l db2gse.ST_LineString)

返回类型

整数

示例

以下 CREATE TABLE 语句创建 NUMPOINTS_TEST 表。 GEOTYPE 列包含 G1 几何图形列中存储的几何图形类型。

```
CREATE TABLE NUMPOINTS_TEST (geotype varchar(12), g1 db2gse.ST_Geometry)
```

以下 INSERT 语句插入一条线条。

```
INSERT INTO NUMPOINTS_TEST VALUES( linestring,
db2gse.ST_LineFromText('linestring (10.02 20.01, 23.73 21.92)',
db2gse.coordref()..srid(0)))
```

以下 SELECT 语句和相应的结果集列示几何图形类型和每个几何图形类型内包含的点数。

```
SELECT geotype, db2gse.ST_NumPoints(g1)
FROM NUMPOINTS_TEST

GEOTYPE      Number of points
------------ ----------------
ST_linestring          2
  1 record(s) selected.
```

ST_OrderingEquals

ST_OrderingEquals 比较两个几何图形，若这两个几何图形相等且坐标次序相同，则返回 1 (TRUE)；否则，它返回 0 (FALSE)。

语法

db2gse.ST_OrderingEquals(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

返回类型

整数

示例

以下 CREATE TABLE 语句创建 LINESTRING_TEST 表，它具有两个线条列 L1 和 L2。

```
CREATE TABLE LINESTRING_TEST (lid integer, l1 db2gse.ST_LineString,
l2 db2gse.ST_LineString);
```

以下 INSERT 语句将两个相等且具有相同坐标次序的线条插入 L1 和 L2。

```
INSERT INTO linestring_test VALUES (1,
db2gse.LineFromText('linestring (10.01 20.02, 21.50 12.10)', db2gse.coordref()..srid(0)),
db2gse.LineFromText('linestring (10.01 20.02, 21.50 12.10)',
                        db2gse.coordref()..srid(0)));
```

以下 INSERT 语句将两个相等但坐标次序不相同的线条插入 L1 和 L2。

```
INSERT INTO linestring_test VALUES (2,
db2gse.LineFromText('linestring (10.01 20.02, 21.50 12.10)', db2gse.coordref()..srid(0)),
db2gse.LineFromText('linestring (21.50 12.10,10.01 20.02)', db2gse.coordref()..srid(0)));
```

以下 SELECT 语句和相应结果集显示，不管坐标次序如何，ST_Equals 函数都返回 1 (TRUE)。若两个几何图形既不相等又不具有相同坐标次序，则 ST_OrderingEquals 函数返回 0 (FALSE)。

```
SELECT lid, db2gse.ST_Equals(l1,l2) equals, db2gse.ST_OrderingEquals(l1,l2)
OrderingEquals
FROM linestring_test

lid   equals   OrderingEquals
---   ------   -----------
 1       1           1
 2       1           0
```

ST_Overlaps

ST_Overlaps 接受两个几何图形对象，若两个对象的交集产生相同维数的几何图形对象且该对象不等于任一个源对象，则返回 1 (TRUE)；否则，它返回 0 (FALSE)。

语法

db2gse.ST_Overlaps(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

返回类型

整数

示例

县级主管人员需要一个有害废物地点的列表，这些地点的五英里半径覆盖了一些敏感区域。

以下 CREATE TABLE 语句创建 SENSITIVE_AREAS 表。除了包含 ZONE 列（用于存储公共设施的多边形几何图形）之外，SENSITIVE_AREAS 表还包含描述受威胁的公共设施的几列。

```
CREATE TABLE SENSITIVE_AREAS (id        integer,
                              name      varchar(128),
                              size      float,
                              type      varchar(10),
                              zone      db2gse.ST_Polygon);
```

HAZARDOUS_SITES 表将地点的标识存储在 SITE_ID 和 NAME 列中，而将每个地点的实际地理位置存储在 LOCATION 点列中。

```
CREATE TABLE HAZARDOUS_SITES (site_id   integer,
                              name      varchar(128),
                              location  db2gse.ST_Point);
```

在以下 SELECT 语句中，通过 ST_Overlaps 函数将 SENSITIVE_AREAS 和 HAZARDOUS_SITES 表连接在一起。对于其区域多边形与 HAZARDOUS_SITES 位置点的缓冲五英里半径重叠的 SENSITIVE_AREAS 表的所有行，它返回 1 (TRUE)。

```
SELECT hs.name
FROM HAZARDOUS_SITES hs, SENSITIVE_AREAS sa
WHERE db2gse.ST_Overlaps (buffer(hs.location,(5 * 5280)),sa.zone) = 1;
```

在第235页的图37中，医院和学校与该县的两个有害废物地点的五英里半径重叠，而私人疗养院则不重叠。

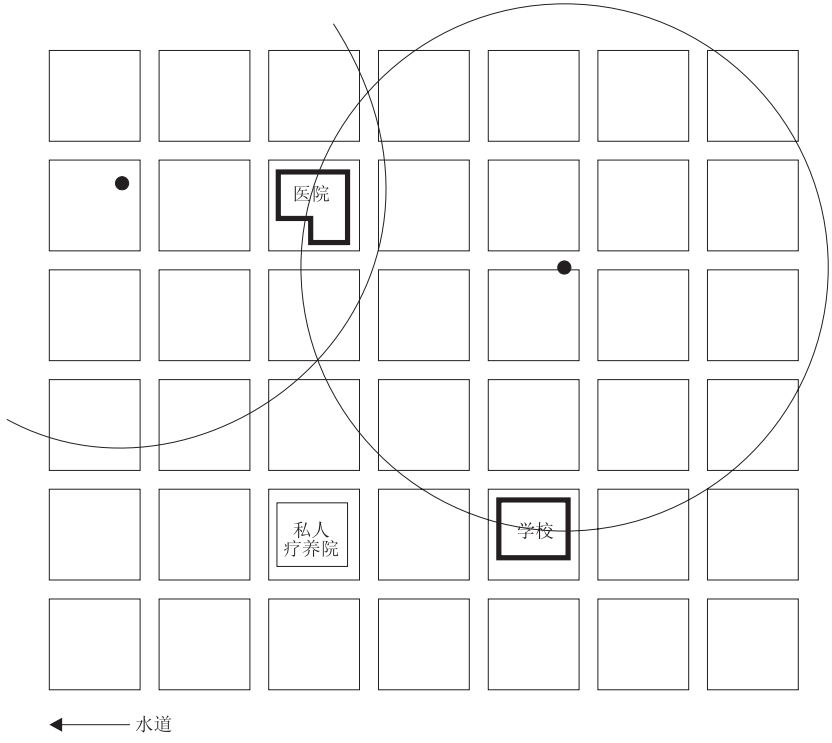


图 37. 使用 ST_Overlaps 确定至少有一部分在有害废物区域内的建筑物

ST_Perimeter

ST_Perimeter 返回 ST_Surface 的周长。

语法

db2gse.ST_Perimeter(s db2gse.ST_Surface)
db2gse.ST_Perimeter(ms db2gse.ST_MultiSurface)

返回类型

双精度

示例

研究湖岸线鸟类的生态学家需要确定特定区域内的湖的湖岸线。将湖泊以复合多边形的形式存储在用以下 CREATE TABLE 语句创建的 WATERBODIES 表中。

```
CREATE TABLE WATERBODIES (wbid  integer,
                          waterbody  db2gse.ST_MultiPolygon);
```

在以下 SELECT 语句中，ST_Perimeter 函数返回围绕每个水体的周长，而 SUM 函数汇总这些周长并返回它们的总和。

```
SELECT SUM(db2gse.ST_Perimeter(waterbody))
FROM waterbodies;
```

ST_PointFromText

ST_PointFromText 接受点类型的公认文本表示和 Spatial 参考系标识，并返回点。

语法

db2gse.ST_PointFromText(pointTaggedText Varchar(4000), cr db2gse.coordref)

返回类型

db2gse.ST_Point

示例

以下 CREATE TABLE 语句创建 POINT_TEST 表，该表具有单个点列 PT1。

```
CREATE TABLE POINT_TEST (pt1 db2gse.ST_Point)
```

在 INSERT 语句将点插入 PT1 列之前，ST_PointFromText 函数将点文本坐标转换为点格式。

```
INSERT INTO POINT_TEST VALUES (
       db2gse.ST_PointFromText ('point(10.01 20.03)', db2gse.coordref()..srid(0)))
```

ST_PointFromWKB

ST_PointFromWKB 接受点类型的公认二进制表示和 Spatial 参考系标识，并返回点。

语法

db2gse.ST_PointFromWKB(WKBPoint Blob(1M), srs SRID)

返回类型

db2gse.ST_Point

示例

以下代码段填充 HAZARDOUS_SITES 表。

危险地点存储在用以下 CREATE TABLE 语句创建的 HAZARDOUS_SITES 表中。定义为点的 LOCATION 列存储每个危险地点的地理中心的位置。

```
CREATE TABLE HAZARDOUS_SITES (site_id   integer,
                              name      varchar(128),
                              location  db2gse.ST_Point);

/* Create the SQL insert statement to populate the site_id, name and
   location. The question marks are parameter markers that indicate the
   site_id, name and location values that will be retrieved at runtime. */
strcpy (wkb_sql,"insert into HAZARDOUS_SITES (site_id, name, location)
values (?,?, db2gse.ST_PointFromWKB(cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");

/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);

/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)wkb_sql, SQL_NTS);

/* Bind the site_id integer value to the first parameter. */
pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_INTEGER,
     SQL_INTEGER, 0, 0, &site_id, 0, &pcbvalue1);

/* Bind the name varchar value to the second parameter. */
pcbvalue2 = name_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
   SQL_CHAR, 0, 0, name, 0, &pcbvalue2);

/* Bind the location shape to the third parameter. */
pcbvalue3 = location_len;
rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_BINARY,
```

```
     SQL_BLOB, location_len, 0, location_wkb, location_len, &pcbvalue3);

/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

ST_Point

ST_Point 返回 ST_Point，并给出 x 坐标、y 坐标和 Spatial 参考系。

语法

db2gse.ST_Point(X Double, Y Double, srs SRID)

返回类型

db2gse.ST_Point

示例

以下 CREATE TABLE 语句创建 POINT_TEST 表，该表具有单个点列 PT1。

```
CREATE TABLE POINT_TEST (pt1 db2gse.ST_Point)
```

在 INSERT 语句将点坐标插入 PT1 列之前，ST_Point 函数将点坐标转换为点几何图形。

```
INSERT INTO point_test VALUES(
    db2gse.ST_Point(10.01,20.03, db2gse.coordref()..srid(0))
)
```

ST_PointN

ST_PointN 接受线条和整数索引，并返回线条路径中第 n 个顶点的点。

语法

db2gse.ST_PointN(l db2gse.ST_Curve, n Integer)

返回类型

db2gse.ST_Point

示例

以下 CREATE TABLE 语句创建 POINTN_TEST 表，该表具有两列: GID 列（唯一地标识每行）和 LN1 线条列。

```
CREATE TABLE POINTN_TEST (gid integer, ln1 db2gse.ST_LineString)
```

下列 INSERT 语句插入两个线条值。第一条线条没有 Z 坐标或度量单位，而第二个线条则两者都有。

```
INSERT INTO POINTN_TEST VALUES(1,
db2gse.ST_LineFromText('linestring (10.02 20.01,23.73 21.92,30.10 40.23)',
db2gse.coordref()..srid(0)))

INSERT INTO POINTN_TEST VALUES(2,
db2gse.ST_LineFromText('linestring  zm (10.02 20.01 5.0 7.0,23.73 21.92 6.5 7.1,30.10
40.23 6.9 7.2)', db2gse.coordref()..srid(0)))
```

以下 SELECT 语句和相应的结果集列示 GID 列和每个线条的第二个顶点。第一行产生没有 Z 坐标或度量单位的点，而第二行产生具有 Z 坐标和度量单位的点。ST_PointN 函数返回具有 Z 坐标或度量单位的点（若它们在源线条中存在的话）。

```
SELECT gid, CAST(db2gse.ST_AsText(db2gse.ST_PointN(ln1,2)) AS varchar(60))
"The 2nd vertice"
FROM POINTN_TEST

