IBM® DB2® 通用数据库

XML Extender 管理和程序设计

版本 7

IBM® DB2® 通用数据库

XML Extender 管理和程序设计

版本 7

在使用本资料和它支持的产品之前，请阅读第259页的『注意事项』中的一般信息。

本文档包含 IBM 的专利信息。它在许可证协议下提供，并受版权法保护。本出版物包含的信息不包括任何产品保证，且本手册提供的任何声明不应作如此解释。

通过您当地的 IBM 代表或 IBM 分部可订购出版物，或者，通过致电 1-800-879-2755（在美国）或 1-800-IBM-4YOU（在加拿大）来订购出版物。

当您发送信息给 IBM 后，即授予 IBM 非专有权， IBM 对于您所提供的任何信息，有权利以任何它认为适当的方式使用或散发，而不必对您负任何责任。

© Copyright International Business Machines Corporation 1999, 2000. All rights reserved.

目录

© Copyright IBM Corp. 1999, 2000

表

© Copyright IBM Corp. 1999, 2000

关于本书

本节描述以下信息:

本书读者

本书供下列人员使用:

- 在 DB2 应用程序中使用 XML 数据并熟悉 XML 概念的人员。 此文档的读者应对 XML 和 DB2 有一般的了解。 要了解有关 XML 和相关主题的详情，参考以下 Web 站点:

  http://www.w3c.org/XML

  要了解有关 DB2 的详情，参考以下 Web 站点:

  http://www.ibm.com/software/data/db2/library
- 熟悉 DB2 管理概念、工具和技术的 DB2 数据库管理员。
- 熟悉 SQL 和可用于 DB2 应用程序的一种或多种程序设计语言的 DB2 应用程序员。

如何获取本书的当前版本

可在以下 XML Extender Web 站点获取本书的最新版本:

http://www.ibm.com/software/data/db2/extenders/xmlext/library.html

如何使用本书

本书的结构如下:

第 1 部分. 介绍

此部分提供了 XML Extender 的概述以及如何在商业应用程序中使用它的概述。它包含了帮助您设置和运行的入门方案。

© Copyright IBM Corp. 1999, 2000

第 2 部分. 管理

此部分描述如何准备和维护用于 XML 数据的 DB2 数据库。如果您需要管理包含 XML 数据的 DB2 数据库，则阅读此部分。

第 3 部分. 程序设计

此部分描述如何管理 XML 数据。如果您需要访问和处理 DB2 应用程序中的 XML 数据，则阅读此部分。

第 4 部分. 参考

此部分描述如何使用 XML Extender 管理命令、用户定义类型、用户定义函数和存储过程。 它还列示了 XML Extender 发出的消息和代码。如果您熟悉 XML Extender 概念和任务，但需要有关用户定义类型 (UDT)、用户定义函数 (UDF)、命令、消息、元数据表、控制表或代码的信息，则阅读此部分。

第 5 部分. 附录

附录描述了用于文档访问定义的 DTD、示例和入门方案的样本和其他 IBM XML 产品。

本书中有关 DB2 UDB 版本修订包 2 的新增内容

本文档包含有关下列内容的新的或重新编写的信息:

- 设置和启动管理向导
- “更新”UDF 如何修改 XML 文档
- XMLClob UDF 如何返回结果
- 如何为存储过程增加 CLOB 限制
- DAD 需求更改:
  - 当组合新的 XML 文档时包括样式表处理指令
  - 当仅指定一张表时，为第一个 RDB 节点使用丢失或空的条件。
- JDBC 如何使参数标志符强制转型
- 使用代码页:
  - 合法编码说明
  - 转换假设
  - 转换方案
- XML Extender 参数限制附录

更改的信息用垂直更改条 『|』 进行标记。

参见自述文件以了解有关此修订包的所有缺陷修订。

有关常见问题的修订和解决方案的更多信息，检查 XML Extender Web 站点的“支持”页中的 FAQ 和已知问题列表：
http://www.ibm.com/software/data/db2/extenders/xmlext/support.html

此书包括在 DB2 UDB 版本 7 信息中心中

XML Extender 可使用以下步骤包括在 DB2 信息中心中：

- 将此书的 HTML 文件从安装了您所用语言的 XML 产品的 DOC 子目录，复制到您所用语言的 DB2 UDB 文档目录：

  在 Windows 上，信息中心所在的目录是 sqllib\doc\html\db2sx

  在 UNIX 上，信息中心所在的目录是 *install directory*/doc/html/db2sx
- 重新启动信息中心，而后此书将包括在**书籍**标签中。

突出显示约定

本书使用下列约定：

粗体	粗体文本指示： • 命令 • 字段名 • 菜单名 • 按钮
斜体	斜体文本指示： • 要用值来替换的变量参数 • 强调字 • 词汇表术语的首次使用
大写字母	大写字母指示： • 数据类型 • 列名 • 表名
示例	示例文本指示： • 系统信息 • 您输入的值 • 编码示例 • 目录名 • 文件名

- 路径名

如何阅读语法图解

在整本书中，命令和 SQL 语句的语法都是使用语法图解来描述的。

阅读语法图解的方法如下所述:

- 从左到右、从上到下、沿着直线所指的路径阅读语法图。

  ▶▶── 符号表示一条语句的开始。

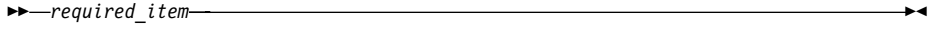
  ──▶ 符号表示语句语法要继续到下一行。

  ▶── 符号表示此语句是从上一行延续的。

  ──▶◀ 符号表示一条语句的结束。

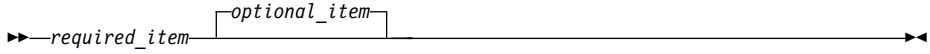
  语法单位的图例不同于其他完整的语句，它以 ▶── 符号开头，以 ──▶ 符号结束。

- 必需的项出现在水平直线（主路径）上。



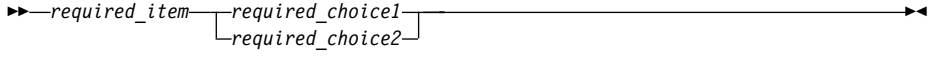
- 可选的项出现在主路径的下面。

  如果某可选项出现在主路径的上面，说明那个项对本语句的执行没有影响，只用于增加可读性。

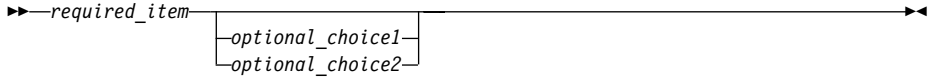


- 若您可从两项或多项中选择，则它们按纵向出现在一个堆栈中。

  若*必须*选择其中的一项，则该堆栈中的一项出现在主路径上。



  若选择其中一项是可选的，则整个堆栈出现在主路径之下。



  如果某项是缺省的，它会出现在主路径上方，而剩余的选项出现在主路径下方。

- 一个箭头返回左边，它出现在主路线的上方，表示这个选项可以被重复。

如果重复箭头中包含了标点符号，则必须使用这个指定的标点符号来分隔重复的项。

堆栈上方的重复箭头表示可以重复堆栈中的项。

- 关键字以大写形式出现（例如，FROM）。在 XML Extender 中，关键字可以大写或小写形式出现。关键字以外的其他术语以小写字母形式出现（例如，*column-name*）。它们表示用户提供的名称或值。
- 如果出现标点符号、括号、算术运算符或其它此类符号，则必须将它们作为语法的一部分输入。

相关信息

当使用 XML Extender 及其相关产品时，下列文档可能会有用:

文档	订单号	描述
Call Level Interface Guide and Reference	SC09–2950	本书描述如何编写使用 CLI 来访问 DB2 服务器的应用程序。
DB2 Application Development Guide	SC09–2949	本书描述应用程序开发过程以及如何编码、编译和执行使用嵌入式 SQL 和 API 来访问数据库的应用程序。
DB2 Extender 页	N/A	此页包含有关 DB2 Extender 的信息，以及与 extender 有关的技术。DB2 Extender 页的 Web 地址为: http://www.software.ibm.com/data/db2/extenders

文档	订单号	描述
DB2 SQL Reference for Universal Database Parts 1 and 2	• 第 1 部分: SC09–2974 • 第 2 部分: SC09–2975	这些书籍描述 SQL 语言的语法、语义和规则。还包括关于发行版间的不兼容性、产品限制和目录视图的信息。
• *DB2 通用数据库管理指南: 实现* • *DB2 通用数据库管理指南: 性能* • *DB2 通用数据库管理指南: 计划*	• 实现: SB84–0218 • 性能: SB84–0243 • 计划: SB84–0219	这些书籍描述如何设计、实现和维护 DB2 数据库。
DB2 通用数据库 Image、Audio 和 Video Extender 管理和程序设计	SB84–0247	本书描述如何管理用于图像、音频和视频数据的 DB2 数据库。它还描述了如何使用由 extender 应用程序设计接口，来访问和处理这些类型的数据。
DB2 通用数据库 Text Extender 管理和程序设计	SB84–0248	本书描述如何管理用于文本数据的 DB2 数据库。它还描述了如何使用由 extender 应用程序设计接口，来访问和处理这些类型的数据。

第1部分 简介

此部分概述了 XML Extender，以及在商务应用程序中如何使用它。

© Copyright IBM Corp. 1999, 2000

第1章 XML Extender 的介绍

IBM® DB2® Extender™ 系列提供了用于处理传统和非传统数据的数据和元数据管理解决方案。 XML Extender 有助于您将 IBM DB2 通用数据库 (DB2 UDB)™ 的功能与 XML 的灵活性集成起来。

DB2 的 XML Extender 提供了存储和访问 XML 文档、或从现存关系数据生成 XML 文档及将 XML 文档分解（分解，存储未标记的元素或属性内容）为关系数据的能力。XML Extender 提供了用来在 DB2 中管理 XML 数据的新数据类型、函数和存储过程。

XML Extender 在下列操作系统上可用:

- Windows NT
- AIX
- Sun Solaris
- Linux
- NUMA-Q

XML 文档

在计算机业界中有许多应用程序，每一个都有它自己的长处和短处。今天，用户有机会选择最适合于他们特定任务需要的应用程序。但是由于用户倾向于在独立的应用程序间共享数据，他们不断地碰到这样的问题: 要以可导入另一应用程序的另一格式来复制、变换、导出或保存数据。这可能是商业应用程序中的关键问题，因为许多这样的变换过程都有可能会丢失一些数据，或至少它们还要求用户完成一个冗长乏味的过程来确保数据的一致性。这既浪费时间又浪费金钱。

今天，已有一种解决此问题的方法，就是由应用程序开发人员来编写*开放式数据库链接 (ODBC)* 应用程序，以将数据保存至数据库管理系统。从此处，可对数据进行处理，且数据被表示为另一应用程序所需的格式。需要编写数据库应用程序，来将数据转换为应用程序所需的格式，但是应用程序变化得很快，随时都会过时。将数据转换为 HTML 的应用程序提供了表示解决方案，但所表示的数据不能实际用于其他用途。如果有另一个方法可使数据与表示内容相分离，则此方法可用作应用程序间的实际交换格式。

XML 的出现解决了此问题。XML 是*可扩展标记语言（eXtensible Markup Language）*的缩写。它是可扩展的，因此该语言本身就是元语言，它允许您根据

© Copyright IBM Corp. 1999, 2000

您的企业需要来创建自己的语言。使用 XML 不仅能从特定的应用程序中捕捉数据，还能捕捉数据结构。XML 不仅仅是交换格式。XML 已渐渐成为数据交换的可接受标准。通过遵循此标准，应用程序最终可共享数据而无需使用专用格式来变换数据。

XML 应用程序

因为 XML 现在是数据交换的可接受的标准，许多现有的应用程序都将能够运用它。

假设您正在使用特定的项目管理应用程序，且想要与您的日历应用程序共享它的某些数据。使用 XML，可以很容易地完成这项任务。在今天这个相互连接的世界里，除非应用程序供应商将 XML 交换实用程序构建到其应用程序内，否则该供应商将不具有竞争力。所以，在此示例中，您的项目管理应用程序可以 XML 格式导出任务，然后，可将这些任务按原样导入到您的日历应用程序中（如果该信息符合某个可接受的 DTD 的话）。

为什么是 XML 和 DB2?