GID         The 2nd vertice
----------- ------------------------------------------------------------
          1 POINT ( 23.73000000 21.92000000)
          2 POINT ZM ( 23.73000000 21.92000000 7.00000000 7.10000000)

  2 record(s) selected.
```

ST_PointOnSurface

ST_PointOnSurface 既接受多边形也接受复合多边形，并返回 ST_Point。

语法

db2gse.ST_PointOnSurface(s db2gse.ST_Surface)
db2gse.ST_PointOnSurface(ms db2gse.ST_MultiSurface)

返回类型

db2gse.ST_Point

示例

城市工程师需要为每个建筑物占地形状创建标记点。

建筑物占地形状存储在用以下 CREATE TABLE 语句创建的 BUILDINGFOOTPRINTS 表中。

```
CREATE TABLE BUILDINGFOOTPRINTS (   building_id integer,
                                    lot_id      integer,
                                    footprint   db2gse.ST_MultiPolygon);
```

ST_PointOnSurface 函数生成保证在建筑物占地形状的表面上的点。ST_PointOnSurface 函数返回一个点，AsBinaryShape 函数将它转换为形状，再将该形状转换为 1 MB 的字符串供应用程序使用。

```
SELECT CAST(db2gse.AsBinaryShape(db2gse.ST_PointOnSurface(footprint)) as blob(1m))
FROM BUILDINGFOOTPRINTS;
```

ST_PolyFromText

ST_PolyFromText 接受多边形类型的公认文本表示和 Spatial 参考系标识，并返回多边形。

语法

db2gse.ST_PolyFromText(polygonTaggedText Varchar(4000), cr db2gse.coordref)

返回类型

db2gse.ST_Polygon

示例

以下 CREATE TABLE 语句创建具有单个多边形列的 POLYGON_TEST 表。

```
CREATE TABLE POLYGON_TEST (pl1 db2gse.ST_Polygon)
```

以下 INSERT 语句通过使用 ST_PolyFromText 函数将多边形插入多边形列。

```
INSERT INTO POLYGON_TEST VALUES (1,
db2gse.ST_PolyFromText('polygon((10.01 20.03,10.52 40.11,30.29 41.56,31.78 10.74,10.01
20.03))', db2gse.coordref()..srid(0)))
```

ST_PolyFromWKB

ST_PolyFromWKB 接受多边形类型的公认二进制表示和 Spatial 参考系标识，以返回多边形。

语法

db2gse.ST_PolyFromWKB(WKBPolygon Blob(1M), SRID Integer)

返回类型

db2gse.ST_Polygon

示例

以下代码段填充 SENSITIVE_AREAS 表。

除了包含 zone 列（用于存储公共设施的多边形几何图形）之外，SENSITIVE_AREAS 表还包含描述受威胁的公共设施的几列。

```
CREATE TABLE SENSITIVE_AREAS (id        integer,
                              name      varchar(128),
                              size      float,
                              type      varchar(10),
                              zone      db2gse.ST_Polygon);


/* Create the SQL insert statement to populate the id, name, size, type and
   zone. The question marks are parameter markers that indicate the id,name,
   size, type and zone values that will be retrieved at runtime. */
strcpy (shp_wkb,"insert into SENSITIVE_AREAS (id, name, size, type, zone)
values (?,?,?,?, db2gse.ST_PolyFromWKB (cast(? as blob(1m)),
db2gse.coordref()..srid(0)))");

/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */
rc = SQLAllocStmt (handle, &hstmt);

/* Prepare the SQL statement for execution. */
rc = SQLPrepare (hstmt, (unsigned char *)wkb_sql, SQL_NTS);

/* Bind the id integer value to the first parameter. */
pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_INTEGER,
   SQL_INTEGER, 0, 0, &id, 0, &pcbvalue1);

/* Bind the name varchar value to the second parameter. */
pcbvalue2 = name_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
   SQL_CHAR, 0, 0, name, 0, &pcbvalue2);

/* Bind the size float to the third parameter. */
pcbvalue3 = 0;
```

```
rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_FLOAT,
     SQL_REAL, 0, 0, &size, 0, &pcbvalue3);

/* Bind the type varchar to the fourth parameter. */
pcbvalue4 = type_len;
rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR,
     SQL_VARCHAR, type_len, 0, type, type_len, &pcbvalue4);

/* Bind the zone polygon to the fifth parameter. */
pcbvalue5 = zone_len;
rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_BINARY,
     SQL_BLOB, zone_len, 0, zone_wkb, zone_len, &pcbvalue5);

/* Execute the insert statement. */
rc = SQLExecute (hstmt);
```

ST_Polygon

ST_Polygon 从 ST_LineString 和 Spatial 参考系生成 ST_Polygon。

语法

db2gse.ST_Polygon(l db2gse.ST_LineString, cr db2gse.coordref)

返回类型

db2gse.ST_Polygon

示例

以下 CREATE TABLE 语句创建 POLYGON_TEST 表，该表具有单个列 P1。

```
CREATE TABLE POLYGON_TEST (p1 db2gse.ST_polygon)
```

以下 INSERT 语句通过在 ST_Polygon 函数内使用 ST_LineFromText 函数，将环（封闭且简单的线条）转换为多边形并将它插入 P1 列。

```
INSERT INTO POLYGON_TEST VALUES (
db2gse.ST_Polygon(db2gse.ST_LineFromText('linestring(10.01 20.03,20.94
21.34,35.93 10.04,10.01 20.03)', db2gse.coordref()..srid(0))),
db2gse.coordref()..srid(0)))
)
```

ST_Relate

ST_Relate 比较两个几何图形，若这两个几何图形满足 DE-9IM 模式矩阵字符串指定的条件，则返回 1 (TRUE)；否则，返回 0 (FALSE)。有关 DE-9IM 模式矩阵的信息，参见第122页的『谓词函数』。

语法

db2gse.ST_Relate(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry, patternMatrix String)

返回类型

整数

示例

DE-9IM 模式矩阵是用于比较几何图形的设备。这样的矩阵有几种类型。例如，等于模式矩阵将告知任何两个几何图形是否相等。

在此示例中，将表56中显示的等于模式矩阵从左至右从上至下读入某个字符串（『T*F**FFF*』）。

表 56. 等于模式矩阵

		b		
		内部	**边界**	**外部**
a	**内部**	T	*	F
	边界	*	*	F
	外部	F	F	*

下一步，用以下 CREATE TABLE 语句创建表 RELATE_TEST。

```
CREATE TABLE RELATE_TEST (rid integer, g1 db2gse.ST_Geometry,
g2 db2gse.ST_Geometry, g3 db2gse.ST_Geometry);
```

下列 INSERT 语句将样本子类插入 RELATE_TEST 表。

```
INSERT INTO RELATE_TEST VALUES(
       1,
        db2gse.ST_PointFromText('point (10.02 20.01)',db2gse.coordref()..srid(0),
        db2gse.ST_PointFromText('point (10.02 20.01)',db2gse.coordref()..srid(0),
        db2gse.ST_PointFromText('point (30.01 20.01)',db2gse.coordref()..srid(0)
)
```

以下 SELECT 语句和相应的结果集列示 GEOTYPE 列中存储的子类名与该 geotype 的维数。

```
SELECT rid, relate(g1,g2) equals, relate(g1,g3) not_equals
FROM relate_test
```

```
RID     equals    not_equals
------- -----------  -----------------
1         1              0

  1 record(s) selected.
```

ST_SRID

ST_SRID 接受几何图形对象并返回它的 Spatial 参考系标识。

语法

db2gse.ST_SRID(g1 db2gse.ST_Geometry)

返回类型

整数

示例

在 DB2 Spatial Extender 的安装期间，创建 SPATIAL_REFERENCES 表。当创建几何图形时，会将该几何图形的 SRID 输入 SPATIAL_REFERENCES 表。ST_SRID 函数返回该项的值。

例如，在 CREATE TABLE 语句中使用几何图形类型:

```
CREATE TABLE SRID_TEST(g1 db2gse.ST_Geometry)
```

在以下 INSERT 语句中，将位于坐标 10.01,50.76 的点几何图形插入几何图形列 G1。当 ST_PointFromText 函数创建该点几何图形时，给它指定了srid 值 1。

```
INSERT INTO SRID_TEST
VALUES (db2gse.ST_PointFromText('point(10.01 50.76)', db2gse.coordref()..srid(0)))
```

ST_SRID 函数返回刚才输入的几何图形的 Spatial 参考系标识，如以下的 SELECT 语句和相应结果集所示。

```
SELECT db2gse.ST_SRID(g1) FROM SRID_TEST

g1
--------------
1
```

ST_StartPoint

ST_StartPoint 接受线条并返回该线条的第一点。

语法

db2gse.ST_StartPoint(c db2gse.ST_Curve)

返回类型

db2gse.ST_Point

示例

以下 CREATE TABLE 语句创建 STARTPOINT_TEST 表。 STARTPOINT_TEST 具有两列: GID 整数列（唯一地标识表的行）和 LN1 线条列。

```
CREATE TABLE STARTPOINT_TEST (gid integer, ln1 db2gse.ST_LineString)
```

下列 INSERT 语句将线条插入 LN1 列。第一条线条没有 Z 坐标或度量单位，而第二个线条则两者都有。

```
INSERT INTO STARTPOINT_TEST VALUES(1,
db2gse.ST_LineFromText('linestring (10.02 20.01,23.73
21.92,30.10 40.23)', db2gse.coordref()..srid(0)))

INSERT INTO STARTPOINT_TEST VALUES(2,
db2gse.ST_LineFromText('linestring  zm (10.02 20.01 5.0 7.0,23.73 21.92 6.5 7.1,30.10
40.23 6.9 7.2)', db2gse.coordref()..srid(0)))
```

以下 SELECT 语句和相应结果集显示 ST_StartPoint 函数如何抽出每个线条的第一点。 ST_AsText 函数将该点转换为它的文本格式。列表中的第一点没有 Z 坐标或度量单位，而第二点则二者都具有，因为源线条具有。

```
SELECT gid, CAST(db2gse.ST_AsText(db2gse.ST_StartPoint (ln1)) as varchar(60))
"Startpoint"
FROM STARTPOINT_TEST

GID         Startpoint
----------- ------------------------------------------------------------
          1 POINT ( 10.02000000 20.01000000)
          2 POINT ZM ( 10.02000000 20.01000000 5.00000000 7.00000000)

  2 record(s) selected.
```

ST_SymmetricDiff

ST_SymmetricDiff 接受两个几何图形对象，并返回是源对象的对称差异的几何图形对象。

语法

db2gse.ST_SymmetricDiff(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

返回类型

db2gse.ST_Geometry

示例

县级主管人员必须确定敏感区域和五英里危险地点半径不相交的区域。

以下 CREATE TABLE 语句创建 SENSITIVE_AREAS 表，该表包含描述受威胁的公共设施的几列。 SENSITIVE_AREAS 表还包含 ZONE 列，它存储公共设施的多边形几何图形。

```
CREATE TABLE SENSITIVE_AREAS (id        integer,
                              name      varchar(128),
                              size      float,
                              type      varchar(10),
                              zone      db2gse.ST_Polygon);
```

以下 CREATE TABLE 语句创建 HAZARDOUS_SITES 表，它将地点的标识存储在 SITE_ID 和 NAME 列中，而将每个地点的实际地理位置存储在 LOCATION 点列中。

```
CREATE TABLE HAZARDOUS_SITES (site_id   integer,
                              name      varchar(128),
                              location  point);
```

ST_Buffer 函数生成环绕有害废物地点位置的五英里缓冲区。 ST_SymmetricDiff 函数从缓冲的有害废物地点多边形和敏感区域的交集生成多边形。 ST_Area 函数对每个危险地点返回交集多边形的区域。

```
SELECT sa.name, hs.name,
       db2gse.ST_Area(db2gse.ST_SymmetricDiff (db2gse.ST_Buffer(hs.location,
(5 * 5280)),sa.zone))
FROM HAZARDOUS_SITES hs, SENSITIVE_AREAS sa
```

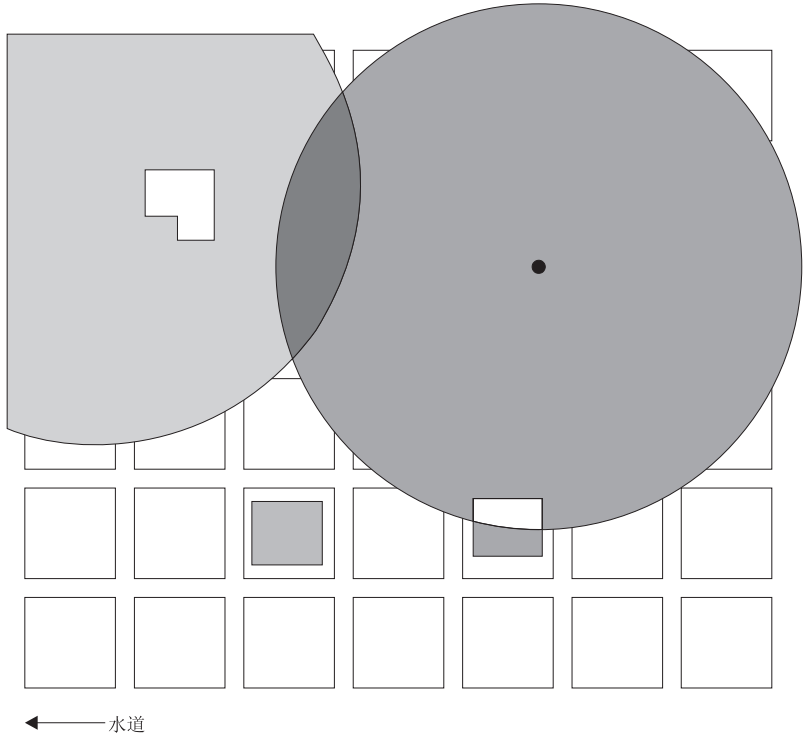


图 38. 使用 ST_SymmetricDiff 确定不包含敏感区域（居住建筑物）的有害废物区域

在图38中，有害废物地点和敏感区域的对称差异导致相交区域减小。

ST_Touches

若两个几何图形的任何公共点都不与两个几何图形的内部相交，则 ST_Touches返回 1 (TRUE)；否则，它返回 0 (FALSE)。至少一个几何图形必须为线条、多边形、多线条或复合多边形。

语法

db2gse.ST_Touches(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

返回类型

整数

示例

GIS 技术人员需要提供其端点与另一排水管道相交的所有排水管道的列表。

以下 CREATE TABLE 语句创建 SEWERLINES 表，该表具有三列。第一列 SEWER_ID 唯一地标识每条排水管道。第二列 CLASS 为整数类型，标识排水管道的类型，它通常与管道的能力相关。第三列 SEWER 为线条类型，存储排水管道的几何图形。

```
CREATE TABLE SEWERLINES (sewer_id integer, class integer, sewer db2gse.ST_LineString);
```

以下 SELECT 语句返回彼比接触的 SEWER_IDS 的有序列表。

```
SELECT s1.sewer_id, s2.sewer_id
FROM sewerlines s1, sewerlines s2
WHERE db2gse.ST_Touches (s1.sewer, s2.sewer) = 1,
ORDER BY 1,2;
```

ST_Transform

ST_Transform 将几何图形指定至一个 Spatial 参考系，该 Spatial 参考系不是该几何图形当前被指定至的 Spatial 参考系。

语法

db2gse.ST_Transform(g db2gse.ST_Geometry, cr db2gse.coordref)

返回类型

db2gse.ST_Geometry

示例

以下 CREATE TABLE 语句创建 TRANSFORM_TEST 表，该表具有两个线条列 L1 和 L2。

```
CREATE TABLE TRANSFORM_TEST (tid integer, l1 db2gse.ST_LineString,
l2 db2gse.ST_LineString)
```

以下 INSERT 语句将线条插入 SRID 为 102 的 l1 中。

```
INSERT INTO TRANSFORM_TEST VALUES (1, db2gse.ST_LineFromText('linestring(10.01 40.43,
92.32 29.89)', db2gse.coordref()..srid(102)),NULL)
```

ST_Transform 函数将 L1 的线条从指定为 SRID 102 的坐标参考转换为指定为 SRID 105 的坐标参考。以下 UPDATE 语句将转换后的线条存储在列 l2 中。

```
UPDATE TRANSFORM_TEST SET l2 = db2gse.ST_Transform(l1, db2gse.coordref()..srid(105))
```

ST_Union

ST_Union 接受两个几何图形对象，并返回是源对象的并集的几何图形对象。

语法

db2gse.ST_Union(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

返回类型

db2gse.ST_Geometry

示例

以下 CREATE TABLE 语句创建 SENSITIVE_AREAS 表，该表包含描述受威胁的公共设施的几列。 SENSITIVE_AREAS 表还包含 ZONE 列，它存储公共设施的多边形几何图形。

```
CREATE TABLE SENSITIVE_AREAS (id        integer,
                              name      varchar(128),
                              size      float,
                              type      varchar(10),
                              zone      db2gse.ST_Polygon);
```

以下 CREATE TABLE 语句创建 HAZARDOUS_SITES 表，该表将地点的标识存储在 SITE_ID 和 NAME 列中。每个地点的实际地理位置存储在 LOCATION 点列中。

```
CREATE TABLE HAZARDOUS_SITES (site_id integer, name varchar(128),
                              location  db2gse.ST_Point);
```

以下 SELECT 语句使用 ST_Buffer 函数生成环绕有害废物地点位置的五英里缓冲区。 ST_Union 函数从缓冲的有害废物地点多边形和敏感区域的并集生成多边形。ST_Area 函数返回多边形区域的并集。

```
SELECT sa.name, hs.name,
       db2gse.ST_Area(db2gse.ST_Union(db2gse.ST_Buffer(hs.location,
(5 * 5280)),sa.zone))
FROM HAZARDOUS_SITES hs, SENSITIVE_AREAS sa;
```

ST_Within

ST_Within 接受两个几何图形对象，若第一个对象完全在第二个对象之内，则返回 1 (TRUE)；否则，它返回 0 (FALSE)。

语法

db2gse.ST_Within(g1 db2gse.ST_Geometry, g2 db2gse.ST_Geometry)

返回类型

整数

示例

在下面的示例中创建两个表。第一个表 BUILDINGFOOTPRINTS 包含城市的建筑物占地形状。第二个表 LOTS 包含城市的地块。城市工程师想要确保所有建筑物占地形状完全在它们的地块内。

在两个表中，复合多边形数据类型存储建筑物占地形状和地块的几何图形。数据库设计者为两种地形选择了复合多边形，因为地块可能被自然地形（比如江河）分开，建筑物占地形状可能经常由几个建筑物组成。

```
CREATE TABLE BUILDINGFOOTPRINTS (   building_id integer,
                                    lot_id      integer,
                                    footprint   db2gse.ST_MultiPolygon);


CREATE TABLE LOTS (  lot_id integer, lot db2gse.ST_MultiPolygon );
```

通过使用以下 SELECT 语句，城市工程师首先选择不完全在一个地块内的建筑物。

```
SELECT building_id
  FROM BUILDINGFOOTPRINTS, LOTS
 WHERE db2gse.ST_Within(footprint,lot) = 0;
```

虽然第一次查询将提供占地形状在地块多边形之外的所有建筑物 ID 的列表，但是它不能确定是否给其余 ID 指定了正确的 lot_id。第二个 SELECT 语句对 BUILDINGFOOTPRINTS 表的 LOT_ID 列执行数据完整性检查。

```
SELECT bf.building_id "building id",
       bf.lot_id "buildings lot_id",
       LOTS.lot_id "LOTS lot_id"
  FROM BUILDINGFOOTPRINTS bf, LOTS
 WHERE db2gse.ST_Within(footprint,lot) = 1 AND
       LOTS.lot_id <> bf.lot_id;
```

ST_WKBToSQL

ST_WKBToSQL 构造 ST_Geometry 值，给出它的公认二进制表示。自动使用值为 0 的 SRID。

语法

db2gse.ST_WKBToSQL(WKBGeometry Blob(1M))

返回类型

db2gse.ST_Geometry

示例

以下 CREATE TABLE 语句创建 LOTS 表，该表具有两列: LOT_ID 列和LOT 复合多边形列，前者唯一地标识每个地块，后者包含每个地块的几何图形。

```
CREATE TABLE lots (lot_id  integer,
                   lot     db2gse.ST_MultiPolygon);
```

以下 C 代码段包含 ODBC 函数，这些函数嵌有将数据插入 LOTS 表的 DB2 Spatial Extender SQL 函数。

ST_WKBToSQL 函数将 WKB 表示转换为 DB2 Spatial Extender 几何图形。整个 INSERT 语句被复制到 wkb_sql 字符串。 INSERT 语句包含参数标记以动态接受 LOT_ID 数据和 LOT 数据。

```
/* Create the SQL insert statement to populate the lot id and the
   lot multipolygon. The question marks are parameter markers that
   indicate the lot_id and lot values that will be retrieved at
   run time. */

strcpy (wkb_sql,"insert into lots (lot_id, lot)
  values(?, db2gse.ST_WKBToSQL(cast(? as blob(1m))))");

/* Allocate memory for the SQL statement handle and associate the
   statement handle with the connection handle. */

rc = SQLAllocStmt (handle, &hstmt);

/* Prepare the SQL statement for execution. */

rc = SQLPrepare (hstmt, (unsigned char *)wkb_sql, SQL_NTS);

/* Bind the integer key value to the first parameter. */

pcbvalue1 = 0;
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_SLONG,
     SQL_INTEGER, 0, 0, &lot_id, 0, &pcbvalue1);

/* Bind the shape to the second parameter. */
```

```
pcbvalue2 = blob_len;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
     SQL_BLOB, blob_len, 0, shape_blob, blob_len, &pcbvalue2);

/* Execute the insert statement. */

rc = SQLExecute (hstmt);
```

ST_WKTToSQL

ST_WKTToSQL 构造 ST_Geometry 值，给出它的公认文本表示。自动使用值为 0 的 SRID。

语法

db2gse.ST_WKTToSQL(geometryTaggedText Varchar(4000))

返回类型

db2gse.ST_Geometry

示例

以下 CREATE TABLE 语句创建 GEOMETRY_TEST 表，该表具有两列：整数类型的 GID 列（唯一地标识每行）和 G1 列（存储几何图形）。

```
CREATE TABLE GEOMETRY_TEST (gid smallint, g1 db2gse.ST_Geometry)
```

下列 INSERT 语句将数据插入 GEOMETRY_TEST 表的 GID 和 G1 列。ST_WKTToSQL 函数将每个几何图形的文本表示转换为它对应的 DB2 Spatial Extender 可例示子类。

```
INSERT INTO GEOMETRY_TEST VALUES(
1, db2gse.ST_WKTToSQL ('point (10.02 20.01)')
)

INSERT INTO GEOMETRY_TEST VALUES(
2, db2gse.ST_WKTToSQL('linestring (10.01 20.01, 10.01 30.01, 10.01 40.01)')
)

INSERT INTO GEOMETRY_TEST VALUES(
3, db2gse.ST_WKTToSQL('polygon ((10.02 20.01, 11.92 35.64, 25.02 34.15, 19.15 33.94,
10.02 20.01))')
)

INSERT INTO GEOMETRY_TEST VALUES(
4, db2gse.ST_WKTToSQL('multipoint (10.02 20.01,10.32 23.98,11.92 25.64)')
)

INSERT INTO GEOMETRY_TEST VALUES(
5, db2gse.ST_WKTToSQL('multilinestring ((10.02 20.01, 10.32 23.98,11.92 25.64),
(9.55 23.75,15.36 30.11))')
)

INSERT INTO GEOMETRY_TEST VALUES(
6, db2gse.ST_WKTToSQL('multipolygon (((10.02 20.01, 11.92 35.64,
25.02 34.15, 19.15 33.94,10.02 20.01)),
((51.71 21.73, 73.36 27.04, 71.52 32.87, 52.43 31.90, 51.71 21.73)))')
)
```

ST_X

ST_X 接受点并返回它的 X 坐标。

语法

ST_X(p ST_Point)

返回类型

双精度

示例

以下 CREATE TABLE 语句创建 X_TEST 表，该表具有两列: GID 列（唯一地标识每行）和 PT1 点列。