即使 XML 通过提供数据交换的标准格式解决了许多问题，但仍有其他问题有待解决。当构建企业数据应用程序时，您需要回答如下问题:

- 想多久复制一次数据？
- 何种类型的信息需要在应用程序间共享？
- 如何快速搜索需要的信息？
- 如何执行特定的操作，如添加新项，在所有应用程序间触发自动数据交换？

这些类型的问题仅可通过数据库管理系统来解决。通过直接将 XML 信息和元信息合并入数据库，可更直接地（和更快的）获得其他应用程序需要用于特定用途的 XML 结果。这就是 XML Extender 可帮助您的地方。借助 XML Extender，可在许多 XML 应用程序中充分利用 DB2 的能力。

使用 DB2 数据库中的结构化 XML 文档的内容，您可将结构化 XML 信息与传统关系数据结合起来。视应用程序的不同，可选择将整个 XML 文档作为非传统的用户定义数据类型存储在 DB2 中，或可将 XML 内容映射为关系表中的传统数据。对于非传统 XML 数据类型，除 DB2 UDB Text Extender 提供的结构文本搜索之外， XML Extender 还增加了搜索大量 XML 元素或属性值的数据类型的能力。

通过使用 XML Extender，应用程序可:

- 在应用程序表中存储整个 XML 文档作为*列数据*，或在外部以本地文件的形式存储该文档，（在将期望的 XML 元素或属性值抽取至*辅助表*中以进行搜索时。）使用 XML 列方法，可以:
  - 对已经抽取到辅助表中并进行了索引的 SQL 一般*数据类型*的 XML 元素或属性执行快速搜索
  - 更新 *XML 元素*的内容或 *XML 属性*的值
  - 使用 SQL 查询动态地抽取 XML 元素或属性
  - 在插入和更新期间验证 XML 文档
  - 对 Text Extender 执行结构化文本搜索
- 使用 XML 集合存储和访问方法，来组合或分解带有一个或多个关系表的 XML 文档的内容

将 XML 集成到 DB2 中

XML Extender 提供了下列功能部件来帮助您使用 DB2 管理和利用 XML 数据:

- 管理工具，帮助您管理关系表中的 XML 数据的集成
- 存储和使用 XML 数据的方法。
- DTD 库 (DTD_REF)，供您存储用于验证 XML 数据的 DTD
- 一个称为“文档访问定义”(DAD) 文件的映射模式，用于将 XML 文档映射为关系数据

管理工具

XML Extender 管理工具帮助您对 XML 启用数据库和表列，并将 XML 数据映射至 DB2 关系结构。XML Extender 提供了几个管理工具供您使用，这取决于您是否想开发一个应用程序来执行管理任务，或是否只是想使用向导。您可使用下列工具以完成 XML Extender 的管理任务:

- XML Extender 管理向导提供了用于管理任务的图形用户界面。
- **dxxadm** 命令提供了用于管理任务的命令行选项。
- XML Extender 管理存储过程提供了用于管理任务的应用程序开发选项。

存储和访问方法

XML Extender 为将 XML 文档集成到 DB2 中提供了两种存储和访问方法: XML 列和 XML 集合。这两种方法的使用很不相同，但可在同一应用程序中使用。

XML 列

此方法帮助您将完整的 XML 文档存储在 DB2 中。对于归档文档， XML 列非常有用。这些文档被插入到对 XML 启用的列中，且可对这些文档进

行更新、检索和搜索。元素和属性数据可被映射至 DB2 表（辅助表），可依次对它们进行索引以便进行快速的结构化搜索。

XML 集合

此方法帮助您将 XML 文档结构映射至 DB2 表，以使您可从现存 DB2 数据组合 XML 文档，或将 XML 文档分解（存储未标记的元素或属性内容）为 DB2 数据。对于数据交换应用程序，此方法非常有用，尤其是在 XML 文档的内容频繁更新时。

DTD 库

XML Extender 提供了 XML *文档类型定义 (DTD) 库*，它是对 XML 元素和属性的一组说明。当对 XML *启用*数据库时，亦创建了 *DTD 参考表 (DTD_REF)*。此表的每一行都表示一个具有附加元数据信息的 DTD。用户可访问此表以插入他们自己的 DTD。DTD_REF 表中的 *DTD* 用来验证 XML 文档。

文档访问定义 (DAD)

指定在*文档访问定义 (DAD)* 中处理结构化 XML 文档的方式。DAD 本身就是一种 XML 格式化文档。它在使用 XML 列或 XML 集合时将 XML 文档结构与 DB2 数据库相关联。与 XML 集合相反，在定义 XML 列时，DAD 的结构是不同的。

DAD 文件是使用 XML_USAGE 表进行管理的，且是在对 XML 启用数据库时创建的。

XML 列: 结构化文档存储和检索

因为 XML 包含创建一组文档必需的所有信息，所以有些时候想要将文档结构存储并保留为它当前的状态。

例如，如果您是一个通过 Web 发表文章的新闻发布公司，可能会想要维护已发布的文章的档案。在这样的方案中， XML Extender 使您可以在 DB2 表的列中存储完整的或部分的 XML 文章。此类型的 XML 文档存储器称为 *XML 列*， 如第7页的图1中所示。

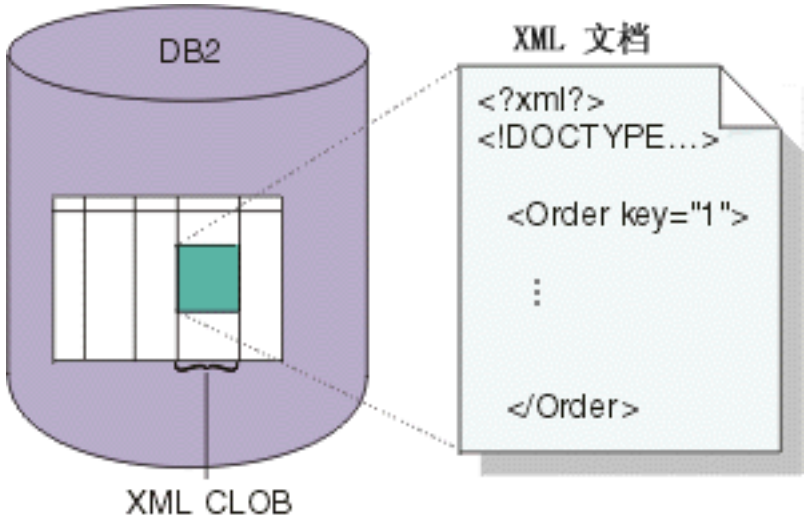


图 1. 在 DB2 表列中存储结构化的 XML 文档

XML Extender 提供了下列用户定义类型 (UDT)，以与 XML 列配合使用:

- XMLVarchar
- XMLCLOB
- XMLFILE

所有 XML Extender 的 UDT 都有前缀 db2xml，它是 DB2 XML Extender UDT 的*模式名*。这些数据类型用于标识应用程序表中 XML 文档的存储类型。XML Extender 支持传统的普通文本文件；不需要将 XML 文档存储在 DB2 中。还可将 XML 文档以文件形式存储在*本地文件系统*上，该文件系统由本地文件名指定。

DB2 XML Extender 提供了功能强大的*用户定义函数 (UDF)*，以在 XML 列中存储和检索 XML 文档，以及抽取 XML 元素或属性值。 UDF 是对数据库管理系统定义的函数，且此后可在 SQL 查询中引用。XML Extender 提供了下列类型的 UDF:

- 存储: 将完整的 XML 文档以 XML 数据类型的形式存储在启用了 XML 的列中
- 抽取: 抽取 XML 文档，或抽取将元素和属性指定为基本数据类型的值
- 更新: 更新整个 XML 文档或指定的元素和属性值

抽取功能允许您对一般 SQL 数据类型执行功能强大的搜索。此外，还可将 DB2 UDB Text Extender 与 XML Extender 配合使用，以对 XML 文档中的文本执行结构化和全文本*搜索*。例如，这种强大的检索能力可用于（比如说）改进 Web 站

点的可用性，该 Web 站点发布大量可读文本（例如新闻文章）或*电子数据交换 (EDI)* 应用，它们具有可频繁搜索的元素或属性。

所有 XML Extender 的 UDF 都有前缀 db2xml，它是 DB2 XML Extender UDF 的模式名。这些 UDF 适用于 XML UDT，而 XML UDT 用于 XML 列。

位置路径

*位置路径*是标识 XML 元素或属性的 *XML 标记*序列。XML Extender 使用位置路径来标识 XML 文档的结构，并指示元素或属性的上下文。单个斜杠 (/) 路径指示上下文是整个文档。位置路径在下列情况中使用:

- 要在抽取 UDF 时，标识要抽取的元素和属性
- 要对 XML 列定义 DAD 中的索引模式时，指定 XML 元素或属性与 DB2 列之间的映射文件
- 要在使用 Text Extender 进行结构化文本搜索时，标识 XML 元素或属性

图2显示了位置路径及它与 XML 文档结构之间的关系的示例

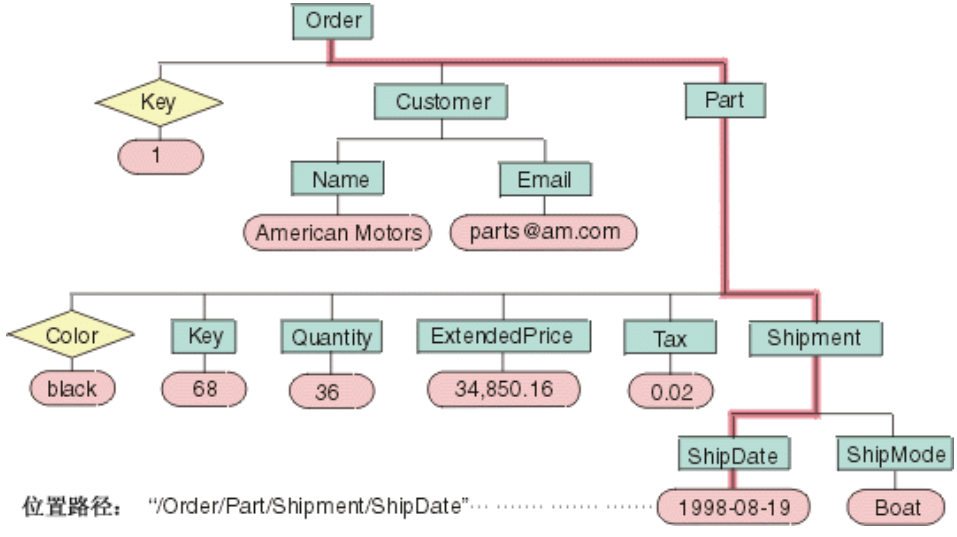


图 2. 将文档作为结构化 XML 文档存储在 DB2 表列中

要指定位置路径，XML Extender 使用 *XML 样式表语言变换 (XSLT)* 和 *XML 路径语言 (XPath)* 的一个子组。本书使用了术语*位置路径*，它是在 XPath 的说明中定义的。位置路径是标识 XML 元素或属性的一个 XML 标记序列。本书还使用了*绝对位置路径*的 XSLT 或 XPath 缩略语法，它是在 XPath 说明中指定的。绝对位置路径是对象的全路径名。

XSLT 是将 XML 文档变换为其他 XML 文档的语言。它是为用作 *XML 样式表语言 (XSL)* 的一部分而设计的，该 XSL 是 XML 的样式表语言。除 XSLT 之外，XSL 还包括 XML 用于指定格式化的词汇表。通过使用 XSLT 来描述一个 XML 文档如何变换为另一使用格式化词汇表的 XML 文档， XSL 可以指定 XML 文档的样式。

XPath 是用于 XML 文档的寻址部分的一种语言，供 XSLT 使用。每个位置路径都可用为 XPath 定义的语法来表示。

有关 XSLT 和 XPath 的详情，参见下列 Web 页:

- 有关 XSLT，参见: http://www.w3.org/TR/WD-xslt
- 有关 XPath，参见: http://www.w3.org/TR/xpath

参见第48页的『位置路径』以了解语法和限制。

XML 列术语

本节描述本书中引用的 XML 概念和术语。

文档访问定义 (DAD)
: 对于 XML 列而言，是指将 XML 文档结构映射为 DB2 辅助表，并对这种映射建立索引以用于结构化查询。

DXX_INSTALL
: XML Extender 安装目录。

辅助表 是 XML Extender 创建的附加表，用以在搜索 XML 列中的元素或属性时提高性能。

XML 列
: 通过对 XML 数据类型启用 DB2 列并将完整的 XML 文档存储在启用的列中，来存储和访问 XML 文档的一种方法。亦指已使用其中一个管理工具对 XML 启用的列。

XML 表
: 一个应用程序表，它包括已使用其中一个管理工具对 XML 启用的一列或多列。

XML UDF
: 由 XML Extender 提供的 DB2 用户定义函数。

XML UDT
: 由 XML Extender 提供的 DB2 用户定义类型。

XML 集合: 集成数据管理

传统 SQL 数据是从输入的 XML 文档*分解*出来的，或用于*组合*输出的 XML 文档。如果数据与其他应用程序一起共享，则您可能想要能够组合或分解输入的和输出的 XML 文档，并在必要时管理数据以利用 DB2 的关系能力。此类型的 XML 文档存储器称为 *XML 集合*。

XML 集合的示例显示在图3中。

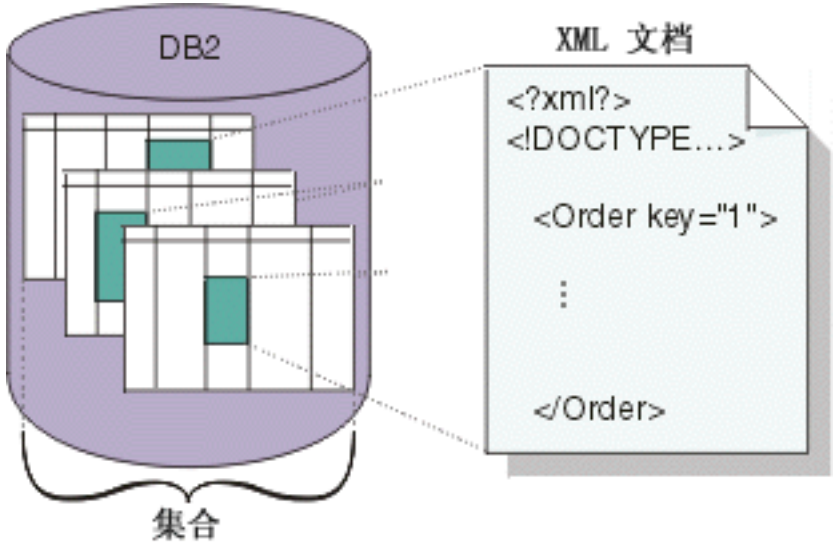


图 3. 将文档作为未标记的数据存储在 DB2 表中

XML 集合是在 DAD 文件中定义的，它指定元素和属性映射至一个或多个关系表的方式。可通过启用集合名来定义它，然后将其与存储过程配合使用来组合或分解 XML 文档。