```
CREATE TABLE X_TEST (gid integer, pt1 db2gse.ST_Point)
```

下列 INSERT 语句插入两行。一列是没有 Z 坐标或度量单位的点。 另一列具有 Z 坐标和度量单位。

```
INSERT INTO X_TEST VALUES(1,
db2gse.ST_PointFromText('point (10.02 20.01)', db2gse.coordref()..srid(0)))

INSERT INTO X_TEST VALUES(2,
db2gse.ST_PointFromText('point  zm (10.02 20.01 5.0 7.0)', db2gse.coordref()..srid(0)))
```

以下 SELECT 语句和相应的结果集列示 GID 列和点的双精度 X 坐标。

```
SELECT gid, db2gse.ST_X(pt1) "The X coordinate" FROM X_TEST

GID         The X coordinate
----------- ------------------------
          1   +1.00200000000000E+001
          2   +1.00200000000000E+001

  2 record(s) selected.
```

ST_Y

ST_Y 接受点并返回它的 Y 坐标。

语法

db2gse.ST_Y(p db2gse.ST_Point)

返回类型

双精度

示例

以下 CREATE TABLE 语句创建 Y_TEST 表，该表具有两列: GID 列（唯一地标识每行）和 PT1 点列。

```
CREATE TABLE Y_TEST (gid integer, pt1 db2gse.ST_Point)
```

下列 INSERT 语句插入两行。一列是没有 Z 坐标或度量单位的点。另一列具有 Z 坐标和度量单位。

```
INSERT INTO Y_TEST VALUES(1,
db2gse.ST_PointFromText('point (10.02 20.01)', db2gse.coordref()..srid(0)))

INSERT INTO Y_TEST VALUES(2,
db2gse.ST_PointFromText('point  zm (10.02 20.01 5.0 7.0)', db2gse.coordref()..srid(0)))
```

以下 SELECT 语句和相应的结果集列示 GID 列和点的双精度 Y 坐标。

```
SELECT gid, db2gse.ST_Y(pt1) "The Y coordinate" FROM Y_TEST

GID         The Y coordinate
----------- ------------------------
          1   +2.00100000000000E+001
          2   +2.00100000000000E+001

  2 record(s) selected.
```

Z

Z 接受点并返回它的 Z 坐标。

语法

Z(p db2gse.ST_Point)

返回类型

双精度

示例

以下 CREATE TABLE 语句创建 Z_TEST 表，该表具有两列: GID 列（唯一地标识每行）和 PT1 点列。

```
CREATE TABLE Z_TEST (gid integer, pt1 db2gse.ST_Point)
```

下列 INSERT 语句插入两行。一列是没有 Z 坐标或度量单位的点。另一列具有 Z 坐标和度量单位。

```
INSERT INTO Z_TEST VALUES(1,
db2gse.ST_PointFromText('point (10.02 20.01)', db2gse.coordref()..srid(0)))

INSERT INTO Z_TEST VALUES(2,
db2gse.ST_PointFromText('point  zm (10.02 20.01 5.0 7.0)', db2gse.coordref()..srid(0)))
```

以下 SELECT 语句和相应的结果集列示 GID 列和点的双精度 Z 坐标。第一行为空，因为该点没有 Z 坐标。

```
SELECT gid, db2gse.Z(pt1) "The Z coordinate" FROM Z_TEST

GID         The Z coordinate
----------- ------------------------
          1                        -
          2   +5.00000000000000E+000

  2 record(s) selected.
```

第15章 坐标系

本章提供有关用于解释 Spatial 数据的 Spatial 参考系 (SRS) 和坐标值的参考信息。

坐标系的概述

Spatial 参考系的公认文本表示为 Spatial 参考系信息提供标准的文本表示。公认文本表示的定义是按照 POSC/EPSG 坐标系数据模型创建的。

Spatial 参考系是地理（纬线 - 经线）坐标系、投影 (X,Y) 坐标系或地心 (X,Y,Z) 坐标系。该坐标系由几个对象组成。每个对象有一个大写的关键字（例如 DATUM 或 UNIT），后跟用括号括起并用逗号分隔的对象定义参数。某些对象由其他对象组成，因此结果是一个嵌套的结构。

注: 可自由地用标准括号 ( ) 替换方括号 [ ]，应准备读取两种形式的括号。

使用方括号的坐标系的字符串表示的 EBNF（扩展 Backus Naur 形式）定义如下（参见上面有关括号的使用的注意事项）：

```
<coordinate system> = <projected cs> | <geographic cs> | <geocentric cs>
<projected cs> = PROJCS["<name>", <geographic cs>, <projection>, {<parameter>,}*
                 <linear unit>]
<projection> = PROJECTION["<name>"]
<parameter> = PARAMETER["<name>", <value>]
<value> = <number>
```

若数据使用投影坐标，则用 PROJCS 关键字标识数据集的坐标系（若使用地理坐标，则用 GEOGCS 进行标识；若使用地心坐标，则用 GEOCCS 进行标识）。

PROJCS 关键字后跟定义投影坐标系的所有“构件”。任何对象的第一个构件总是相同的。投影坐标系名后跟多个对象：地理坐标系、地图投影、一个或多个参数以及长度单位。所有投影坐标系基于地理坐标系，因此此节首先描述特定于投影坐标系的构件。例如，定义 NAD83 数据上的 UTM 区域 10N:

```
PROJCS["NAD_1983_UTM_Zone_10N",
<geographic cs>,
PROJECTION["Transverse_Mercator"],
PARAMETER["False_Easting",500000.0],
PARAMETER["False_Northing",0.0],
PARAMETER["Central_Meridian",-123.0],
PARAMETER["Scale_Factor",0.9996],
PARAMETER["Latitude_of_Origin",0.0],
UNIT["Meter",1.0]]
```

名称和以下几个对象依次定义地理坐标系对象：数据、本初子午线和角度计量单位。

```
<geographic cs> = GEOGCS["<name>", <datum>, <prime meridian>, <angular unit>]
<datum> = DATUM["<name>", <spheroid>]
<spheroid> = SPHEROID["<name>", <semi-major axis>, <inverse flattening>]
<semi-major axis> = <number>
    (semi-major axis is measured in meters and must be > 0.)
<inverse flattening> = <number>
<prime meridian> = PRIMEM["<name>", <longitude>]
<longitude> = <number>
```

NAD83 上的 UTM 区域 10 的地理坐标系字符串:

```
GEOGCS["GCS_North_American_1983",
DATUM["D_North_American_1983",
SPHEROID["GRS_1980",6378137,298.257222101]],
PRIMEM["Greenwich",0],
UNIT["Degree",0.0174532925199433]]
```

UNIT 对象可表示角度计量单位或长度单位:

```
<angular unit> = <unit>
<linear unit> = <unit>
<unit> = UNIT["<name>", <conversion factor>]
<conversion factor> = <number>
```

转换因子指定每单位的米数（对于长度单位）或弧度数（对于角度单位），且必须大于零。

因此 UTM 区域 10N 的整个字符串表示如下:

```
PROJCS["NAD_1983_UTM_Zone_10N",
GEOGCS["GCS_North_American_1983",
DATUM[ "D_North_American_1983",SPHEROID["GRS_1980",6378137,298.257222101]],
PRIMEM["Greenwich",0],UNIT["Degree",0.0174532925199433]],
```

```
PROJECTION["Transverse_Mercator"],PARAMETER["False_Easting",500000.0],
PARAMETER["False_Northing",0.0],PARAMETER["Central_Meridian",-123.0],
PARAMETER["Scale_Factor",0.9996],PARAMETER["Latitude_of_Origin",0.0],
UNIT["Meter",1.0]]
```

地心坐标系与地理坐标系相当类似:

```
<geocentric cs> = GEOCCS["<name>", <datum>, <prime meridian>, <linear unit>]
```

受支持的长度单位

表 57. 受支持的长度单位

单位	转换因子
Meter	1.0
Foot (International)	0.3048
U.S. Foot	12/39.37
Modified American Foot	12.0004584/39.37
Clarke's Foot	12/39.370432
Indian Foot	12/39.370141
Link	7.92/39.370432
Link (Benoit)	7.92/39.370113
Link (Sears)	7.92/39.370147
Chain (Benoit)	792/39.370113
Chain (Sears)	792/39.370147
Yard (Indian)	36/39.370141
Yard (Sears)	36/39.370147
Fathom	1.8288
Nautical Mile	1852.0

受支持的角度单位

表 58. 受支持的角度单位

单位	转换因子
Radian	1.0
Decimal Degree	p/180
Decimal Minute	(p/180)/60
Decimal Second	(p/180)/36000
Gon	p/200

表 58. 受支持的角度单位 (续)

单位	转换因子
Grad	p/200

受支持的球体

表 59. 受支持的球体

名称	半主轴	扁度的倒数
Airy	6377563.396	299.3249646
Modified Airy	6377340.189	299.3249646
Australian	6378160	298.25
Bessel	6377397.155	299.1528128
Modified Bessel	6377492.018	299.1528128
Bessel (Namibia)	6377483.865	299.1528128
Clarke 1866	6378206.4	294.9786982
Clarke 1866 (Michigan)	6378693.704	294.978684677
Clarke 1880	6378249.145	293.465
Clarke 1880 (Arc)	6378249.145	293.466307656
Clarke 1880 (Benoit)	6378300.79	293.466234571
Clarke 1880 (IGN)	6378249.2	293.46602
Clarke 1880 (RGS)	6378249.145	293.465
Clarke 1880 (SGA)	6378249.2	293.46598
Everest 1830	6377276.345	300.8017
Everest 1975	6377301.243	300.8017
Everest (Sarawak and Sabah)	6377298.556	300.8017
Modified Everest 1948	6377304.063	300.8017
Fischer 1960	6378166	298.3
Fischer 1968	6378150	298.3
Modified Fischer (1960)	6378155	298.3
GEM10C	6378137	298.257222101
GRS 1980	6378137	298.257222101
Hayford 1909	6378388	297.0
Helmert 1906	6378200	298.3
Hough	6378270	297.0
International 1909	6378388	297.0

表 59. 受支持的球体 (续)

名称	半主轴	扁度的倒数
International 1924	6378388	297.0
New International 1967	6378157.5	298.2496
Krasovsky	6378245	298.3
Mercury 1960	6378166	298.3
Modified Mercury 1968	6378150	298.3
NWL9D	6378145	298.25
OSU_86F	6378136.2	298.25722
OSU_91A	6378136.3	298.25722
Plessis 1817	6376523	308.64
South American 1969	6378160	298.25
Southeast Asia	6378155	298.3
Sphere (radius = 1.0)	1	0
Sphere (radius = 6371000 m)	6371000	0
Sphere (radius =6370997 m)	6370997	0
Struve 1860	6378297	294.73
Walbeck	6376896	302.78
War Office	6378300.583	296
WGS 1960	6378165	298.3
WGS 1966	6378145	298.25
WGS 1972	6378135	298.26
WGS 1984	6378137	298.257223563

受支持的大地测量数据

表 60. 受支持的大地测量数据

Adindan	Lisbon
Afgooye	Loma Quintana
Agadez	Lome
Australian Geodetic Datum 1966	Luzon 1911
Australian Geodetic Datum 1984	Mahe 1971
Ain el Abd 1970	Makassar
Amersfoort	Malongo 1987
Aratu	Manoca

表 60. 受支持的大地测量数据 (续)

Arc 1950	Massawa
Arc 1960	Merchich
Ancienne Triangulation Francaise	Militar-Geographische Institute
Barbados	Mhast
Batavia	Minna
Beduaram	Monte Mario
Beijing 1954	M'poraloko
Reseau National Belge 1950	NAD Michigan
Reseau National Belge 1972	North American Datum 1927
Bermuda 1957	North American Datum 1983
Bern 1898	Nahrwan 1967
Bern 1938	Naparima 1972
Bogota	Nord de Guerre
Bukit Rimpah	NGO 1948
Camacupa	Nord Sahara 1959
Campo Inchauspe	NSWC 9Z-2
Cape	Nouvelle Triangulation Francaise
Carthage	New Zealand Geodetic Datum 1949
Chua	OS (SN) 1980
Conakry 1905	OSGB 1936
Corrego Alegre	OSGB 1970 (SN)
Cote d'Ivoire	Padang 1884
Datum 73	Palestine 1923
Deir ez Zor	Pointe Noire
Deutsche Hauptdreiecksnetz	Provisional South American Datum 1956
Douala	Pulkovo 1942
European Datum 1950	Qatar
European Datum 1987	Qatar 1948
Egypt 1907	Qornoq
European Reference System 1989	RT38
Fahud	South American Datum 1969
Gandajika 1970	Sapper Hill 1943
Garoua	Schwarzeck

表 60. 受支持的大地测量数据 (续)

Geocentric Datum of Australia 1994	Segora
Guyane Francaise	Serindung
Herat North	Stockholm 1938
Hito XVIII 1963	Sudan
Hu Tzu	Shan Tananarive 1925
Hungarian Datum 1972	Timbalai 1948
Indian 1954	TM65
Indian 1975	TM75
Indonesian Datum 1974	Tokyo
Jamaica 1875	Trinidad 1903
Jamaica 1969	Trucial Coast 1948
Kalianpur	Voirol 1875
Kandawala	Voirol Unifie 1960
Kertau	WGS 1972
Kuwait Oil Company	WGS 1972 Transit Broadcast Ephemeris
La Canoa	WGS 1984
Lake	Yacare
Leigon	Yoff
Liberia 1964	Zanderij

受支持的本初子午线

表 61. 受支持的本初子午线

位置	坐标
Greenwich	0° 0' 0"
Bern	7° 26' 22.5" E
Bogota	74° 4' 51.3" W
Brussels	4° 22' 4.71" E
Ferro	17° 40' 0" W
Jakarta	106° 48' 27.79" E
Lisbon	9° 7' 54.862" W
Madrid	3° 41' 16.58" W
Paris	2° 20' 14.025" E
Rome	12° 27' 8.4" E

表 61. 受支持的本初子午线 (续)

位置	坐标
Stockholm	18° 3' 29" E

受支持的地图投影

表 62. 受支持的地图投影

Cylindrical projections	Pseudocylindrical projections
Behrmann	Craster parabolic
Cassini	Eckert I
Cylindrical equal area	Eckert II
Equirectangular	Eckert III
Gall's stereographic	Eckert IV
Gauss-Kruger	Eckert V
Mercator	Eckert VI
Miller cylindrical	McBryde-Thomas flat polar quartic
Oblique	Mercator (Hotine) Mollweide
Plate-Carée	Robinson
Times	Sinusoidal (Sansom-Flamsteed)
Transverse Mercator	Winkel I

圆锥投影

表 63. 圆锥投影

Albers 圆锥等面积	Chamberlin 三度投影
双极不对称共轭圆锥	两点等距
Bonne	Hammer-Aitoff 等面积
等距圆锥	Van der Grinten I
Lambert 共轭圆锥	其他
多圆锥	Alaska 系列 E
简单圆锥	Alaska 网格（Snyder 修改的立体投影）

水平或平面投影

- 水平等距
- 一般垂直近边透视
- 心射
- Lambert 水平等面积
- 正交
- 极坐标立体
- 立体

地图投影参数

表 64. 地图投影参数

参数	说明
central_meridian	选作 x 坐标的原点的经线。
scale_factor	通常用于减少地图投影的变形量。
standard_parallel_1	通常没有变形的纬线。也称作“真实比例的纬线”。
standard_parallel_2	通常没有变形的纬线。
longitude_of_center	定义地图投影中心点的经度。
latitude_of_center	定义地图投影中心点的纬度。
latitude_of_origin	选作 y 坐标的原点的纬度。
false_easting	被添加至 x 坐标。用于给出正值。
false_northing	被添加至 y 坐标。用于给出正值。
azimuth	定义斜投影的中心线的正北偏东角。
longitude_of_point_1	地图投影需要的第一点的经度。
latitude_of_point_1	地图投影需要的第一点的纬度。
longitude_of_point_2	地图投影需要的第二点的经度。
latitude_of_point_2	地图投影需要的第二点的纬度。
longitude_of_point_3	地图投影需要的第三点的经度。
latitude_of_point_3	地图投影需要的第三点的纬度。
landsat_number	地球资源技术卫星的号码。
path_number	特定卫星的轨道号。
perspective_point_height	地图投影的透视点离地球的高度。
fipszone	“州平面坐标系”区号。

表 64. 地图投影参数 (续)

参数	说明
区域	UTM 区号。

第16章 Spatial 数据的文件格式

本章说明 DB2 Spatial Extender 的公认表示。因为这些表示由 ESRI 提供而不是特定于 DB2 Spatial Extender 的，所以将它们描述为*公认的*。下列三种 Spatial 值对理解调入和调出 Spatial 数据很重要:

- “开放式 GIS 国际性协议”(OGC) 公认文本表示
- OGC 公认二进制 (WKB) 表示
- ESRI 形状表示

OGC 公认文本表示

DB2 Spatial Extender 具有几个从文本说明生成几何图形的函数:

ST_GeomFromText
从任何几何图形类型的文本表示创建几何图形。

ST_PointFromText
从点文本表示创建点。

ST_LineFromText
从线条文本表示创建线条。

ST_PolyFromText
从多边形文本表示创建多边形。

ST_MPointFromText
从多点文本表示创建多个点。

ST_MLineFromText
从多线条文本表示创建多线条。

ST_MPolyFromText
从复合多边形文本表示创建复合多边形。

文本表示是一个 ASCII 文本格式字符串，它允许以 ASCII 文本形式交换几何图形。可在第三代或第四代语言（3GL 或 4GL）程序中使用这些函数，因为这些函数不需要定义任何特殊程序结构。ST_AsText 函数将现存几何图形转换为文本表示。

每个几何图形类型具有公认文本表示，可用于构造该类型的新实例和将现存实例转换为文本形式以便用字母数字显示。

几何图形的公认文本表示定义如下：表示法 {}* 表示大括号内记号的 0 次或多次重复；大括号不会出现在输出记号列表中。

```
<Geometry Tagged Text> :=
| <Point Tagged Text>
| <LineString Tagged Text>
| <Polygon Tagged Text>
| <MultiPoint Tagged  Text>
| <MultiLineString Tagged Text>
| <MultiPolygon Tagged Text>

<Point Tagged Text> :=
POINT <Point Text>

<LineString Tagged Text> :=
LINESTRING <LineString Text>

<Polygon Tagged Text> :=
POLYGON <Polygon Text>

<MultiPoint Tagged Text> :=
MULTIPOINT <Multipoint Text>

<MultiLineString Tagged Text> :=
MULTILINESTRING <MultiLineString Text>

<MultiPolygon Tagged Text> :=
MULTIPOLYGON <MultiPolygon Text>

<Point Text> := EMPTY
|    <Point>
| Z  <PointZ>
| M  <PointM>
| ZM <PointZM>

<Point> :=  <x>  <y>
<x> := double precision literal
<y> := double precision literal
<PointZ> :=  <x>  <y>  <z>
<x> := double precision literal
<y> := double precision literal
<z> := double precision literal
<PointM> :=  <x>  <y>  <m>
<x> := double precision literal
<y> := double precision literal
<m> := double precision literal
<PointZM> :=  <x>  <y>  <z>  <m>
<x> := double precision literal
<y> := double precision literal
<z> := double precision literal
<m> := double precision literal

<LineString Text> := EMPTY
|    ( <Point Text >   {,  <Point Text> }*  )
| Z  ( <PointZ Text >  {,  <PointZ Text> }*  )
```

```
| M  ( <PointM Text >  {,  <PointM Text> }*  )
| ZM ( <PointZM Text > {,  <PointZM Text> }*  )

<Polygon Text> := EMPTY
| ( <LineString Text > {,< LineString Text > }*  )

<Multipoint Text> := EMPTY
| ( <Point Text >   {,  <Point Text > }*  )

<MultiLineString Text> := EMPTY
| ( <LineString Text > {,< LineString Text>}*  )

<MultiPolygon Text> := EMPTY
| ( < Polygon Text > {,  < Polygon Text > }*  )
```

基本的函数语法为:

```
function (<text description>,<SRID>)
```

SRID（Spatial 参考系标识符和 SPATIAL_REFERENCES 表的主关键字）标识 SPATIAL_REFERENCES 表中存储的几何图形的 Spatial 参考系。在将几何图形插入 Spatial 列之前，它的 SRID 必须与该 Spatial 列的 SRID 匹配。

文本说明由包括在单引号内的三个基本部分组成，例如:

```
<'几何图形类型'> ['坐标类型'] [''坐标列表']
```

其中:

几何图形类型

是下列其中一项: point、linestring、polygon、multipoint、multilinestring 或 multipolygon。

坐标类型

指定几何图形是否具有 Z 坐标或度量单位。若几何图形两项都没有，则将此自变量留为空白。否则，对包含 Z 坐标的几何图形，将坐标类型设置为 Z; 对具有度量单位的几何图形，将坐标类型设置为 M; 对这二者都具有的几何图形，将坐标类型设置为 ZM。

坐标列表

定义几何图形的顶点。坐标列表用逗号分隔且包括在括号内。具有多个部件的几何图形需要多组括号来包括每个部件。若几何图形是空的，则 EMPTY 关键字替换坐标。

表65显示所有可能的文本表示示例的完整列表。

表 65. 几何图形类型及其文本表示

几何图形类型	文本说明	注解
point	point empty	空点

表 65. 几何图形类型及其文本表示 (续)

几何图形类型	文本说明	注解
point	point z empty	具有 z 坐标的空点
point	point m empty	具有度量单位的空点
point	point zm empty	具有 z 坐标和度量单位的空点
point	point (10.05 10.28)	点
point	point z (10.05 10.28 2.51)	具有 z 坐标的点
point	point m (10.05 10.28 4.72)	具有度量单位的点
point	point zm (10.05 10.28 2.51 4.72)	具有 z 坐标和度量单位的点
linestring	linestring empty	空线条
linestring	linestring z empty	具有 z 坐标的空线条
linestring	linestring m empty	具有度量单位的空线条
linestring	linestring zm empty	具有 z 坐标和度量单位的空线条
linestring	linestring (10.05 10.28 , 20.95 20.89)	线条
linestring	linestring z (10.05 10.28 3.09, 20.95 31.98 4.72, 21.98 29.80 3.51)	具有 z 坐标的线条
linestring	linestring m (10.05 10.28 5.84, 20.95 31.98 9.01, 21.98 29.80 12.84)	具有度量单位的线条
linestring	linestring zm ( )	具有 z 坐标和度量单位的线条
polygon	polygon empty	空多边形
polygon	polygon z empty	具有 z 坐标的空多边形
polygon	polygon m empty	具有度量单位的空多边形
polygon	polygon zm empty	具有 z 坐标和度量单位的空多边形
polygon	polygon ((10 10, 10 20, 20 20, 20 15, 10 10))	多边形
polygon	polygon z (( ))	具有 z 坐标的多边形
polygon	polygon m (( ))	具有度量单位的多边形

表 65. 几何图形类型及其文本表示 (续)

几何图形类型	文本说明	注解
polygon	polygon zm (( ))	具有 z 坐标和度量单位的多边形
multipoint	multipoint empty	空多点
multipoint	multipoint z empty	具有 z 坐标的空多点
multipoint	multipoint m empty	具有度量单位的空多点
multipoint	multipoint zm empty	具有 z 坐标与度量单位的空多点
multipoint	multipoint empty	空多点
multipoint	multipoint (10 10, 20 20)	具有两点的多点
multipoint	multipoint z (10 10 2, 20 20 3)	具有 z 坐标的多点
multipoint	multipoint m (10 10 4, 20 20 5)	具有度量单位的多点
multipoint	multipoint zm (10 10 2 4, 20 20 3 5)	具有 z 坐标和度量单位的多点
multilinestring	multilinestring empty	空多线条
multilinestring	multilinestring z empty	具有 z 坐标的空多线条
multilinestring	multilinestring m empty	具有度量单位的空多线条
multilinestring	multilinestring zm empty	具有 z 坐标和度量单位的空多线条
multilinestring	multilinestring (( ))	多线条
multilinestring	multilinestring z (( ))	具有 z 坐标的多线条
multilinestring	multilinestring m (( ))	具有度量单位的多线条
multilinestring	multilinestring zm (( ))	具有 z 坐标和度量单位的多线条
multipolygon	multipolygon empty	空复合多边形
multipolygon	multipolygon z empty	具有 z 坐标的空复合多边形
multipolygon	multipolygon m empty	具有度量单位的空复合多边形
multipolygon	multipolygon z	具有 z 坐标和度量单位的空复合多边形
multipolygon	multipolygon ((( )))	复合多边形
multipolygon	multipolygon z ((( )))	具有 z 坐标的复合多边形

表 65. 几何图形类型及其文本表示 (续)

几何图形类型	文本说明	注解
multipolygon	multipolygon m (((10 10 2, 10 20 3, 20 20 4, 20 15 5, 10 10 2), (50 40 7, 50 50 3, 60 50 4, 60 40 5, 50 40 7)))	具有度量单位的复合多边形
multipolygon	multipolygon zm ((( )))	具有 z 坐标和度量单位的复合多边形

OGC 公认二进制 (WKB) 表示

DB2 Spatial Extender 具有几个从二进制表示生成几何图形的函数:

ST_GeomFromWKB
从任何几何图形类型的 WKB 表示创建几何图形。

ST_PointFromWKB
从点 WKB 表示创建点。

ST_LineFromWKB
从线条 WKB 表示创建线条。

ST_PolyFromWKB
从多边形 WKB 表示创建多边形。

ST_MPointFromWKB
从多点 WKB 表示创建多个点。

ST_MLineFromWKB
从多线条 WKB 表示创建多线条。

ST_MPolyFromWKB
从复合多边形 WKB 表示创建复合多边形。

公认二进制表示是相邻的字节流。它允许以二进制形式在 ODBC 客户机和 SQL 数据库之间交换几何图形。因为这些几何图形函数需要定义 C 程序设计语言结构来映射二进制表示，所以只打算在第三代语言 (3GL) 程序内使用它们。它们不适合第四代语言 (4GL) 环境。 ST_AsBinary 函数将现存几何图形值转换为公认二进制表示。

几何图形的公认二进制表示是通过将几何图形实例串行化为数字类型的序列来实现的。从集合（无符号整数，双精度）抽取这些类型，然后将每个数字类型串行化为字节的序列。使用数字类型的两个良好定义的标准二进制表示 (NDR, XDR) 之

一使这些类型串行化。已串行化的字节之前的一字节标记描述用于几何图形字节流的特定二进制编码 （NDR 或 XDR）。几何图形的两种编码之间的唯一差异是字节次序的差异: XDR 编码是 “大尾结构”； NDR 编码是 “小尾结构”。

数字类型定义

*无符号整数*是 32 位（4 字节）的数据类型，它对 [0, 4294967295] 范围内的非负整数进行编码。

*双精度*是 64 位（8 字节）的双精度数据类型，它使用 IEEE 754 双精度格式对双精度数进行编码。

这些定义是 XDR 和 NDR 的公共定义。

数字类型的 XDR（大尾结构）编码

无符号整数的 XDR 表示是 “大尾结构” （最高有效位字节优先）。

双精度数的 XDR 表示是 “大尾结构” （符号位是第一个字节）。

数字类型的 NDR（小尾结构）编码

无符号整数的 NDR 表示是 “小尾结构” （最低有效位优先）。

双精度数的 NDR 表示是 “小尾结构” （符号位是最后一个字节）。

NDR 和 XDR之间的转换

无符号整数和双精度数的 NDR 和 XDR 数据类型之间的转换是一个简单操作。它涉及使字节流中的每个无符号整数或双精度数内的字节次序反向。

WKBGeometry 字节流的说明

本节描述几何图形的公认二进制表示。基本构件块是点的字节流，它由两个双精度数组成。其他几何图形的字节流是使用已定义的几何图形的字节流构建的。

```
// Basic Type definitions
// byte : 1 byte
// uint32 : 32 bit unsigned integer  (4 bytes)
// double : double precision number (8 bytes)

// Building Blocks : Point, LinearRing

Point {
  double x;
  double y;
};
```

```
LinearRing   {
  uint32  numPoints;
  Point   points[numPoints];
};
enum wkbGeometryType {
  wkbPoint = 1,
  wkbLineString = 2,
  wkbPolygon = 3,
  wkbMultiPoint = 4,
  wkbMultiLineString = 5,
  wkbMultiPolygon = 6,
};
enum wkbByteOrder {
  wkbXDR = 0,                                    // Big Endian
  wkbNDR = 1                                   // Little Endian
};
WKBPoint {
  byte     byteOrder;
  uint32   wkbType;                                        // 1
  Point    point;
};
WKBLineString {
  byte     byteOrder;
  uint32   wkbType;                                        // 2
  uint32   numPoints;
  Point    points[numPoints];
}

WKBPolygon    {
  byte                byteOrder;
  uint32            wkbType;                               // 3
  uint32            numRings;
  LinearRing        rings[numRings];
}
WKBMultiPoint    {
  byte                byteOrder;
  uint32            wkbType;                               // 4
  uint32            num_wkbPoints;
  WKBPoint            WKBPoints[num_wkbPoints];
}
WKBMultiLineString    {
  byte              byteOrder;
  uint32            wkbType;                               // 5
  uint32            num_wkbLineStrings;
  WKBLineString     WKBLineStrings[num_wkbLineStrings];
}

wkbMultiPolygon {
  byte              byteOrder;
  uint32            wkbType;                               // 6
  uint32            num_wkbPolygons;
  WKBPolygon        wkbPolygons[num_wkbPolygons];
}

WKBGeometry  {
```

```
  union {
    WKBPoint                point;
    WKBLineString           linestring;
    WKBPolygon              polygon;
    WKBMultiPoint           mpoint;
    WKBMultiLineString      mlinestring;
    WKBMultiPolygon         mpolygon;
  }
};
```

下图显示一种 NDR 表示。

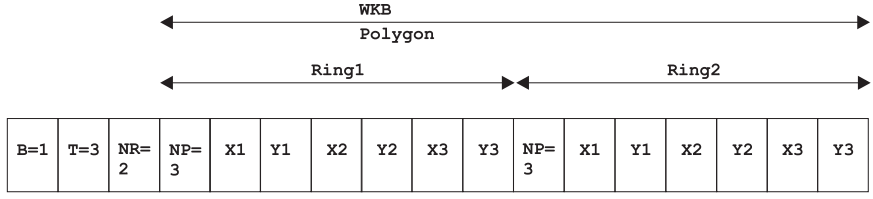


图 39. NDR 格式的表示. (B=1)，类型为多边形 (T=3)，具有 2 条直线 (NR=2)，每个环具有 3 个点 (NP=3)。

WKB 表示的要求

几何图形的公认二进制表示设计成能表示一些几何图形的实例，在“几何图形模型”和“OpenGIS 抽象规范”中描述了这些几何图形。

这些要求对环、多边形和复合多边形隐含下列要求:

线性环 环是简单且封闭的，这意指线性环不能自相交。

多边形 多边形的边界内的任何两个线性环不能彼此交叉。多边形的边界内的线性环最多可在一个点相交，但只能为正切。

复合多边形

若两个多边形是某个复合多边形的元素，则其内部不能相交。若任何两个多边形是某个复合多边形的元素，则其边界仅可接触有限的点数。

ESRI 形状表示

DB2 Spatial Extender 具有几个从 ESRI 形状表示生成几何图形的函数: 除了开放式 GIS 公认二进制表示支持的二维表示之外，ESRI 形状表示还支持可选的 Z 坐标和度量单位。下列函数从 ESRI 形状生成几何图形:

ST_GeometryFromShape

从任何几何图形类型的形状表示创建几何图形。

ST_PointFromShape
从点形状表示创建点。

ST_LineFromShape
从线条形状表示创建线条。

ST_PolyFromShape
从多边形形状表示创建多边形。

ST_MPointFromShape
从多点形状表示创建多个点。

ST_MLineFromShape
从多线条形状表示创建多线条。

ST_MPolyFromShape
从复合多边形形状表示创建复合多边形。

这些函数的一般语法相同。第一个自变量是形状表示，以二进制大对象 (BLOB) 数据类型的格式输入。第二个自变量是给几何图形指定的 Spatial 参考系标识符整数。GeometryFromShape 函数具有以下语法:

```
db2gse.GeometryFromShape(ShapeGeometry Blob(1M), cr db2gse.coordref)
```

因为这些形状函数需要定义 C 程序设计语言结构来映射二进制表示，所以只打算在 3GL 程序内使用它们，它们不适合 4GL 环境。 AsShape 函数将几何图形值转换为 ESRI 形状表示。

形状类型 0 指示空形状，即该形状没有几何图形数据。

值	形状类型
0	空形状
1	点
3*	折线
5	多边形
8	多点
11	PointZ
13	PolyLineZ
15	PolygonZ
18	MultiPointZ
21	PointM
23	PolyLineM
25	PolygonM

值	形状类型
28	MultiPointM

注: * 上面没有指定的形状类型（2、4、6 等）保留供将来使用。

XY 空间中的形状类型

点

点由按 X、Y 次序的双精度坐标对组成。

表 66. 点字节流内容

位置	字段	值	类型	数目	次序
字节 0	形状类型	1	整数	1	小尾结构
字节 4	X	X	双精度	1	小尾结构
字节 12	Y	Y	双精度	1	小尾结构

多个点

多个点由点的集合组成。按次序 Xmin、Ymin、Xmax、Ymax 存储的边界框。

表 67. 多点字节流内容

位置	字段	值	类型	数目	次序
字节 0	形状类型	8	整数	1	小尾结构
字节 4	框	框	双精度	4	小尾结构
字节 36	点数	点数	整数	1	小尾结构
字节 40	点	点	点	点数	小尾结构

折线

折线是由一个或多个部件组成的有序顶点集。每个部件是两点或多个点的连接序列。 点彼此可能连接或可能不连接。部件彼此可能相交或可能不相交。

因为此规范不禁止具有相同坐标的连续点，因此形状文件读程序必须处理这种情况。在另一方面，不允许可能导致的退化零长度部件。

“折线”的字段如下:

框 按次序 Xmin、Ymin、Xmax、Ymax 存储的“折线”的边界框。

部件数 “折线”中的部件数。

点数 所有部件的点的总数。

部件 长度为“部件数”的数组。 每个“折线”存储它在点数组中的第一个点的下标。数组下标是相对于 0 的。

点 长度为“点数”的数组。 头尾相接地存储“折线”中每个部件的点。第 2 个部件的点跟随第 1 个部件的点，以此类推。部件数组保持每个部件的起始点的数组下标。 在点数组中的部件之间没有定界符。

表 68. 折线字节流内容

位置	字段	值	类型	数目	次序
字节 0	形状类型	3	整数	1	小尾结构
字节 4	框	框	双精度	4	小尾结构
字节 36	部件数	部件数	整数	1	小尾结构
字节 40	点数	点数	整数	1	小尾结构
字节 44	部件	部件	整数	部件数	小尾结构
字节 X	点	点	点	点数	小尾结构

注: X = 44 + 4 * 部件数。

多边形

多边形由一个或多个环组成。环是四个或更多点的连接序列，形成封闭的不自相交的环路。一个多边形可包含多个外环。环的顶点的次序或方向指示环的哪一边是多边形的内部。按顶点次序沿着环行走的观察者的右边邻近区域是多边形内部的邻近区域。定义多边形中孔的环的顶点按反时针方向。因此，单个有环的多边形的顶点总是按顺时针次序。多边形的环称作部件。

因为此规范不禁止具有相同坐标的连续点，因此形状文件读程序必须处理这种情况。在另一方面，不允许可能导致的退化零长度部件。

多边形的字段如下:

框 按次序 Xmin、Ymin、Xmax、Ymax 存储的“多边形”的边界框。

部件数 多边形中的环数。

点数 所有环的点的总数。

部件 长度为“部件数”的数组。 存储每个环在点数组中的第一个点的下标。 数组下标是相对于 0 的。

点 长度为“点数”的数组。 头尾相接地存储多边形中每个环的点。第 2 个环的点跟随第 1 个环的点，以此类推。 部件数组保持每个环的起始点的数组下标。 在点数组中的环之间没有定界符。

关于“多边形”形状的重要注意事项:

- 环是封闭的（环的第一个顶点和最后一个顶点“必须”相同）。
- 点数组中环的次序是无意义的。

- 形状文件中存储的多边形必须是清楚的。清楚的多边形是这样一个多边形:
  - 没有自相交点。这意指属于某个环的线段不能与属于另一个环的线段相交。多边形的环可在顶点处彼此接触但不能沿线段接触。共线的线段被认为是相交的。
  - 多边形的内部在定义它的直线的“正确”一边。按顶点次序沿着环行走的观察者的右边邻近区域是多边形的内部。因此，单个有环的多边形的顶点总是按顺时针次序。在这些多边形中定义孔的环具有反时针方向。

  当在多边形中定义孔的环也按顺时针方向时，“复杂”多边形出现，导致内部重叠。

多边形实例示例:

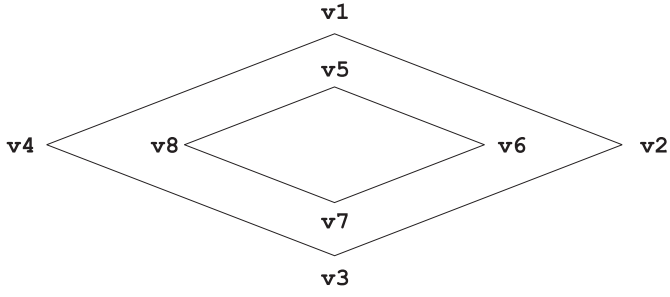


图 40. 具有一个孔和八个顶点的多边表

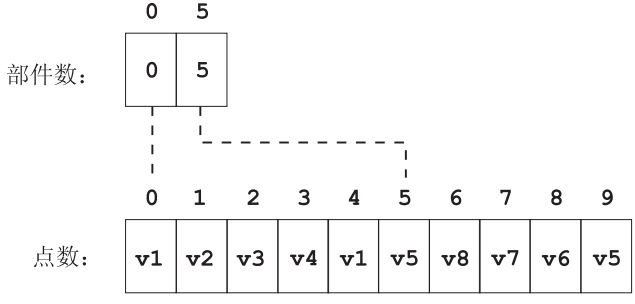


图 41. 多边形字节流的内容. “部件数”等于 2，“点数”等于 10。注意环形（孔）多边形的点的次序已反向。

表 69. 多边形字节流内容

位置	字段	值	类型	数目	次序
字节 0	形状类型	5	整数	1	小尾结构
字节 4	框	框	双精度	4	小尾结构
字节 36	部件数	部件数	整数	1	小尾结构
字节 40	点数	点数	整数	1	小尾结构
字节 44	部件	部件	整数	部件数	小尾结构
字节 X	点	点	点	点数	小尾结构

注: X = 44 + 4 * 部件数。

XY 空间中带度量单位的形状类型

PointM

PointM 由按 X、Y 次序的双精度坐标对加上度量单位 M 组成。

表 70. PointM 字节流内容

位置	字段	值	类型	数目	次序
字节 0	形状类型	21	整数	1	小尾结构
字节 4	X	X	双精度	1	小尾结构
字节 12	Y	Y	双精度	1	小尾结构
字节 20	M	M	双精度	1	小尾结构

MultiPointM

MultiPointM 的字段如下:

框 按次序 Xmin、min、Xmax、Ymax 存储的 MultiPointM 的边界框。

点数 点的数目。

点 长度为“点数”的“点”数组。

M 数 跟随的“度量单位”数。“M 数”仅可为两个值，若没有“度量单位”跟随此字段则为零，若有“度量单位”，则“M 数”等于“点数”。

M 范围
按 Mmin、Mmax 次序存储的 MultiPointM 的最小和最大度量单位。

M 数组
长度为“点数”的“度量单位”数组。

表 71. MultiPointM 字节流内容

位置	字段	值	类型	数目	次序
字节 0	形状类型	28	整数	1	小尾结构
字节 4	框	框	双精度	4	小尾结构
字节 36	点数	点数	整数	1	小尾结构
字节 40	点	点	点	点数	小尾结构
字节 X	M 数	M 数	整数	1	小尾结构
字节 X+4*	Mmin	Mmin	双精度	1	小尾结构
字节 X+12*	Mmax	Mmax	双精度	1	小尾结构
字节 X+20*	M 数组	M 数组	双精度	点数	小尾结构

注:

1. X = 40 + (16 * 点数)
2. * 可选

PolyLineM

一个形状文件 PolyLineM 由一个或多个部件组成。每个部件是两点或多个点的连接序列。 部件彼此可能连接或可能不连接。 部件彼此可能相交或可能不相交。

PolyLineM 的字段如下:

框 按次序 Xmin、min、Xmax、Ymax 存储的 PolyLineM 的边界框。

部件数 PolyLineM 中的部件数。

点数 所有部件的点的总数。

部件 长度为“部件数”的数组。 存储每个部件在点数组中的第一个点的下标。数组下标是相对于 0 的。

点 长度为“点数”的数组。 头尾相接地存储 PolyLineM 中每个部件的点。第 2 个部件的点跟随第 1 个部件的点，以此类推。部件数组保持每个部件的起始点的数组下标。 在点数组中的部件之间没有定界符。

M 数 跟随的“度量单位”数。 “M 数”仅可为两个值，若没有“度量单位”跟随此字段则为零，若有“度量单位”，则“M 数”等于“点数”。

M 范围

按 Mmin、Mmax 次序存储的 PolyLineM 的最小和最大度量单位。

M 数组

长度为“点数”的数组。头尾相接地存储 PolyLineM 中每个部件的点。第

2 个部件的度量单位跟随第 1 个部件的度量单位，以此类推。部件数组保持每个部件的起始点的数组下标。 在度量单位数组中的部件之间没有定界符。

表 72. PolyLineM 字节流内容

位置	字段	值	类型	数目	次序
字节 0	形状类型	13	整数	1	小尾结构
字节 4	框	框	双精度	4	小尾结构
字节 36	部件数	部件数	整数	1	小尾结构
字节 40	点数	点数	整数	1	小尾结构
字节 44	部件	部件	整数	部件数	小尾结构
字节 X	点	点	点	点数	小尾结构
字节 Y	M 数	M 数	整数	1	小尾结构
字节 Y+4*	Mmin	Mmin	双精度	1	小尾结构
字节 Y+12*	Mmax	Mmax	双精度	1	小尾结构
字节 Y+20*	M 数组	M 数组	双精度	点数	小尾结构

注:

1. X = 44 + (4 * 部件数), Y = X + (16 * 点数)。
2. * 可选

PolygonM

一个 PolygonM 由几个环组成。环是封闭的不自相交的环路。 注意: 交点是在 XY 空间而*不是*在 XYM 空间中计算。 PolygonM 可包含多个外环。PolygonM 的环称作部件。

PolygonM 的字段如下:

框 按次序 Xmin、min、Xmax、Ymax 存储的 PolygonM 的边界框。

部件数 PolygonM 中的环数。

点数 所有环的点的总数。

部件 长度为“部件数”的数组。 存储每个环在点数组中的第一个点的下标。 数组下标是相对于 0 的。

点 长度为“点数”的数组。头尾相接地存储 PolygonM 中每个环的点。第 2 个环的点跟随第 1 个环的点，以此类推。部件数组保持每个环的起始点的数组下标。在点数组中的环之间没有定界符。

M 数 跟随的“度量单位”数。 “M 数”仅可为两个值，若没有“度量单位”跟随此字段则为零，若有“度量单位”，则“M 数”等于“点数”。

M 范围

按 Mmin、Mmax 次序存储的 PolygonM 的最小和最大度量单位。

M 数组

长度为“点数”的数组。头尾相接地存储 PolygonM 中每个环的度量单位。第 2 个环的度量单位跟随第 1 个环的度量单位，以此类推。部件数组保持每个环的起始度量单位的数组下标。在度量单位数组中的环之间没有定界符。

关于 PolygonM 形状的重要注意事项:

- 环是封闭的（环的第一个顶点和最后一个顶点必须是同一点）。
- 点数组中环的次序是无意义的。

表 73. PolygonM 字节流内容

位置	字段	值	类型	数目	次序
字节 0	形状类型	15	整数	1	小尾结构
字节 4	框	框	双精度	4	小尾结构
字节 36	部件数	部件数	整数	1	小尾结构
字节 40	点数	点数	整数	1	小尾结构
字节 44	部件	部件	整数	部件数	小尾结构
字节 X	点	点	点	点数	小尾结构
字节 Y	M 数	M 数	整数	1	小尾结构
字节 Y+4*	Mmin	Mmin	双精度	1	小尾结构
字节 Y+12*	Mmax	Mmax	双精度	1	小尾结构
字节 Y+20*	M 数组	M 数组	双精度	点数	小尾结构

注:

1. X = 44 + (4 * 部件数), Y = X + (16 * 点数)。
2. * 可选

XYZ 空间中的形状类型

PointZ

一个 PointZ 由按 X、Y、Z 次序的三个一组的双精度坐标加上度量单位组成。

表 74. PointZ 字节流内容

位置	字段	值	类型	数目	次序
字节 0	形状类型	11	整数	1	小尾结构
字节 4	X	X	双精度	1	小尾结构
字节 12	Y	Y	双精度	1	小尾结构

表 74. PointZ 字节流内容 (续)

位置	字段	值	类型	数目	次序
字节 20	Z	Z	双精度	1	小尾结构
字节 28	度量单位	M	双精度	1	小尾结构

MultiPointZ

MultiPointZ 表示一组 PointZ，如下所示:

- 按次序 Xmin、Ymin、Xmax、Ymax 存储的边界框。
- 边界 Z 范围按次序 Zmin、Zmax 存储。边界 M 范围按次序 Mmin、Mmax 存储。

表 75. MultiPointZ 字节流内容

位置	字段	值	类型	数目	次序
字节 0	形状类型	18	整数	1	小尾结构
字节 4	框	框	双精度	4	小尾结构
字节 36	点数	点数	整数	1	小尾结构
字节 40	点	点	点	点数	小尾结构
字节 X	Zmin	Zmin	双精度	1	小尾结构
字节 X+8	Zmax	Zmax	双精度	1	小尾结构
字节 X+16	Z 数组	Z 数组	双精度	点数	小尾结构
字节 Y	M 数	M 数	整数	1	小尾结构
字节 Y+4*	Mmin	Mmin	双精度	1	小尾结构
字节 Y+12*	Mmax	Mmax	双精度	1	小尾结构
字节 Y+20*	M 数组	M 数组	双精度	点数	小尾结构

注:

1. X = 40 + (16 * 点数); Y = X + 16 + (8 * 点数)
2. * 可选

PolyLineZ

一个 PolyLineZ 由一个或多个部件组成。每个部件是两点或多个点的连接序列。部件彼此可能连接或可能不连接。部件彼此可能相交或可能不相交。

PolyLineZ 的字段如下:

框 按次序 Xmin、min、Xmax、Ymax 存储的 PolyLineZ 的边界框。

部件数 PolyLineZ 中的部件数。

点数 所有部件的点的总数。

部件 长度为“部件数”的数组。 存储每个部件在点数组中的第一个点的下标。数组下标是相对于 0 的。

点 长度为“点数”的数组。 头尾相接地存储 PolyLineZ 中每个部件的点。第 2 个部件的点跟随第 1 个部件的点，以此类推。部件数组保持每个部件的起始点的数组下标。 在点数组中的部件之间没有定界符。

Z 范围

按次序 Zmin、Zmax 存储的 PolyLineZ 的最小和最大 Z 值。

Z 数组

长度为“点数”的数组。头尾相接地存储 PolyLineZ 中每个部件的 Z 值。第 2 个部件的 Z 值跟随第 1 个部件的 Z 值，以此类推。部件数组保持每个部件的起始点的数组下标。在 Z 数组中的部件之间没有定界符。

M 数 跟随的“度量单位”数。 “M 数”仅可为两个值，若没有“度量单位”跟随此字段则为零，若有“度量单位”，则“M 数”等于“点数”。

M 范围

按 Mmin、Mmax 次序存储的 PolyLineZ 的最小和最大度量单位。

M 数组

长度为“点数”的数组。头尾相接地存储 PolyLineZ 中每个部件的度量单位。第 2 个部件的度量单位跟随第 1 个部件的度量单位，以此类推。部件数组保持每个部件的起始度量单位的数组下标。在度量单位数组中的部件之间没有定界符。

表 76. PolyLineZ 字节流内容

位置	字段	值	类型	数目	次序
字节 0	形状类型	13	整数	1	小尾结构
字节 4	框	框	双精度	4	小尾结构
字节 36	部件数	部件数	整数	1	小尾结构
字节 40	点数	点数	整数	1	小尾结构
字节 44	部件	部件	整数	部件数	小尾结构
字节 X	点	点	点	点数	小尾结构
字节 Y	Zmin	Zmin	双精度	1	小尾结构
字节 Y+8	Zmax	Zmax	双精度	1	小尾结构
字节 Y+16	Z 数组	Z 数组	双精度	点数	小尾结构
字节 Z	M 数	M 数	整数	1	小尾结构
字节 Z+4*	Mmin	Mmin	双精度	1	小尾结构
字节 Z+12*	Mmax	Mmax	双精度	1	小尾结构

表 76. PolyLineZ 字节流内容 (续)

位置	字段	值	类型	数目	次序
字节 Z+20*	M 数组	M 数组	双精度	点数	小尾结构

注:

1. X = 44 + (4 * 部件数), Y = X + (16 * 点数), Z = Y + 16 + (8 * 点数)
2. * 可选

PolygonZ

一个 PolygonZ 由几个环组成。环是封闭的不自相交的环路。一个 PolygonZ 可包含多个外环。PolygonZ 的环称作部件。

PolygonZ 的字段如下:

框 按次序 Xmin、min、Xmax、Ymax 存储的 PolygonZ 的边界框。

部件数 PolygonZ 中的环数。

点数 所有环的点的总数。

部件 长度为“部件数”的数组。存储每个环在点数组中的第一个点的下标。数组下标是相对于 0 的。

点 长度为“点数”的数组。头尾相接地存储 PolygonZ 中每个环的点。第 2 个环的点跟随第 1 个环的点，以此类推。部件数组保持每个环的起始点的数组下标。在点数组中的环之间没有定界符。

Z 范围

按次序 Zmin、Zmax 存储的弧的最小和最大 Z 值。

Z 数组

长度为“点数”的数组。头尾相接地存储 PolygonZ 中每个环的 Z 值。第 2 个环的 Z 值跟随第 1 个环的 Z 值，以此类推。部件数组保持每个环的起始 Z 值的数组下标。在 Z 值数组中的环之间没有定界符。

M 数 跟随的“度量单位”数。“M 数”仅可为两个值，若没有“度量单位”跟随此字段则为零，若有“度量单位”，则“M 数”等于“点数”。

M 范围

按 Mmin、Mmax 次序存储的 PolygonZ 的最小和最大度量单位。

M 数组

长度为“点数”的数组。头尾相接地存储 PolygonZ 中每个环的度量单位。第 2 个环的度量单位跟随第 1 个环的度量单位，以此类推。部件数组保持每个环的起始度量单位的数组下标。在度量单位数组中的环之间没有定界符。

关于 PolygonZ 形状的重要注意事项:

- 环是封闭的（环的第一个顶点和最后一个顶点“必须”相同）。
- 点数组中环的次序是无意义的。

表 77. PolygonZ 字节流内容

位置	字段	值	类型	数目	次序
字节 0	形状类型	15	整数	1	小尾结构
字节 4	框	框	双精度	4	小尾结构
字节 36	部件数	部件数	整数	1	小尾结构
字节 40	点数	点数	整数	1	小尾结构
字节 44	部件	部件	整数	部件数	小尾结构
字节 X	点	点	点	点数	小尾结构
字节 Y	Zmin	Zmin	双精度	1	小尾结构
字节 Y+8	Zmax	Zmax	双精度	1	小尾结构
字节 Y+16	Z 数组	Z 数组	双精度	点数	小尾结构
字节 Z	M 数	M 数	整数	1	小尾结构
字节 Z+4*	Mmin	Mmin	双精度	1	小尾结构
字节 Z+12*	Mmax	Mmax	双精度	1	小尾结构
字节 Z+20*	M 数组	M 数组	双精度	点数	小尾结构

第3部分 附录及附属资料

注意事项

IBM 可能未在所有国家中提供本文档中讨论的产品、服务或功能部件。关于您所在区域目前可用的产品及服务的信息，请向当地的 IBM 代表咨询。任何对 IBM 产品、程序或服务的引用并不说明或暗示只能使用 IBM 的产品、程序或服务。凡是同等功能的产品、程序或服务，只要不侵犯 IBM 的知识产权，都可以用来替代 IBM 产品、程序或服务。当然，评估和验证非 IBM 产品、程序或服务均由用户自行负责。

本文档的议题可能涉及 IBM 的某些专利或正在申请中的专利的应用。提供本文档，并不表示允许您使用这些专利。您可以将许可证查询以书面形式发送给:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

关于双字节 (DBCS) 许可证查询的信息，请与您所在国家的 IBM 知识产权部门联系，将查询以书面形式发送至寄往:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

以下段落不适用于英国与其它当地法律不允许这种供应方式的国家: 国际商用机器公司『按原样』出版此书，不做任何明确或暗示的担保，包括但不限于不侵权、可销售或适用于特殊目的暗示性担保。一些地区在某些事务中不允许否认拒绝明确或暗示的担保，因此本条款可能不适合您。

本信息中可能有技术方面不够准确的地方或印刷错误。此处的信息将定期更改；这些信息将包含在本书新的版本中。IBM 可以随时对本书中说明的产品和/或程序进行改进和/或改动，而不必通知您。

此信息中对非 IBM Web 站点的任何引用仅是为了方便起见，而不以任何方式为那些 Web 站点作保证。那些 Web 站点的资料并非此 IBM 产品资料的一部分，使用那些 Web 站点的风险由您自己承担。

对于您所提供的任何信息，IBM 有权利以任何她认为适当的方式使用或散发，而不必对您负任何责任。

为了以下目的：(1) 允许在独立创建的程序和其他程序（包括本程序）之间进行信息交换 (2) 允许对已经交换的信息进行相互使用，而希望获取本程序有关信息的合法用户请与下列地址联系:

IBM Canada Limited
Office of the Lab Director
1150 Eglinton Ave. East
North York, Ontario
M3C 1H7
CANADA

只要遵守适当的条款和条件，包括某些情形下的一定数量的付款，都可获取这方面的信息。

这些信息中描述的特许程序及其所有可用的特许资料，按 IBM 客户协议、IBM 国际程序许可证协议或任何等价的协议中的条款，由 IBM 提供。

此处包含的所有性能数据都是在受控环境中确定的。因此，在其他操作环境中获得的结果可能与之相差很大。某些测量可能是在开发级的系统上进行的，不能保证这些测量方法在通用系统上同样可用。此外，某些测量方法可能是通过外推法归纳来估计的。实际结果可能会有所不同。此文档的用户应针对他们的特定环境验证数据是否适用。

涉及非 IBM 产品的信息可从这些产品的供应商、其发行公告或其它公众可用源得到。IBM 未测试这些产品，因此不能确认性能的精确度、兼容性或其它对非 IBM 产品的索赔赔偿要求等。有关非 IBM 产品功能方面的问题可向它们的供应商提出。

所有关于 IBM 未来方向或意向的声明都可能随时更改或撤消，而不作任何通知，并且仅代表发展目标。

此信息包含了用于日常商业处理的数据和报表的示例。为了尽可能完整地说明问题，这些示例中包含了个人、公司、品牌和产品的名称。所有这些名称都是虚构的，如与实际商业企业所使用的名称和地址相似，纯属巧合。

版权许可证:

本信息中可能包含用源语言编写的示例应用程序，它们说明了各种不同的操作平台上的程序设计技术。您可以为了开发、使用、市场营销或分发应用程序(这些应用程序遵守编写这些示例程序的操作平台的应用程序接口)的目的，以任何形式复

制、修改和分发这些示例程序，不用向 IBM 付费。这些例子未经所有条件下的完整测试。因此，IBM 不能保证或暗示其可靠性、可用性或这些程序的功能。

这些样本程序或任何派生产品的每个副本或任何部分必须包含如下的版权公告:

©（您的公司名称）（年度）。此代码各部分派生自 “IBM 公司样本程序”。© Copyright IBM Corp. _输入年份_。All rights reserved.

注册商标

以星号 (*) 标出的下列术语是 IBM 公司在美国和／或其他国家的商标。

ACF/VTAM
AISPO
AIX
AIX/6000
AIXwindows
AnyNet
APPN
AS/400
BookManager
CICS
C Set++
C/370
DATABASE 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extender
DB2 OLAP Server
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eNetwork
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
LAN DistanceMVS
MVS/ESA
MVS/XA
Net.Data
OS/2
OS/390
OS/400
PowerPC
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/DS
SQL/400
System/370
System/390
SystemView
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WIN-OS/2

下列各项是其他公司的商标或注册商标:

Microsoft、Windows、和 Windows NT 是 Microsoft 公司的商标或注册商标。

Java 或所有基于 Java 的商标和标志以及 Solaris 是 Sun Microsystems 公司在美国和／或其他国家的商标。

Tivoli 和 NetView 是 Tivoli Systems 公司在美国和／或其他国家的商标。

UNIX 是经 X/Open 有限公司唯一许可的在美国和／或其它国家的注册商标。

以双星号 (**) 标出的其他公司、产品或服务名，可能是其他公司的商标或服务标志。

索引

[F]

[J]

[K]

[L]

[M]

[N]

[P]

[Q]

[R]

[S]

[T]

[W]

[X]

[Y]

[Z]

[特别字符]

A

B

D

E

G

I

U

W

X

Y

Z

与 IBM 联系

如果有技术问题，请在与“DB2 客户支持中心”联系之前复查并执行 *Troubleshooting Guide* 所建议的操作。本指南对您可以收集哪些信息以使“DB2 客户支持中心”更好地为您服务提出了建议。

要获取信息或订购任何“DB2 通用数据库”产品，与当地分支机构的 IBM 代表联系，或与任何特许 IBM 软件经销商联系。

您如果住在美国，请致电下列其中一个号码:

- 1-800-237-5511，可获得客户支持
- 1-888-426-4343，可了解所提供的服务项目

产品信息

您如果住在美国，请致电下列其中一个号码:

- 1-800-IBM-CALL (1-800-426-2255) 或 1-800-3IBM-OS2 (1-800-342-6672)，可订购产品或获取一般信息。
- 1-800-879-2755，可订购出版物。

http://www.ibm.com/software/data/

DB2 万维网网页提供关于新闻、产品说明、培训计划等等的当前 DB2 信息。

http://www.ibm.com/software/data/db2/library/

“DB2 产品和服务技术库”可供您访问常见问题、修订、书籍以及最新的 DB2 技术资料。

注: 此资料可能只有英文版。

http://www.elink.ibmlink.ibm.com/pbl/pbl/

International Publications Ordering Web 站点提供关于如何订购书籍的信息。

http://www.ibm.com/education/certify/

IBM Web 站点中的“专业认证程序”提供各种 IBM 产品（包括 DB2）的认证测试信息。

ftp.software.ibm.com

以匿名形式注册。可在目录 `/ps/products/db2` 中找到有关 DB2 和许多其他产品的演示程序、修订、信息和工具。

comp.databases.ibm-db2, bit.listserv.db2-l

这些 Internet 新闻组可供用户来讨论使用 DB2 产品的经验。

On Compuserve: GO IBMDB2

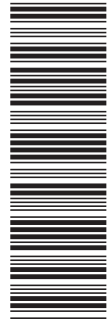
输入此命令来访问 IBM DB2 系列论坛。这些论坛支持所有的 DB2 产品。

有关如何在美国以外的地区与 IBM 联系的信息，参见 *IBM Software Support Handbook* 的附录 A。要访问此文档，访问以下 Web 页面：http://www.ibm.com/support/，然后选择该页面底部附近的 IBM Software Support Handbook 链接。

注： 在某些国家，IBM 特许经销商应与他们的经销商支持机构联系，而不是与“IBM 支持中心”联系。

Part Number: CT7C0SC

Printed in China



CT7C0SC

SB84-0249-00

Spine information:

IBM® DB2® Spatial Extender

DB2 Spatial Extender 用户指南和参考 *版本 7*