定义 DAD 文件中的集合时，使用两种类型的映射模式 （SQL 映射或 RDB_node 映射）其中一种。SQL 映射使用 SQL SELECT 语句来定义用于集合的 DB2 表和条件。RDB_node 映射使用基于 XPath 的 RDB_node 来定义表、列和条件。

提供了存储过程来组合或分解 XML 文档。存储过程使用前缀 db2xml，它是 XML Extender 的*模式名*。将下列存储过程与 XML 集合配合使用:

- 组合:
  - dxxGenXML(): 使用 XML 集合的 DAD 文件来组合 XML 文档
  - dxxRetrieveXML(): 使用启用的 XML 集合来组合 XML 文档
- 分解:

– dxxShredXML(): 使用 XML 集合的 DAD 文件来分解 XML 文档
– dxxInsertXML(): 使用启用的 XML 集合来分解 XML 文档

XML 集合术语

下列术语对于 XML Extender 都是唯一的，且在本书中频繁使用。

组合 按 DAD 文件中定义的那样从现存关系数据生成 XML 文档。

分解 按 DAD 文件中定义的那样来将 XML 文档存储为未标记的关系数据。

文档访问定义 (DAD)
对于 XML 集合，是 XML 文档结构至 DB2 数据结构的映射，用于组合或分解 XML 文档。

DXX_INSTALL
XML Extender 安装目录。

XML 集合
使用关系表存储和访问 XML 数据的一种方法。未标记的数据可组合成 XML 文档，也可从 XML 文档分解出来。它还引用一组表， XML 文档可由这些表组合或从这些表中分解出来。

XML 存储过程
用来组合或分解 XML 文档的存储过程。

第2章 XML Extender 入门

本章向您传授如何使用 XML Extender 来访问和修改应用程序的 XML 数据的入门知识。遵循所提供的教程的指导，您就可以使用所提供的样本数据来建立数据库、将 SQL 数据映射至 XML 文档、在数据库中存储 XML 文档，然后搜索 XML 文档以及从 XML 文档中抽取数据。

在管理课程中，将 DB2 命令窗口与管理命令配合使用。利用 XML Extender 管理向导可以完成这些任务，本书中也描述了该向导。 在 XML 数据管理课程中，您将使用 XML Extender 所提供的 UDF 和存储过程。 本书其余章节中的大多数示例都采用本章中所使用的样本数据。

要求: 要完成本章中的课程，必须具有 DB2 UDB 版本 6.1 或更高版本。如果使用 “版本 6.1”，则必须安装 “修订包 2”。此外，这些课程中的步骤假定您正在使用 Windows NT®。

课程如下所示:

- 将完整的 XML 文档存储在 DB2 表列中
  - 计划以其存储文档和将频繁搜索的 XML 元素和属性的 XML UDT。
  - 建立数据库和表
  - 为 XML 启用数据库
  - 将 DTD 插入 DTD 库表
  - 为 XML 列准备 DAD
  - 向 XML 类型的现存表中添加一列
  - 为 XML 启用新列
  - 创建辅助表的索引
  - 将 XML 文档存储在 XML 列中
  - 使用 XML Extender UDF 来搜索 XML 列
- 根据现存数据创建 XML 文档
  - 计划 XML 文档的数据结构
  - 建立数据库和表
  - 为 XML 启用数据库
  - 为 XML 集合准备文档访问定义 (DAD) 文件
  - 根据现存数据组合 XML 文档

© Copyright IBM Corp. 1999, 2000

- 从数据库检索 XML 文档
- 清除数据库

课程的方案

在这些课程中，您为 ACME Auto Direct 工作， ACME Auto Direct 是一个将小汽车和卡车分发给汽车零售商的公司。您的任务有两项。首先，您将安装一个系统，在该系统中，可将订单归档在 SALES_DB 数据库中，以供销售部门查询。第二个任务是在现存的采购订单数据库 SALES_DB 中采集信息，并从中抽取要存储在 XML 文档中的关键信息。

课程: 将 XML 文档存储在 XML 列中

方案

您的任务是将销售数据归档以供服务部门使用。这些数据存储在使用同一 DTD 的 XML 文档中。服务部门将在处理客户请求和意见时使用这些 XML 文档。

服务部门已经提供了 XML 文档的建议结构，并指定了他们认为将会被最频繁地查询的元素数据。他们希望 XML 文档存储在 SALES_TAB 数据库的 SALES_DB 表中，且能够迅速的搜索它们。SALES_DB 表有两列包含有关每项销售的数据，第三列则包含 XML 文档。此列称为 ORDER。

您将确定以其存储 XML 文档的 XML 数据类型，以及哪些 XML 元素将会被频繁查询。下一步，将为 XML 安装 SALES_DB 数据库，创建 SALES_TAB 表，并启用 ORDER 列以使您可将完整的文档存储在 DB2 中。您还将插入 XML 文档的 DTD 以进行验证，并将文档以 XMLVARCHAR 数据类型的形式存储。启用该列时，将为在文档访问定义 (DAD) 文件中文档的结构搜索定义要进行索引的辅助表，该文档访问定义 (DAD) 文件是一个指定辅助表的结构的 XML 文档。要查看 DAD 文件、DTD 和 XML 文档的样本，参见第239页的『附录B. 样本』。

SALES_TAB 在表1中作了描述。

表 1. SALES_TAB 表

列名	数据类型
INVOICE_NUM	CHAR(6) NOT NULL PRIMARY KEY
SALES_PERSON	VARCHAR(20)
ORDER	XMLVARCHAR

计划

在开始使用 XML Extender 存储文档之前，需要了解 XML 文档的结构，以使您可确定如何搜索该文档。在计划搜索文档的方式时，需要确定:

- 将以其存储 XML 文档的 XML 用户定义类型
- 服务部门将频繁搜索的 XML 元素和属性，以便可对它们进行索引以改进性能。

下列各节将描述如何作出这些决策。

XML 文档结构

本课程的 XML 文档结构采集某个特定订单的信息，该订单是以订单键作为顶级，客户、部件和交付信息作为下一级来构造的。XML 文档在第16页的图4中作了描述。

此课程还提供了样本 DTD，供您在了解和验证 XML 文档结构时使用。可以在第239页的『附录B. 样本』中看到该 DTD 文件。它与第16页的图4中的结构相匹配。

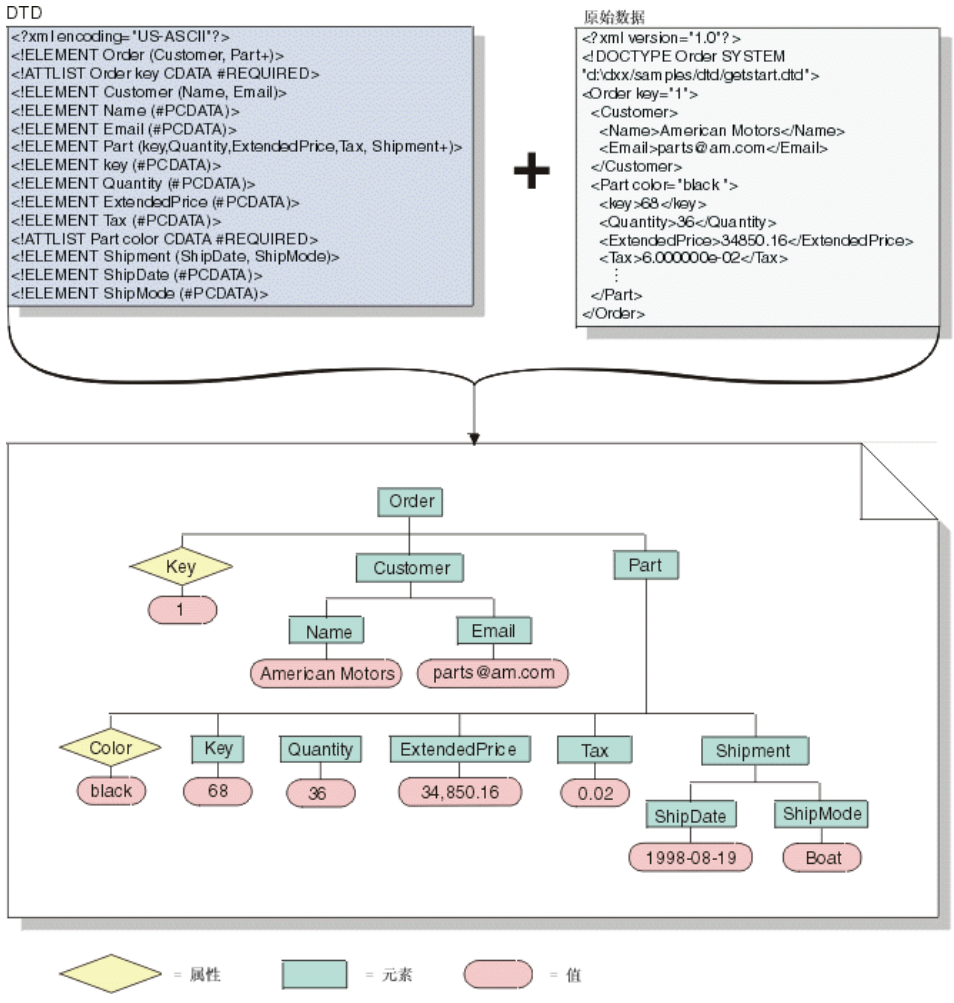


图 4. DTD 和 XML 文档的层次结构

确定 XML 列的 XML 数据类型

XML Extender 提供了以其定义要保存 XML 文档的列的 XML 用户数据类型。这些数据类型为:

- XMLVarchar: 用于存储在 DB2 中的小型文档
- XMLCLOB: 用于存储在 DB2 中的大型文档
- XMLFILE: 用于存储在 DB2 之外的文档

在本课程中，会将小型文档存储在 DB2 中，从而将使用 XMLVarchar 数据类型。

确定要搜索的元素和属性

当了解 XML 文档结构和应用程序的需求时，就可确定要搜索的元素和属性，将会被最频繁地搜索或抽取的元素和属性，或查询成本将为最高的元素或属性。服务部门已经指示他们将会频繁地查询订单的订单键、客户姓名、价格和交付日期，并需要快速进行这些搜索。此信息包含在 XML 文档结构的元素和属性中。 表2描述了每个元素和属性的位置路径。

表 2. 要搜索的元素和属性

数据	位置路径
order_key	/Order/@key
customer	/Order/Customer/Name
price	/Order/Part/ExtendedPrice
shipping_date	/Order/Part/Shipment/ShipDate

将 XML 文档映射至辅助表

在此教程中，您将为 XML 列创建一个 DAD 文件，用来将 XML 文档存储在 DB2 中。它还将 XML 元素和属性内容映射至用于进行索引的 DB2 辅助表，这样可改进搜索性能。在上一节中，您了解到了要搜索的元素和属性。在此节中，您将了解有关将这些元素和属性值映射至可进行索引的 DB2 表的详情。

在找出要搜索的元素和属性之后，确定应在辅助表中组织它们的方式，有多少个表及哪个表中有哪些列。通常，通过将相似的信息放入同一表中来组织辅助表。该结构还由任何元素的位置路径是否可在该文档中重复多次来确定。例如，在本文档中，部件元素可重复多次，从而价格和日期元素也可出现多次。元素的多次出现必须是在它们自己的表中进行的。

此外，还需要确定元素或属性值应使用哪些 DB2 基本类型。通常，这可根据数据的格式来非常容易地确定。如果数据为文本，则选择 VARCHAR；如果数据为整数，则选择 INTEGER；或如果数据为日期且您想要进行范围搜索，则选择 DATE。

在此教程中，元素和属性被映射至下列辅助表:

ORDER_SIDE_TAB

列名	数据类型	位置路径	多次出现吗?
ORDER_KEY	INTEGER	/Order/@key	否
CUSTOMER	VARCHAR(16)	/Order/Customer/Name	否

PART_SIDE_TAB

列名	数据类型	位置路径	多次出现吗?
PRICE	DECIMAL(10,2)	/Order/Part/ExtendedPrice	是

SHIP_SIDE_TAB

列名	数据类型	位置路径	多次出现吗?
DATE	DATE	/Order/Part/Shipment/ShipDate	是

对于此教程，我们提供了一组脚本，以便您用来设置您的环境。这些脚本都在 *DXX_INSTALL*\samples\cmd 目录（其中 *DXX_INSTALL* 是安装有 XML Extender 的驱动器和目录，例如 c:\dxx\samples\cmd）中，下面列示了它们:

getstart_db.cmd
创建数据库并填充四个表。

getstart_prep.cmd
将数据库与 XML Extender 存储过程和 DB2 CLI 绑定。

getstart_insertDTD.cmd
插入用来验证 XML 列中的 XML 文档的 DTD。

getstart_createTabCol.cmd
创建将具有启用了 XML 的列的应用程序表。

getstart_alterTabCol.cmd
通过添加将对 XML 启用的列来改变应用程序表。

getstart_enableCol.cmd
启用 XML 列。

getstart_createIndex.cmd
对 XML 列的辅助表创建索引。

getstart_insertXML.cmd
将 XML 文档插入到 XML 列中。

getstart_queryCol.cmd
对应用程序表运行选择语句，并返回 XML 文档。

getstart_clean.cmd
清除教程环境。

设置

在本节中，准备数据库以与 XML Extender 配合使用。您将:

1. 创建数据库。
2. 启用数据库。

创建数据库

在本节中，使用一个命令来建立数据库。此命令将创建样本数据库，与它进行连接，创建用来保存数据的表，然后插入数据。

要创建该数据库:

1. 更改为 *DXX_INSTALL*\samples\cmd 目录，其中，*DXX_INSTALL* 是安装 XML Extender 的驱动器和目录。在这些课程中采用 c:\dxx 目录。如果您的驱动器和目录不同，则更改这些值。
2. 从 Windows NT “开始” 菜单打开 “DB2 命令窗口”，或从 Windows NT 命令提示处输入以下命令:

   ```
   DB2CMD
   ```

3. 从 “DB2 命令窗口”，运行以下命令:

   ```
   getstart_db.cmd
   ```

启用数据库

要在数据库中存储 XML 信息，需要为 XML Extender 启用该数据库。当为 XML 启用数据库时，XML Extender 会:

- 创建所有用户定义类型 (UDT) 和用户定义函数 (UDF)。
- 创建控制表，并用 XML Extender 所需要的元数据来填充该控制表。
- 创建 db2xml 模式并分配必需的特权。

要为 XML 启用数据库:

从 “DB2 命令窗口”，运行以下脚本以启用 SALES_DB 数据库:

```
getstart_prep.cmd
```

此脚本将数据库与 XML Extender 存储过程和 DB2 CLI 进行绑定。 它还运行用来启用数据库的 **dxxadm** 命令选项:

```
dxxadm enable_db SALES_DB
```

创建 XML 列

XML Extender 提供了在数据库中存储和访问整个 XML 文档的方法，该方法称为 XML 列。通过使用 XML 列方法，可使用 XMLFILE 文件类型来存储该文档，对

辅助表中的列进行索引，然后查询或搜索 XML 文档。此存储方法对于归档应用程序特别有用，在这类应用程序中不会频繁更新文档。为在此教程中使用，将所提供的 XML 文档存储在 XML 列中。

在本课程中，将在 SALES_TAB 表中存储文档。要存储文档，您将:

1. 将 XML 文档的 DTD 插入到 DTD 参考表 DTD_REF 中。
2. 准备 DAD 文件，它指定 XML 文档位置和辅助表以进行结构搜索。
3. 在具有 XML 用户定义类型 XMLVARCHAR 的 SALES_TAB 表中添加一列。
4. 为 XML 启用列。
5. 对辅助表进行索引以进行结构搜索。
6. 使用 XML Extender 提供的用户定义函数来存储文档。

将 DTD 存储在 DTD 库中

可使用 DTD 来验证 XML 列中的 XML 数据。XML Extender 在启用了 XML 的数据库中创建一个表，该表称为 DTD_REF。该表即为 DTD 参考表，可供您存储 DTD 使用。如果决定验证 XML 文档，则您必须将 DTD 存储在此库中。此教程的 DTD 为 `c:\dxx\samples\dtd\getstart.dtd`。

要插入 DTD:

从“DB2 命令窗口”，在同一行上输入以下 SQL INSERT 命令:

```
DB2 CONNECT TO SALES_DB
DB2 INSERT into db2xml.dtd_ref values('c:\dxx\samples\dtd\getstart.dtd',
    db2xml.XMLClobFromFile('c:\dxx\samples\dtd\getstart.dtd'), 0, 'user1',
    'user1', 'user1')
```

您还可运行以下命令文件来插入 DTD:

```
getstart_insertDTD.cmd
```

准备 DAD 文件

XML 列的 DAD 文件具有简单的结构。指定存储模式为 XML 列，并定义要进行索引的表和列。

在下列步骤中，将 DAD 中的元素称为*标记*，将 XML 文档结构的元素称为*元素*。类似于您将创建的 DAD 文件的一个 DAD 文件样本在 `c:\dxx\samples\dad\getstart_xcolumn.dad` 中。它与在下列步骤中生成的文件稍有差别。若将它用于本课程，注意，文件路径可能与您所在环境的文件路径不同， <validation> 值应设置为“否”，而不是“是”。

要准备 DAD 文件:

1. 打开文本编辑器并将文件命名为 `getstart_xcolumn.dad`

注意，DAD 文件中使用的所有标记都是区分大小写的。

2. 创建带有 XML 和 Doctype 说明的 DAD 头。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:\dxx\dtd\dad.dtd">
```

DAD 文件为 XML 文档，需要 XML 说明。

3. 插入打开和关闭 <DAD></DAD> 标记。其他所有标记都位于这些标记内。
4. 插入打开和关闭 <DTDID></DTDID> 标记来指定 DTD ID 标识符，该标识符将 DAD 与 XML 文档 DTD 相关联并指定 DTD 在客户机上的位置。

```
<dtdid>c:\dxx\samples\dtd\getstart.dtd</dtdid>
```

验证此字符串是否与在将 DTD 插入到第20页的『将 DTD 存储在 DTD 库中』的 DTD 参考表中时用作首个参数值的值相匹配。例如，用于 DTDID 的路径可能与上述字符串不同（如果在另一机器的驱动器上工作的话）。

5. 指定打开和关闭 <validation></validation> 标记，以指示 XML Extender 将使用您插入到 DTD 库表中的 DTD 来验证 XML 文档结构。

```
<validation>YES</validation>
```

<validation> 的值必须采用大写字母。

6. 插入打开和关闭 <Xcolumn></Xcolumn> 标记，以将存储方法定义为 XML 列。该方法定义：要将 XML 数据存储在 XML 列中。

```
<Xcolumn>
</Xcolumn>
```

7. 为要生成的每个辅助表插入打开和关闭 <table></table> 标记。

```
<Xcolumn>
     <table name="order_side_tab">
</table>
     <table name="part_side_tab">
</table>
     <table name="ship_side_tab">
</table>
</Xcolumn>
```

8. 为要包括在这些辅助表中的每一列插入打开和关闭 <column></column> 标记。每个 <column> 标记具有四种属性：
   - **名称**：列的名称
   - **类型**：列的数据类型
   - **路径**：XML 文档中对应元素的位置路径（使用 XPath 语法）。参见第48页的『位置路径』以了解位置路径语法。
   - **多次出现**：指示元素的位置路径在 XML 文档结构中可否出现多次

```
<Xcolumn>
     <table name="order_side_tab">
  <column name="order_key"
          type="integer"
          path="/Order/@key"
          multi_occurrence="NO"/>
  <column name="customer"
          type="varchar(50)"
          path="/Order/Customer/Name"
          multi_occurrence="NO"/>
</table>
     <table name="part_side_tab">
  <column name="price"
          type="decimal(10,2)"
          path="/Order/Part/ExtendedPrice"
          multi_occurrence="YES"/>
</table>
     <table name="ship_side_tab">
  <column name="date"
          type="DATE"
          path="/Order/Part/Shipment/ShipDate"
          multi_occurrence="YES"/>
</table>
</Xcolumn>
```

9. 必须确保在最后一个 </table> 标记之后，有关闭 </Xcolumn>。
10. 必须确保在 </Xcolumn> 标记之后，有关闭 </DAD>。
11. 将文件另存为 getstart_xcolumn.dad。

可将刚刚创建的文件与样本文件 c:\dxx\samples\dad\getstart_xcolumn.dad 进行比较。此文件是启用 XML 列和创建辅助表所必需的 DAD 文件的工作副本。样本文件包含路径语句，可能需要更改这些语句以与您的环境相匹配，以便成功运行。

创建 SALES_TAB 表

在本节中要创建 SALES_TAB 表。最初，该表有两列中包含了关于订购的销售信息。

要创建该表:

从 “DB2 命令窗口”，输入以下 CREATE TABLE 语句:

```
DB2 CONNECT TO SALES_DB

DB2 CREATE TABLE SALES_TAB(INVOICE_NUM CHAR(6) NOT NULL PRIMARY KEY,
        SALES_PERSON VARCHAR(20))
```

此外，您还可运行以下命令文件来创建该表:

```
getstart_createTabCol.cmd
```

添加 XML 类型的列

现在，向 SALES_TAB 表中添加一个新列。此列将包含较早生成的完整的 XML 文档，并且必须是 XML UDT。 XML Extender 提供了 第149页的『第8章 XML Extender 用户定义类型』中所描述的多种数据类型。在此教程中，将以 XMLVARCHAR 的形式存储该文档。

要添加 XML 类型的列:

从 “DB2 命令窗口”，输入以下 SQL 语句:

```
DB2 ALTER TABLE SALES_TAB ADD ORDER DB2XML.XMLVARCHAR
```

此外，您还可运行以下命令文件来改变该表:

```
getstart_alterTabCol.cmd
```

启用 XML 列

在创建 XML 类型的列之后，为 XML Extender 启用该列。启用该列时，XML Extender 读取 DAD 文件并创建辅助表。启用该列之前，必须:

- 确定是否想要创建 XML 列（它包含 XML 文档）和辅助表列的缺省视图。可在查询 XML 文档时指定缺省视图。在本课程中，将使用 `-v` 参数来指定该视图。
- 确定是否想要将主键指定为 *ROOT 标识*，即应用程序表中主键的列名，且它是将所有辅助表与应用程序表相关联的唯一标识符。如果未指定主键，则 XML Extender 将 DXXROOT_ID 列添加至应用程序表和辅助表。ROOT_ID 列将应用程序表和辅助表紧密联系在一起，并允许 XML Extender 自动更新辅助表（如果 XML 文档被更新的话）。在本课程中，将使用 `-r` 参数来在命令 (INVOICE_NUM) 中指定主键的名称。然后， XML Extender 将指定列用作 ROOT_ID 并将该列添加至辅助表。
- 确定是想要指定表空间还是使用缺省表空间。在本课程中，将使用缺省表空间。

要为 XML 启用列:

从 “DB2 命令窗口”，输入以下命令:

```
dxxadm enable_column SALES_DB SALES_TAB ORDER GETSTART_XCOLUMN.DAD
       -v SALES_ORDER_VIEW -r INVOICE_NUM
```

此外，您还可运行以下命令文件来对 XML 启用该列:

```
getstart_enableCol.cmd
```

XML Extender 创建了带有 INVOICE_NUM 列的辅助表，并创建了缺省视图。

重要事项: 不要以任何方式修改辅助表。您应仅使用由 XML Extender 提供的 UDF 来更新 XML 文档。XML Extender 将在更新 XML 列中的 XML 文档时自动更新辅助表。

查看列和辅助表

启用了 XML 列后，就为该 XML 列和辅助表创建了视图。 当处理 XML 列时，可以使用此视图。

要查看 XML 列和辅助表列:

从 “DB2 命令窗口”，输入以下 SQL SELECT 语句:

```
DB2 SELECT * FROM SALES_ORDER_VIEW
```

该视图显示了辅助表中的各列，如 `getstart_xcolumn.dad` 文件中指定的那样。

对辅助表创建索引

对辅助表创建索引允许您对 XML 文档执行快速结构搜索。在此步骤中，将会对在启用 XML 列 ORDER 时创建的辅助表中的键列创建索引。服务部门已经指定了他们的雇员很可能最经常查询的列。 表3描述了这些列，您将对它们进行索引:

表 3. 要进行索引的辅助表列

列	辅助表
ORDER_KEY	ORDER_SIDE_TAB
CUSTOMER	ORDER_SIDE_TAB
PRICE	PART_SIDE_TAB
DATE	SHIP_SIDE_TAB

要对辅助表进行索引:

从 “DB2 命令窗口”，输入下列 SQL 命令:

```
DB2 CREATE INDEX KEY_IDX
      ON ORDER_SIDE_TAB(ORDER_KEY)

DB2 CREATE INDEX CUSTOMER_IDX
      ON ORDER_SIDE_TAB(CUSTOMER)

DB2 CREATE INDEX PRICE_IDX
      ON PART_SIDE_TAB(PRICE)

DB2 CREATE INDEX DATE_IDX
      ON SHIP_SIDE_TAB(DATE)
```

此外，您还可运行以下命令文件来创建索引:

```
getstart_createIndex.cmd
```

存储 XML 文档

既然您启用了可包含 XML 文档的列并对辅助表进行了索引，则可使用 XML Extender 提供的函数来存储该文档。在将数据存储到 XML 列中时，可使用缺省强制转型函数，或者使用 XML Extender UDF。 因为您将把一个基本类型 VARCHAR 的对象存储到类型为 XML UDT XMLVARCHAR 的一列中，所以，您将使用缺省强制转型函数。 有关存储器缺省强制转型函数和 XML Extender 提供的 UDF 的详情，参见第106页的『存储数据』。

要存储 XML 文档:

重要事项: 打开 XML 文档 `c:\dxx\samples\xml\getstart.xml`。确保 DOCTYPE 中的文件路径与 DAD 中指定的 DTD ID 以及在 DTD 库中插入 DTD 时指定的 DTD ID 相匹配。可通过查询 db2xml.DTD_REF 表或检查 DAD 文件中的 DTDID 元素，验证它们是否匹配。如果使用的不是缺省驱动器和目录，可能需要更改 DOCTYPE 说明中的路径。

从 “DB2 命令窗口”，输入以下 SQL INSERT 命令:

```
DB2 INSERT INTO SALES_TAB (INVOICE_NUM, SALES_PERSON, ORDER) VALUES('123456',
   'Sriram Srinivasan', db2xml.XMLVarcharFromFile('c:\dxx\samples\cmd\getstart.xml'))
```

当存储 XML 文档时，XML Extender 将自动更新辅助表。

此外，可运行以下命令文件来存储该文档:

```
getstart_insertXML.cmd
```

要验证是否已更新这些表，在 “DB2 命令窗口” 中对这些表运行下列 SELECT 语句:

```
DB2 SELECT * FROM SALES_TAB

DB2 SELECT * FROM PART_SIDE_TAB

DB2 SELECT * FROM ORDER_SIDE_TAB

DB2 SELECT * FROM SHIP_SIDE_TAB
```

搜索 XML 文档

可通过直接查询辅助表来搜索 XML 文档。在此步骤中，将搜索价格高于 2500.00 的所有订单。

要查询辅助表:

从 “DB2 命令窗口”，输入以下 SELECT 语句:

```
DB2 "SELECT DISTINCT SALES_PERSON FROM SALES_TAB S, PART_SIDE_TAB P
   WHERE PRICE > 2500.00 AND
   S.INVOICE_NUM=P.INVOICE_NUM"
```

结果集应该显示销售了价格高于 2500.00 的商品的销售人员名单。

此外，可运行以下命令文件来搜索该文档:

```
getstart_queryCol.cmd
```

您已经完成了将 XML 文档存储在 DB2 表中的入门教程。本书中的许多示例都是根据这些课程来编写的。

课程: 组合 XML 文档

教程方案

您的任务是采集现存的采购单数据库 SALES_DB 中的信息，并从该数据库中抽取要存储在 XML 文档中的关键信息。然后，当服务部处理客户请求和意见时将使用这些 XML 文档。 服务部已经要求能包括特定数据，并提供了一个 XML 文档的建议性结构。

通过使用现存数据，您将根据这些表中的数据组合 XML 文档 `getstart.xml`。

您还将计划和创建 DAD 文件，该文件将相关表中的列映射至 XML 文档结构，该文档结构提供了采购单记录。因为此文档是由多个表组合的，所以您将创建 XML 集合，将这些表与 XML 结构和 DTD 关联起来。使用此 DTD 来定义 XML 文档的结构。还可用它在应用程序中验证组合的 XML 文档。

下表中描述了 XML 文档中的现存数据库数据。以*斜体*显示的列名都是服务部要求加在 XML 文档结构中的列。

ORDER_TAB

列名	数据类型
ORDER_KEY	INTEGER
CUSTOMER	VARCHAR(16)
CUSTOMER_NAME	VARCHAR(16)
CUSTOMER_EMAIL	VARCHAR(16)

PART_TAB

列名	数据类型
PART_KEY	INTEGER
COLOR	CHAR(6)
QUANTITY	INTEGER
PRICE	DECIMAL(10,2)
TAX	REAL
ORDER_KEY	INTEGER

SHIP_TAB

列名	数据类型
DATE	DATE
MODE	CHAR(6)
COMMENT	VARCHAR(128)
PART_KEY	INTEGER

计划

在开始使用 XML Extender 来组合文档之前，需要确定 XML 文档的结构，以及它与数据库数据的结构的对应关系。本节将概述服务部门所要求的 XML文档结构，用来定义 XML 文档的结构的 DTD，以及此文档映射至包含某些数据（这些数据用来填充这些文档）的列的方式。

确定文档结构

XML 文档结构从多个表中采集关于某个特定订单的信息，并为该订单创建一个 XML 文档。这些表中的每个表都包含有关该订单的相关信息，并且可以在它们的键列上进行连接。服务部门想要这样的文档，它以订单号作为顶级，以客户、部件和交付信息作为下一级来构造的。他们希望文档结构是直观和灵活的，利用元素来描述数据，而不是文档结构。（例如，应该将客户的姓名放入称为 'customer' 的元素中，而不是在一个段落中。）根据他们的请求， DTD 和 XML 文档的层次结构应类似于第28页的图5中所述的内容。

设计了文档结构之后，您应创建一个 DTD 来描述 XML 文档的结构。此教程为您提供了一个 XML 文档和 DTD。可以在第239页的『附录B. 样本』中看到该 DTD 文件。您会发现它符合第28页的图5中的结构。

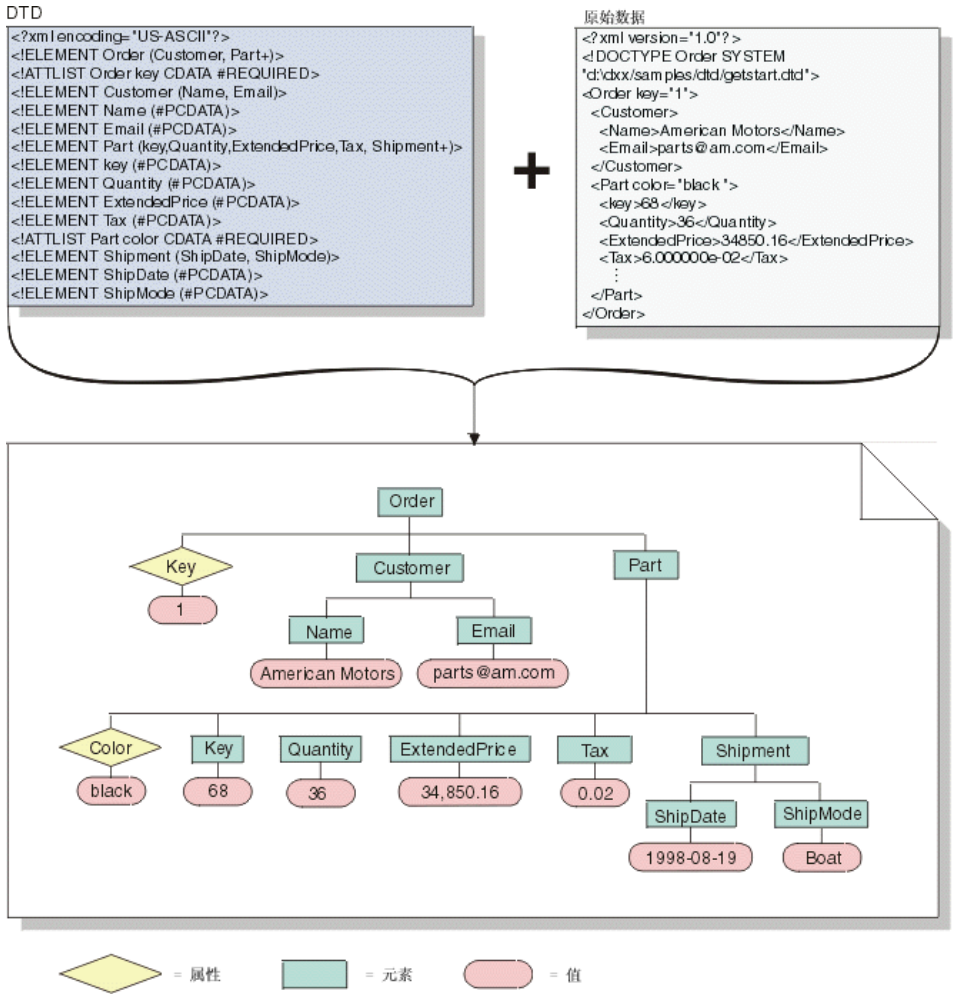


图 5. DTD 和 XML 文档的层次结构

映射 XML 文档和数据库关系

在设计了结构和创建了 DTD 之后，您需要显示文档的结构与将用来填充元素和属性的 DB2 表是如何相关的。可以将层次结构映射至关系表中的特定列，正如第29页的图6中所示。

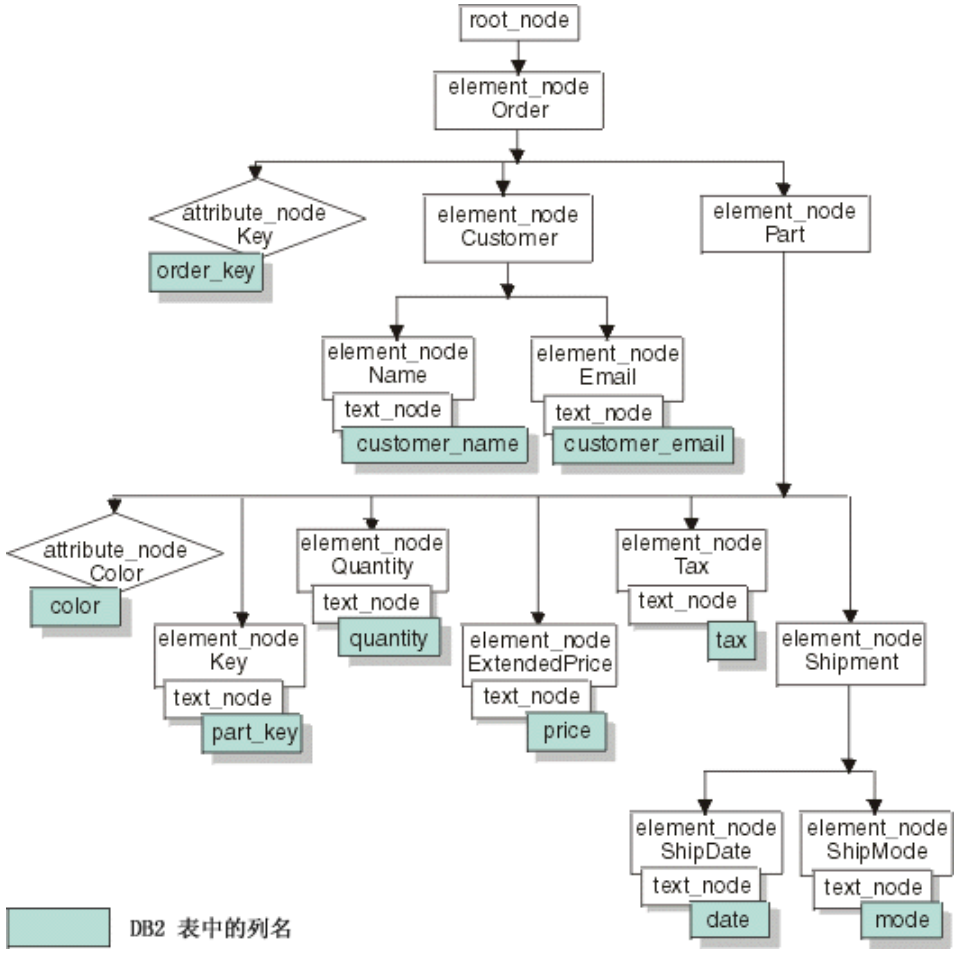


图 6. 映射至关系表列的 XML 文档

使用此关系说明来创建 DAD 文件，这些文件定义关系数据和 XML 文档结构之间的关系。

要创建 XML 集合 DAD 文件，需要了解 XML 文档是如何与数据库结构对应的，正如图6中所述，以便可以描述 XML 文档结构从哪些表和列中为元素和属性派生数据。 您将使用此信息来为 XML 集合创建 DAD 文件。

对于此教程，我们提供了一组脚本，以便您用来设置您的环境。这些脚本都在 *DXX_INSTALL*\samples\cmd 目录（其中 *DXX_INSTALL* 是安装有 XML Extender 的驱动器和目录，例如 c:\dxx\samples\cmd）中，下面列示了它们:

getstart_db.cmd

创建数据库并填充四个表。

getstart_prep.cmd

将数据库与 XML Extender 存储过程和 DB2 CLI 绑定。

getstart_stp.cmd

运行该存储过程来组合 XML 集合。

getstart_exportXML.cmd

从数据库导出 XML 文档以供应用程序使用。

getstart_clean.cmd

清除教程环境。

设置

创建数据库

在本节中，使用一个命令来建立数据库。此命令将创建样本数据库，与它进行连接，创建用来保存数据的表，然后插入数据。

重要事项: 如果完成了 XML 列课程而尚未清除您的环境，则您可能可以跳过此步骤。检查是否有 SALES_DB 数据库。

要创建该数据库:

1. 更改为 *DXX_INSTALL*\samples\cmd 目录，其中，*DXX_INSTALL* 是安装 XML Extender 的驱动器和目录。在这些课程中采用 c:\dxx 目录。如果您的驱动器和目录不同，则更改这些值。
2. 从 Windows NT “开始” 菜单打开 “DB2 命令窗口”，或从 Windows NT 命令提示处输入以下命令:

   ```
   DB2CMD
   ```

3. 从 “DB2 命令窗口”，运行以下命令:

   ```
   getstart_db.cmd
   ```

启用数据库

要在数据库中存储 XML 信息，需要为 XML Extender 启用该数据库。当为 XML 启用数据库时，XML Extender 会:

- 创建所有用户定义类型 (UDT) 和用户定义函数 (UDF)。
- 创建控制表，并用 XML Extender 所需要的元数据来填充该控制表。
- 创建 db2xml 模式并分配必需的特权。

重要事项: 如果完成了 XML 列课程而尚未清除您的环境，则您可能可以跳过此步骤。

要为 XML 启用数据库:

从 “DB2 命令窗口”，运行以下脚本以启用 SALES_DB 数据库:

```
getstart_prep.cmd
```

此脚本将数据库与 XML Extender 存储过程和 DB2 CLI 进行绑定。它还运行用来启用数据库的 **dxxadm** 命令选项:

```
dxxadm enable_db SALES_DB
```

创建 XML 集合: 准备 DAD 文件

因为数据已经存在于多个表之中，所以将创建 XML 集合，它将这些表与 XML 文档相关联。要创建 XML 集合，通过准备 DAD 文件来定义该集合。

在 第27页的『计划』 中确定关系数据库中应该包括哪些列（数据在这些数据库中），以及如何将表中的数据构造成 XML 文档 在本节中，在 DAD 文件中创建映射模式，该映射模式指定表与 XML 文档结构之间的关系。

在下列步骤中，将 DAD 中的元素称为*标记*，将 XML 文档结构的元素称为*元素*。类似于您将创建的 DAD 文件的一个DAD 文件样本在 c:\dxx\samples\dad\getstart_xcollection.dad 中。它与下列步骤中所生成的文件稍有差别。如果将它用于本课程，则要注意，这些文件路径可能与您所在环境的文件路径不同。

要创建 DAD 文件以组合 XML 文档:

1. 从 c:\dxx\samples\cmd 目录，打开文本编辑器并创建文件 getstart_xcollection.dad。
2. 使用以下文本来创建 DAD 头:

   ```
   <?xml version="1.0"?>
   <!DOCTYPE DAD SYSTEM "c:\dxx\dtd\dad.dtd">
   ```

   XML Extender 假定您在 c:\dxx 中安装了产品。如果上面的内容不正确，则在此处和下列步骤中将此值更改为在安装此产品期间您指定的驱动器和目录。
3. 插入 <DAD></DAD> 标记。其他所有标记都位于这些标记内。
4. 指定 <validation> </validation> 标记，以便指示 XML Extender 是否使用您插入到 DTD 库表中的 DTD 来验证 XML 文档结构。

   ```
   <validation>NO</validation>
   ```
5. 使用 <Xcollection> </Xcollection> 标记，将访问和存储方法定义为 XML 集合。 访问和存储方法定义: 将 XML 数据存储在 DB2 表的集合中。

   ```
   <Xcollection>
   </Xcollection>
   ```

6. 指定一个 SQL 语句，该语句指定用于 XML 集合的表和列。此方法称为 SQL 映射，它是将关系表映射至 XML 文档结构的两种方法中的一种。（要更多地了解映射模式，参见第55页的『映射模式的类型』。）输入下列语句:

```
<SQL_stmt>
   SELECT o.order_key, customer_name, customer_email, p.part_key, color, quantity,
   price, tax, ship_id, date, mode from order_tab o, part_tab p,
   table (select substr(char(timestamp(generate_unique())),16)
     as ship_id, date, mode, part_key from ship_tab) s
         WHERE o.order_key = 1 and
               p.price > 20000 and
               p.order_key = o.order_key and
               s.part_key = p.part_key
         ORDER BY order_key, part_key, ship_id
</SQL_stmt>
```

   此 SQL 语句在使用 SQL 映射时使用下列准则。参考第29页的图6以了解文档结构。

   - 列是以自上而下的次序（即按 XML 文档结构的层次结构）来指定的。例如，订单和客户元素的列最先，部件元素的列其次，而交付元素的列最后。
   - 一个实体的各列被分组在一起，每组都具有一个对象标识列: ORDER_KEY、PART_KEY 和 SHIP_ID。
   - 对象标识列是每组中的第一列。例如，O.ORDER_KEY 在与键属性相关的各列之前，而 p.PART_KEY 在“部件”元素的各列之前。
   - SHIP_TAB 表没有一个键条件列，因此，生成唯一 DB2 内部函数用来生成 SHIP_ID 列。
   - 于是，各对象标识列以自上而下的次序列示在 ORDER BY 语句中。ORDER BY 中的各列不应被任何模式和表名限定，而应与 SELECT 子句中的各列名相匹配。

   参见第56页的『映射模式需求』以了解编写 SQL 语句时的需求。

7. 添加以下要在组合 XML 文档中使用的 prolog 信息。

```
<prolog>?xml version="1.0"?</prolog>
```

   此精确文本是所有 DAD 文件都必需的。

8. 添加 <doctype></doctype> 标记以在您正在组合的 XML 文档中使用。<doctype> 标记包含至存储在客户机上的 DTD 的路径。

```
<doctype>!DOCTYPE Order SYSTEM "c:\dxx\samples\dtd\getstart.dtd"</doctype>
```

9. 使用 <root_node></root_node> 标记来定义 XML 文档的根元素。在 root_node 内部，指定构成 XML 文档的元素和属性。
10. 使用下列三种类型的节点，将 XML 文档结构映射至 DB2 关系表结构:

element_node

指定 XML 文档中的元素。Element_node 可具有子代 element_node。

attribute_node

指定 XML 文档中的元素的属性。

text_node

指定元素的文本内容以及底层 element_nodes 的关系表中的列数据。

有关这些节点的详情，参见第51页的『DAD 文件』。 第29页的图6显示了 XML 文档的层次结构以及 DB2 表列，并指示使用了哪种节点。 变黑的框指示将从中抽取数据，以便组合 XML 文档的 DB2 表列名。

下列步骤让您添加每种类型的节点，一次只添加一种类型。

a. 为 XML 文档中的每个元素定义 <element_node> 标记。

```
<root_node>
<element_node name="Order">
  <element_node name="Customer">
    <element_node name="Name">
    </element_node>
     <element_node name="Email">
    </element_node>
  </element_node>
  <element_node name="Part">
    <element_node name="key">
    </element_node>
    <element_node name="Quantity">
    </element_node>
     <element_node name="ExtendedPrice">
    </element_node>
     <element_node name="Tax">
    </element_node>
     <element_node name="Shipment" multi_occurrence="YES">
              <element_node name="ShipDate">
       </element_node>
       <element_node name="ShipMode">
       </element_node>
    </element_node> <!-- end Shipment -->
  </element_node> <!-- end Part -->
</element_node> <!-- end Order -->
</root_node>
```

注意，<Shipment> 子元素的 multiple_occurrence 属性为 ″YES″。此属性用于文档中重复的、没有属性的元素。 <Part> 元素并不使用 multiple_occurrence 属性，原因是它具有颜色属性，这使它为唯一。

b. 为 XML 文档中的每种属性定义 <attribute_node> 标记。这些属性嵌套在它们的 element_node 中。用粗体来突出显示所添加的 attribute_nodes:

```
<root_node>
<element_node name="Order">
  <attribute_node name="key">
  </attribute_node>
  <element_node name="Customer">
    <element_node name="Name">
    </element_node>
     <element_node names"Email">
    </element_node>
  </element_node>
  <element_node name="Part">
    <attribute_node name="color">
    </attribute_node>
    <element_node name="key">
    </element_node>
    <element_node name="Quantity">
    </element_node>

...

  </element_node> <!-- end Part -->
</element_node> <!-- end Order -->
</root_node>
```

c. 为每个底层 element_node 定义 <text_node> 标记，该标记指示 XML 元素中包含组合文档时要从 DB2 中抽取的字符数据。

```
<root_node>
<element_node name="Order">
  <attribute_node name="key">
  </attribute_node>
  <element_node name="Customer">
    <element_node name="Name">
        <text_node>
        </text_node>
    </element_node>
    <element_node name="Email">
        <text_node>
        </text_node>
    </element_node>
  </element_node>
  <element_node name="Part">
    <attribute_node name="color">
    </attribute_node>
    <element_node name="key">
       <text_node>
        </text_node>
    </element_node>
    <element_node name="Quantity">
       <text_node>
        </text_node>
    </element_node>
    <element_node name="ExtendedPrice">
       <text_node>
        </text_node>
```

```
    </element_node>
    <element_node name="Tax">
        <text_node>
        </text_node>
    </element_node>
    <element_node name="Shipment" multi-occurrence="YES">
      <element_node name="ShipDate">
        <text_node>
        </text_node>
      </element_node>
      <element_node name="ShipMode">
        <text_node>
        </text_node>
      </element_node>
    </element_node> <!-- end Shipment -->
  </element_node> <!-- end Part -->
</element_node> <!-- end Order -->
</root_node>
```

d. 为每个底层 element_node 定义 <column> 标记。这些标记指定组合 XML 文档时要从哪些列抽取数据，并且这些标记通常在 <attribute_node> 或 <text_node> 标记内。记住，此处定义的列必须在 <SQL_stmt> SELECT 子句中。

```
<root_node>
<element_node name="Order">
  <attribute_node name="key">
    <column name="order_key"/>
  </attribute_node>
  <element_node name="Customer">
    <element_node name="Name">
                <text_node>
        <column name="customer_name"/>
                </text_node>
    </element_node>
     <element_node name="Email">
                <text_node>
        <column name="customer_email"/>
                </text_node>
    </element_node>
  </element_node>
  <element_node name="Part">
    <attribute_node name="color">
       <column name="color"/>
    </attribute_node>
    <element_node name="key">
                <text_node>
        <column name="part_key"/>
                </text_node>
    <element_node name="Quantity">
                <text_node>
        <column name="quantity"/>
                </text_node>
    </element_node>
    <element_node name="ExtendedPrice">
```

```
                <text_node>
            <column name="price"/>
                </text_node>
        </element_node>
        <element_node name="Tax">
                <text_node>
            <column name="tax"/>
                </text_node>
        </element_node>
        <element_node name="Shipment" multi-occurrence="YES">
          <element_node name="ShipDate">
                <text_node>
              <column name="date"/>
                </text_node>
          </element_node>
          <element_node name="ShipMode">
                <text_node>
              <column name="mode"/>
                </text_node>
          </element_node>
        </element_node> <!-- end Shipment -->
      </element_node> <!-- end Part -->
    </element_node> <!-- end Order -->
    </root_node>
```

11. 必须确保在最后一个 </element_node> 标记之后，有结束标记 </root_node>。
12. 必须确保在 </root_node> 标记之后，有结束标记 </Xcollection>。
13. 必须确保在 </Xcollection> 标记之后，有结束标记 </DAD>。
14. 将文件另存为 getstart_xcollection.dad

可将刚刚创建的文件与样本文件 c:\dxx\samples\dad\getstart_xcollection.dad 相比较。此文件是组合 XML 文档所必需的 DAD 文件的工作副本。样本文件包含路径语句，可能需要更改这些语句以与您的环境相匹配，以便成功运行。

在应用程序中，如果您经常会使用 XML 集合来组合文档，则可通过启用该集合来定义一个集合名。启用该集合即在 XML_USAGE 表中注册了该集合，且若指定了集合名（而不是 DAD 文件名），则在运行存储过程时就可改进性能。在这些课程中，将不启用该集合。要了解有关启用集合的详情，参见第98页的『启用 XML 集合』。

组合 XML 文档

在此步骤中，使用 dxxGenXML() 存储过程来组合由 DAD 文件所指定的 XML 文档。此存储过程将文档作为 XMLVARCHAR UDT 返回。

要组合 XML 文档:

1. 从“DB2 命令窗口”，输入以下命令来运行存储过程:

```
getstart_stp.cmd
```

已经组合了 XML 文档，并且存储在 RESULT_TAB 表中。

可在下列文件中看到可用于此步骤的存储过程的样本:

- c:\dxx\samples\c\tests2x.sqc 显示如何使用嵌入式 SQL 调用存储过程，并生成 texts2x 可执行文件，它是由 getstart_stp.cmd 使用的。
- c:\dxx\samples\cli\sql2xml.c 显示如何使用 CLI 调用存储过程。

2. 使用 XML Extender 检索函数 Content() 将 XML 文档从表导出至文件:

```
DB2 CONNECT TO SALES_DB

DB2 SELECT db2xml.Content(db2xml.xmlvarchar(doc),
   'c:\dxx\samples\cmd\getstart.xml') FROM RESULT_TAB
```

此外，可运行以下命令文件来导出该文件:

```
getstart_exportXML.cmd
```

此课程教您如何使用 DB2 存储过程的结果集功能部件来获取一个或多个组合 XML文档，以使您可取装每一行来获取每个文档。由于您获取了文档的每一行，您就可将其导出至文件，这是演示此功能部件的最简单方法。有关取装数据的更多有效方法，参见 c:\dxx\samples\cli 中的 CLI 示例。

清除教程环境

如果想要清除教程环境，可运行 getstart_clean.cmd 文件。此文件:

- 禁用 XML 列 ORDER
- 删除在该教程中创建的表
- 从 DTD 参考表中删除 DTD

此命令文件不禁用或删除 SALES_DB 数据库；该数据库仍可与 XML Extender 配合使用。如果您未完成本章中的两节课程，则可能会接收到错误消息。您可忽略这些错误。

要清除教程环境:

1. 从 “DB2 命令窗口”，运行以下命令:

```
getstart_clean.cmd
```

2. 如果想要禁用该数据库，您可从 “DB2 命令窗口” 运行以下 XML Extender 命令:

```
dxxadm disable_db SALES_DB
```

此命令删除了管理控制表 DTD_REF 和 XML_USAGE，还除去了 XML Extender 提供的用户定义类型和函数。

3. 如果想要删除数据库，可从“DB2 命令窗口”运行以下命令:

```
db2 drop database SALES_DB
```

此命令删除 SALES_DB。

第2部分 管理

此部分描述如何执行 XML Extender 的管理任务。

© Copyright IBM Corp. 1999, 2000

第3章 准备使用 XML Extender: 管理

本章描述设置和计划 XML Extender 的需求。

设置需求

下列部分描述 XML Extender 的设置需求。

软件需求

XML Extender 在 AIX、Windows NT 和 Sun Solaris 上都是可用的。

必需的软件: XML Extender 需要 DB2 通用数据库版本 7.1 或更高版本。

可选软件:

- 对于结构化文本搜索，可选软件为 DB2 通用数据库 Text Extender 版本 7.1 或更高版本
- 对于 XML Extender 管理工具:
  - DB2 UDB JDBC（随 DB2 UDB 版本 7.1 或更高版本一起提供）
  - JDK 1.1.7 或 JRE 1.1.7（随 DB2 UDB 控制中心一起提供）
  - 带有 Swing 1.1 的 JFC 1.1（随 DB2 UDB 控制中心一起提供）

安装需求

参见您的操作系统的自述文件文件，以了解下列任务:

- 将 XML Extender 绑定至 DB2 UDB 数据库。

  由于安全性原因，必须将 XML Extender 绑定至每个数据库。有关如何完成绑定的详情，参见第186页的『开始之前』，有关示例，则参见

  DXX_INSTALL\samples\cmd\getstart_prep.cmd
- 查看有关 UNIX 的设置指导。
- 为进行 XML 访问创建一个数据库。

权限需求

您需要 DB2ADM 权限才能执行管理任务。

© Copyright IBM Corp. 1999, 2000

管理工具

XML Extender 提供了三种管理方法: XML Extender 管理向导、 XML Extender 管理命令和 XML Extender *存储过程*。

- 管理向导提示您完成管理任务，它是我们建议的管理方法。此工具的使用在第61页的『第4章 管理 XML 数据』中的管理任务中作了描述。
- 管理命令 **dxxadm** 为各种管理任务提供选项。此命令的使用在第61页的『第4章 管理 XML 数据』和 第139页的『第7章 XML Extender 管理命令: **dxxadm**』的管理任务中作了描述。
- 管理存储过程也提供了各种管理任务的选项。 这些存储过程在第187页的『管理存储过程』中作了描述。

管理计划

当计划一个使用 XML 文档的应用程序时，首先需要作出下列设计决定:

- 如果将要使节使用数据库中的数据来组合 XML 文档
- 如果将要存储已存在的 XML 文档，且如果您想要将它们作为完整的 XML 文档存储在列中或将它们分解成标准的 DB2 数据

在您作出这些决定之后，可计划余下的管理任务:

- 是否验证 XML 文档
- 是否为进行快速搜索及检索而创建 XML 列数据的索引
- 如何将 XML 文档结构映射至 DB2 关系表

您如何使用 XML Extender 要视应用程序的要求而定。如 第3页的『第1章 XML Extender 的介绍』所示，可从现存的 DB2 数据组合 XML 文档并将 XML 文档存储在 DB2 中（作为完整的文档或作为 DB2 数据）。每个这样的存储和访问方法都具有不同的计划需求。下列部分讨论这些计划注意事项中的每一个。

选择访问和存储方法

XML Extender 提供了两种将 DB2 用作 XML 库的访问和存储方法: XML 列和 XML 集合。首先要决定哪种方法最能满足您的应用程序访问和处理 XML 数据的需要。

XML 列

将整个 XML 文档作为 DB2 列数据来存储和检索。XML 数据是由 XML 列表示的。

XML 集合

将 XML 文档分解为关系表的集合或从关系表的集合组合 XML 文档。

应用程序的性质确定要使用的访问和存储方法的类型以及构建 XML 数据的方式。下列方案描述每个访问和存储方法都非常适用的情况。

何时使用 XML 列

在下列情况下使用 XML 列:

- XML 文档已经存在或来自某些外部数据源，且您想以本机 XML 格式存储这些文档。您想要在 DB2 中存储它们，以保持其完整性及便于归档和审查。
- XML 文档一般是可读取的，但不能更新。
- 您想使用文件名数据类型，将 XML 文档存储在 DB2 之外的本地或远程文件系统中，并将 DB2 用于管理和搜索操作。
- 您需要根据 XML 元素或属性的值来进行范围搜索，且您知道哪些元素或属性将被频繁地用作搜索自变量。
- 文档具有带有大文本块的元素，且您想要在保持整个文档完整性的同时，将 DB2 Text Extender 用于结构化文本搜索。

何时使用 XML 集合

在下列情况下使用 XML 集合:

- 您在现存的关系表中有数据，且您想要根据某个 DTD 来组合 XML 文档。
- 您有一些需要使用数据集合来存储的 XML 文档，这些数据正好映射至关系表。
- 您想要使用不同的映射模式来创建各种关系数据视图。
- 您具有来自其他数据源的 XML 文档。您关心的是数据而不是标记，且想要将纯数据存储在数据库中。您想要灵活地决定是将数据存储在某些现存的表中还是存储在新表中。
- XML 文档的小子集需要经常更新，且更新性能是非常重要的。
- 您需要存储整个入局 XML 文档的数据，但经常只想检索它们的子集。
- 您的 XML 文档超过了 2G 字节，您必须分解它们。

通过这两种访问和存储方法，使用文档访问定义 (DAD) 文件将 XML 数据与 DB2 表相关联。第44页的图7显示了 DAD 是如何指定访问和存储方法的。

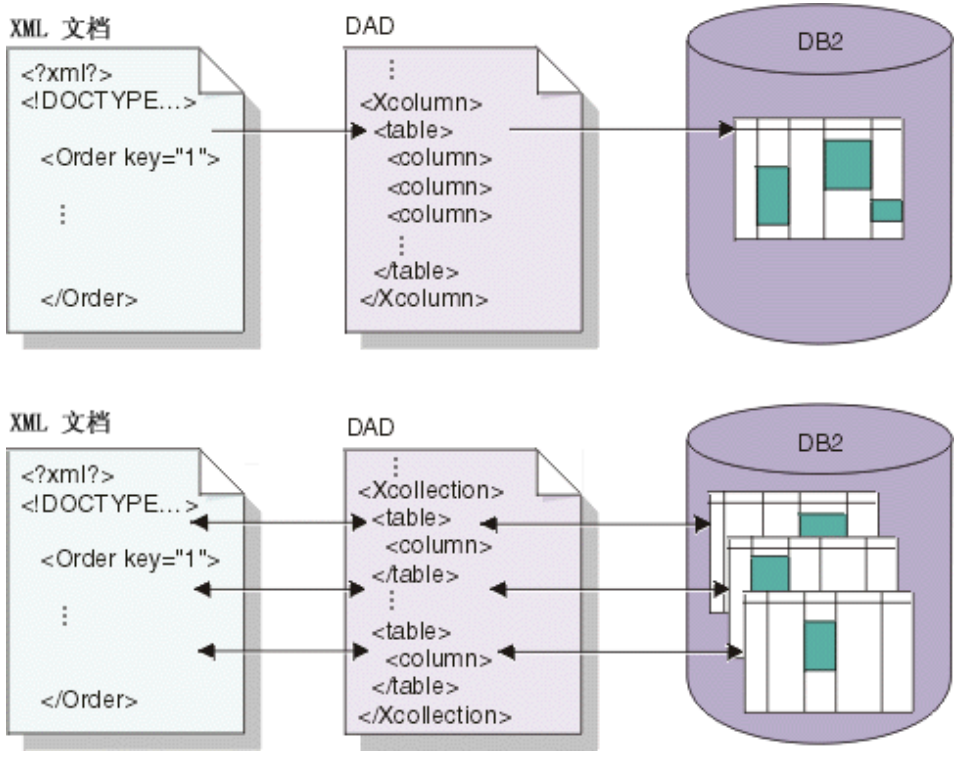


图 7. DAD 文件将 XML 文档结构映射至 DB2，并指定访问和存储方法。

DAD 文件是管理 XML Extender 的重要部分。它定义诸如 DTD 之类的键文件的位置，并指定 XML 文档与 DB2 数据相关的方式。最重要的是，它定义您在应用程序中所使用的访问和存储方法。

计划 XML 列

下列部分描述用于 XML 列的计划任务。

验证

在选择了访问和存储方法之后，您可以确定是否*验证*数据。使用 DTD 验证 XML 数据，以确保 XML 文档是有效的，并对您的 XML 数据执行结构化搜索。DTD 存储在 DTD 库中，也可以将它存储在 DB2 服务器具有访问权的文件系统中。

可使用不同的 DTD，在同一 XML 列中验证文档。换句话说，您拥有的文档可以有类似的结构，类似的元素和属性，但调用的 DTD 不同。 要引用多个 DTD，可使用下列准则:

- DOCTYPE 定义中的 XML 文档的系统标识必须使用全路径名来指定 DTD 文件。

- 您必须指定 YES，才能进行 DAD 文件中的验证。
- 至少其中一个 DTD 必须存储在 DTD_REF 表中。所有 DTD 都可存储在此表中。
- 这些 DTD 应具有一个公共结构，仅在子元素中才有差异。
- DAD 文件应指定的元素或属性，对该列中的文档所引用的所有 DTD 是公共的。

重要事项：决定是否在将 XML 数据插入 DB2 之前进行验证。XML Extender 不支持已经插入 DB2 中的数据的验证。

注意事项:

- 建议您使用 DTD 验证 XML 数据，除非您正在为进行归档而存储 XML 文档。要进行验证，需要在 XML Extender 库中具有 DTD。参见第65页的『将 DTD 存储在 DTD 库中』以了解如何将 DTD 插入库。
- 不需要 DTD 来存储或归档 XML 文档。
- 验证 XML 数据可能有小小的性能影响。
- 可使用多个 DTD，但仅对公共元素和属性创建索引。
- 若您不选择验证文档，则将不处理 XML 文档指定的 DTD。即使在处理不能验证的文档片段时，为了解析实体和属性缺省值，处理 DTD 是一个重要步骤。

XML 用户定义类型

将 XML 文档存储在作为 UDT 的 XML 列中。参见表4以获取可用的 UDT。

表 4. XML Extender UDT

用户定义类型列	源数据类型	用法说明
XMLVARCHAR	VARCHAR(*varchar_len*)	将整个 XML 文档作为 VARCHAR 存储在 DB2 中。
XMLCLOB	CLOB(*clob_len*)	将整个 XML 文档作为*CLOB* 存储在 DB2 中。
XMLFILE	VARCHAR(1024)	将 XML 文档的文件名存储在 DB2 中，并将 XML 文档存储在 DB2 服务器本地的文件中。

辅助表

计划辅助表时，必须考虑如何组织这些表，要创建多少个表，及是否为辅助表创建缺省视图。这些决定部分地取决于下列几个问题：元素和属性是否可出现多次，以及查询性能的需求。

多次出现: 当文档有多次出现的位置路径时，XML Extender 将在每个辅助表中添加类型为 INTEGER 的列 DXX_SEQNO 以记录多次出现的元素的次序。借助 DXX_SEQNO，可通过在 SQL 查询中指定 ORDER BY DXX_SEQNO 来使用与原始 XML 文档中所具有的同一次序来列示这些元素。

缺省视图和查询性能: 当启用 XML 列时，可使用唯一标识（称为 ROOT 标识）来指定将应用程序表与辅助表相连的缺省只读视图。借助缺省视图，可通过查询辅助表来搜索 XML 文档。例如，如果您具有应用程序表 SALES_TAB，及辅助表 ORDER_TAB、PART_TAB 和 SHIP_TAB:

```
SELECT sales_person FROM sales_order_view
   WHERE price > 2500.00
```

SQL 语句在 SALES_TAB 中返回在 ORDER 列中存储有订单且其中 PRICE 大于 2500.00 的销售人员的姓名。

查询缺省视图的优点是提供了应用程序表和辅助表的虚拟单个视图。然而，创建的辅助表越多，查询的成本就越高。因此，仅当辅助表列的总数很少时，才建议创建缺省视图。应用程序可创建它们自己的视图，并将重要的辅助表列连接起来。

XML 列数据的索引

一个重要的计划决定是是否对 XML 列文档创建索引。这个决定取决于您需要访问数据的频率以及在结构化搜索期间，性能的重要程度。

当使用 XML 列（它包含整个 XML 文档）时，可创建要包含 XML 元素列或属性值列的辅助表，然后创建这些列的索引。必须确定您需要对其创建索引的元素或属性。

创建 XML 列的索引，允许使用数据库引擎中的本机 DB2 索引支持，为频繁查询的、一般数据类型（如整数、十进制数或日期）的数据创建索引。XML Extender 从 XML 文档抽取 XML 元素或属性的值，并将它们存储在*辅助表*中，而且允许您创建这些辅助表的索引。

您可使用位置路径来指定辅助表的每一列，该位置路径标识 XML 元素或属性以及 SQL 数据类型。第47页的图8显示了辅助表的 XML 列。

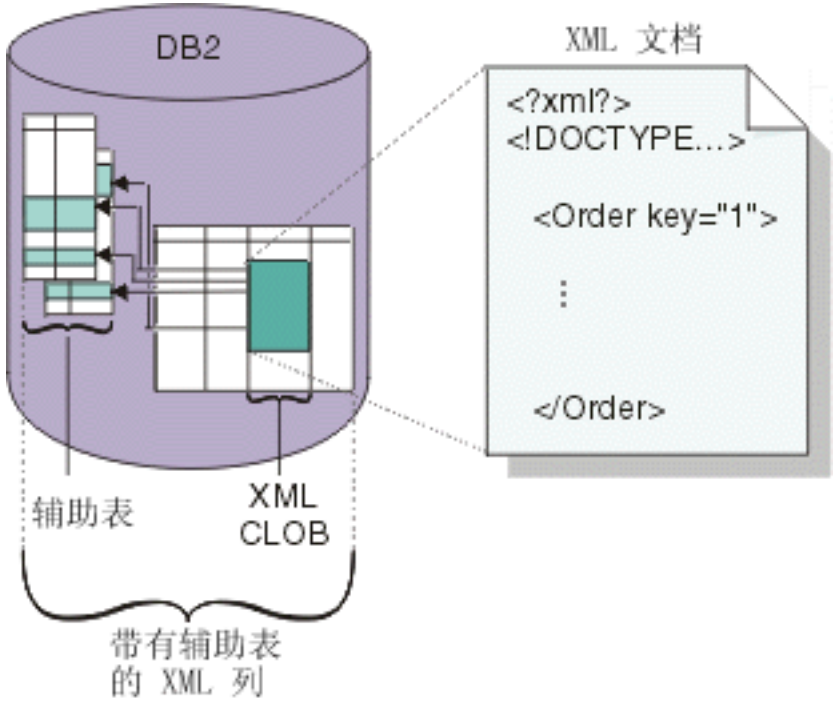


图 8. 辅助表的 XML 列。

当将 XML 文档存储在 XML 列中时，XML Extender 会自动填充辅助表。

为进行快速搜索，可使用 DB2 *B-tree 索引*技术创建这些列的索引。用于创建索引的方法随不同的操作系统而有所变化，XML Extender 支持这些方法。

注意事项:

- 对于在 XML 文档中*多次出现*的元素或属性，您必须为由于 XML 文档的复杂结构而多次出现的每个 XML 元素或属性创建独立的辅助表。

  例如，您可能想要为 `/Order/Part/ExtendedPrice` 创建索引，并将 `/Order/Part/ExtendedPrice` 指定为数据类型 REAL。在这种情况下， XML Extender 将 `/Order/Part/ExtendedPrice` 的值存储在辅助表的 PRICE 列中。
- 可为一个 XML 列创建多个索引。使用前一个示例，可在两个辅助表中创建两列，一列用于 `ExtendedPrice`，一列用于 `ShipDate`。
- 可将辅助表与使用 ROOT 标识的应用程序表、应用程序表中的主键的列名，以及将所有辅助表与应用程序表相关联的唯一标识符相关联。 您可决定是否想要应用程序表的主键为 ROOT 标识，即使它不可为组合键。建议使用此方法。

如果应用程序表中不存在单个主键，或由于某些原因您不想使用它，则 XML Extender 会改变该应用程序表以添加列 DXXROOT_ID，它存储了在插入时创建的唯一标识。 所有辅助表都有一个带有唯一标识的 DXXROOT_ID 列。如果将主键用作 ROOT 标识，则所有辅助表都具有与应用程序表中的主键列的名称和类型相同的列，且将这些主键的值存储起来。

- 如果对 DB2 Text Extender 启用了 XML 列，则还可使用 Text Extender 的结构化文本功能部件。Text Extender 中有 "*部分搜索*" 支持，它允许在由位置路径指定的特定文档上下文内匹配搜索字，因而扩展了常规全文本搜索的能力。*结构化文本索引*可与 XML Extender 的创建一般 SQL 数据类型索引配合使用。

位置路径

*位置路径*是标识 XML 元素或属性的 *XML 标记*序列。XML Extender 在下列情况下使用位置路径:

- 当抽取 UDF 以标识要被抽取的元素和属性时
- 在 XML 列的 DAD 中定义索引模式时，标识 XML 元素或属性与 DB2 列之间的映射文件
- 通过 Text Extender 进行结构化文本搜索

图9显示位置路径及它与 XML 文档结构之间的关系的示例。

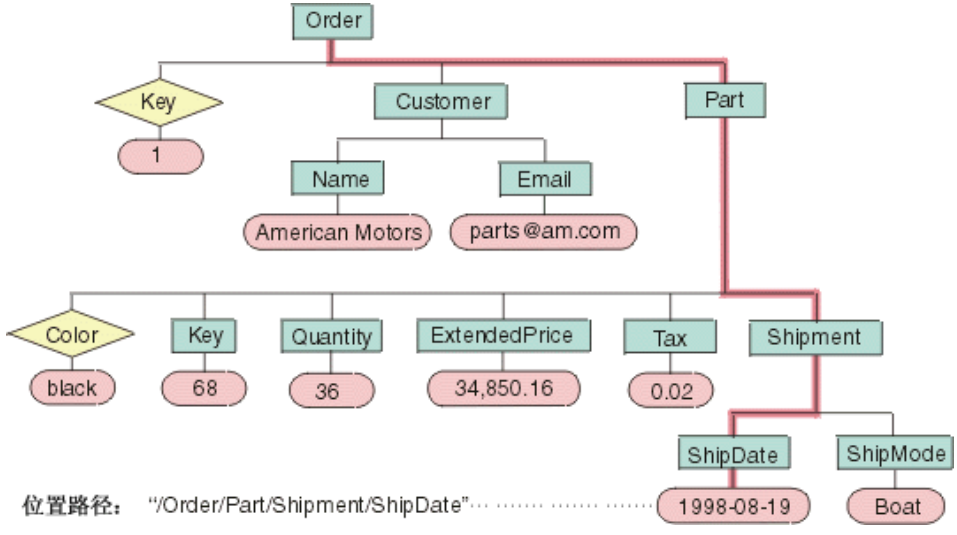


图 9. 将文档作为结构化 XML 文档存储在 DB2 表列中

位置路径语法: 以下列表描述了受 XML Extender 支持的位置路径语法。单个斜杠 (/) 路径指示上下文是整个文档。

1. */* 表示 XML *根元素*。
2. */tag1*
   表示根下面的元素 *tag1*。
3. */tag1/tag2/..../tagn*
   表示名为 *tagn* 的元素，它是从根、 *tag1*、*tag2* 直到 *tagn*-1 的降序链的子辈。
4. *//tagn*
   表示任何名为 *tagn* 的元素，其中双斜杠 (//) 表示零个或多个任意标记。
5. */tag1//tagn*
   表示任何名为 *tagn* 的元素，它是根下面名为 *tag1* 的元素的子辈，其中双斜杠 (//) 表示零个或多个任意标记。
6. */tag1/tag2/@attr1*
   表示名为 *tag2* 的元素的属性 *attr1*，该元素是根下面的元素 *tag1* 的子辈。
7. */tag1/tag2[@attr1="5"]*
   表示名为 *tag2* 的元素，它的属性 *attr1* 的值为 5。*tag2* 是根下面名为 *tag1* 的元素的子辈。
8. */tag1/tag2[@attr1="5"]/.../tagn*
   表示名为 *tagn* 的元素，它是从根、*tag1*、 *tag2* 至 *tagn*-1 的降序链的子辈，其中 *tag2* 的属性 *attr1* 的值为 5。

通配符: 可用星号来替换位置路径中的元素以匹配任何字符串。

简单位置路径: *简单位置路径*是用于指定辅助表的元素和属性的位置路径语法，它在 XML 列 DAD 文件中定义。简单位置路径表示为由单个斜杠 (/) 连接的元素类型名的序列。属性值是在其元素类型后用方括号括起来的。表5总结了简单位置路径的语法。

表 5. 简单位置路径语法

主题	位置路径	描述
XML 元素	*/tag1/tag2/..../tagn-1/tagn*	由名为 *tagn* 的元素及其父辈标识的元素内容
XML 属性	*/tag_1/tag_2/..../tag_n-1/tag_n/@attr1*	由 *tagn* 及其父辈标识的元素的属性（名为 *attr1*）

XML Extender 限制: XML Extender 对当定义 DAD 中的元素或属性时使用位置路径具有限制。 因为 XML Extender 使用元素或属性和 DB2 列之间的一对一映射， 所以它限制 DAD 文件和函数中所允许的语法规则。 第50页的表6描述位

置路径的限制。在“受支持的位置路径”列中指定的编号引用第48页的『位置路径语法』中的语法表示法。

表 6. 使用位置路径的 XML Extender 限制

位置路径的使用	受支持的位置路径
辅助表的 XML 列 DAD 映射中的元素	3, 6（第49页的表5中描述的简单位置路径)
抽取 UDF	1-8[1]
更新 UDF	1-8[1]
Text Extender 的搜索 UDF	1-8

[1] 抽取和更新 UDF 支持具有谓词的位置路径，但这些谓词只能带属性而不能带元素。

DAD 文件

对于 XML 列，DAD 主要指定如何对存储在 XML 列中的文档创建索引。DAD 是 XML 格式化文档，驻留在客户机上。如果您选择用 DTD 验证 XML 文档，则 DAD 文件可与该 DTD 相关联。 DAD 文件的数据类型为 CLOB。此文件最大为 100 KB。

XML 列的 DAD 文件包含 XML 头，它为 DAD 文件和 DTD 指定客户机上的目录路径，并提供要存储在辅助表中，以用于创建索引的任何 XML 数据的映象。

要指定 XML 列访问和存储方法，在 DAD 文件中使用以下标记。

<Xcolumn>

指定 XML 数据将作为 DB2 列中的完整 XML 文档进行存储和检索，这些 DB2 列是对 XML 数据启用的。

启用了 XML 的列属于 XML Extender 的 UDT。应用程序可在任何*用户表*中包括该列。主要通过 SQL 语句和 XML Extender 的 UDF 来访问 XML 列数据。

可使用 XML Extender 管理向导或编辑器来创建和更新 DAD。

计划 XML 集合

计划 XML 集合时，对于从 DB2 数据组合文档和／或将 XML 文档分解为 DB2 数据，具有不同的注意事项。下列章节描述了 XML 集合的计划问题以及组合和分解注意事项。

验证

在选择访问和存储方法之后，可确定是否验证数据。 使用 DTD 来验证 XML 数据。使用 DTD 确保 XML 文档有效且可让您对 XML 数据执行结构化搜索。 DTD 是存储在 DTD 资源库中的。

建议: 使用 DTD 来验证 XML 数据。 要进行验证，需要在 XML Extender 库中具有 DTD。 要了解将 DTD 插入库的方式，参见第65页的『将 DTD 存储在 DTD 库中』。 DTD 需求随您是正在组合还是分解 XML 文档而有所不同。

- 对于组合，您只可对一个 DTD 验证生成的 XML 文档。要使用的 DTD 是在 DAD 文件中指定的。
- 对于分解，可使用不同 DTD 来验证要组合的文档。换句话说，可使用同一 DAD 文件来分解文档，但调用不同的 DTD。要引用多个 DTD，必须使用下列指导:
  - 至少其中一个 DTD 必须存储在 DTD_REF 表中。所有 DTD 都可存储在此表中。
  - 这些 DTD 应具有一个公共结构，仅在子元素中才有差异。
  - 您必须指定 DAD 文件中的验证。
  - XML 文档的系统标识必须使用全路径名来指定 DTD 文件。
  - DAD 文件包含了如何分解文档的规范，因此可指定只分解公共元素和属性。不能分解对 DTD 来说是唯一的元素和属性。

重要事项: 决定是否要在将 XML 数据插入 DB2 之前验证 XML 数据。XML Extender 不支持已经插入 DB2 中的数据的验证。

注意事项:

- 在将 XML 用作交换格式时应使用 DTD。
- 验证 XML 数据可能有小小的性能影响。
- 当将多个 DTD 用于分解时，仅可分解公共元素和属性。
- 当使用一个 DTD 时可分解所有的元素和属性。
- 只可将一个 DTD 用于组合。

DAD 文件

对于 XML 集合，DAD 文件将 XML 文档的结构映射至一些 DB2 表，您要从这些 DB2 表中组合文档或将文档分解到这些表中。

例如，如果 XML 文档中有一个称为 <Tax> 的元素，您可能需要将 <Tax> 映射至一个称为 TAX 的列。定义 XML 数据与 DAD 中的关系数据之间的关系。

DAD 文件是当启用集合，或当您在 XML 集合*存储过程*中使用该 DAD 文件时指定的。 DAD 是 XML 格式化文档，驻留在客户机上。如果您选择用 DTD 验证 XML 文档，则 DAD 文件可与该 DTD 相关联。 当用作 XML Extender 存储过程的输入参数时， DAD 文件的数据类型为 CLOB。此文件最大为 100 KB。

要指定 XML 集合访问方法和存储方法，在 DAD 文件中使用以下标记:

<Xcollection>

指定 XML 数据是从 XML 文档分解为关系表的集合，还是从关系表的集合组合 XML 文档。

XML 集合是包含 XML 数据的一组关系表的虚拟名称。应用程序可启用任何用户表的 XML 集合。这些用户表可以是传统商业数据的现存表或是 XML Extender 最近创建的表。主要通过 XML Extender 提供的存储过程来访问 XML 集合数据。

DAD 文件使用下列几种节点来定义 XML 文档树结构:

root_node

指定文档的根元素。

element_node

标识一个元素，可以是根元素或子辈元素。

text_node

表示元素的 CDATA 文本。

attribute_node

表示元素的属性。

第53页的图10显示了在 DAD 文件中使用的映射分段。节点将 XML 文档内容映射至关系表中的表列。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:\dtd\dad.dtd">
<DAD>
  ...
<Xcollection>
    <SQL_stmt>
       ...
    </SQL_stmt>
    <prolog>?xml version="1.0"?</prolog>
<doctype>!DOCTYPE DAD SYSTEM "c:\dxx\sample\dtd\getstart.dtd"</doctype>
    <root_node>
   <element_node name="Order">          --> 标识元素 <Order>
     <attribute_node name="key">        --> 标识属性 "key"
        <column name="order_key"/>      --> 定义元素和属性被映射至的
                                               列 "order_key" 的名称

     </attribute_node>
     <element_node name="Customer">     --> 将 <Order> 的子辈元素标识为
                                               <Customer>
        <text_node>                     --> 指定元素的 CDATA 文本
                                               <Customer>
          <column name="customer">      --> 定义子元素被映射至的列 "customer" 的名称
        </text_node>
</element_node>
        ...
</element_node>

      ...
    <root_node>
  </Xcollection>
</DAD>
```

图 10. 节点定义

在本示例中，SQL 语句中的前两列将元素和属性映射至这些节点。

XML Extender 还使用 <stylesheet> 元素支持样式表的处理指令。处理指令必须在 DAD 文件的根节点中，并必须为 XML 文档定义 doctype 和 prolog。例如:

```
<Xcollection>
...
<prolog>...</prolog>
<doctype>...</doctype>
<stylesheet>?xml-stylesheet type="text/css" href="order.css"?</stylesheet>
<root_node>...</root_node>
...

  </Xcollection>
```

可使用 XML Extender 管理向导或编辑器来创建和更新 DAD 文件。XML Extender 管理向导当前不支持 <stylesheet> 元素。

XML 集合的映射模式

如果正在使用 XML 集合，必须选择一个*映射模式*，它定义 XML 数据在关系数据库中是如何表示的。因为 XML 集合必须与具有关系结构的 XML 文档中使用的层次结构匹配，所以应了解这两种结构是如何进行比较的。图11 显示了层次结构与关系表列的映射关系。

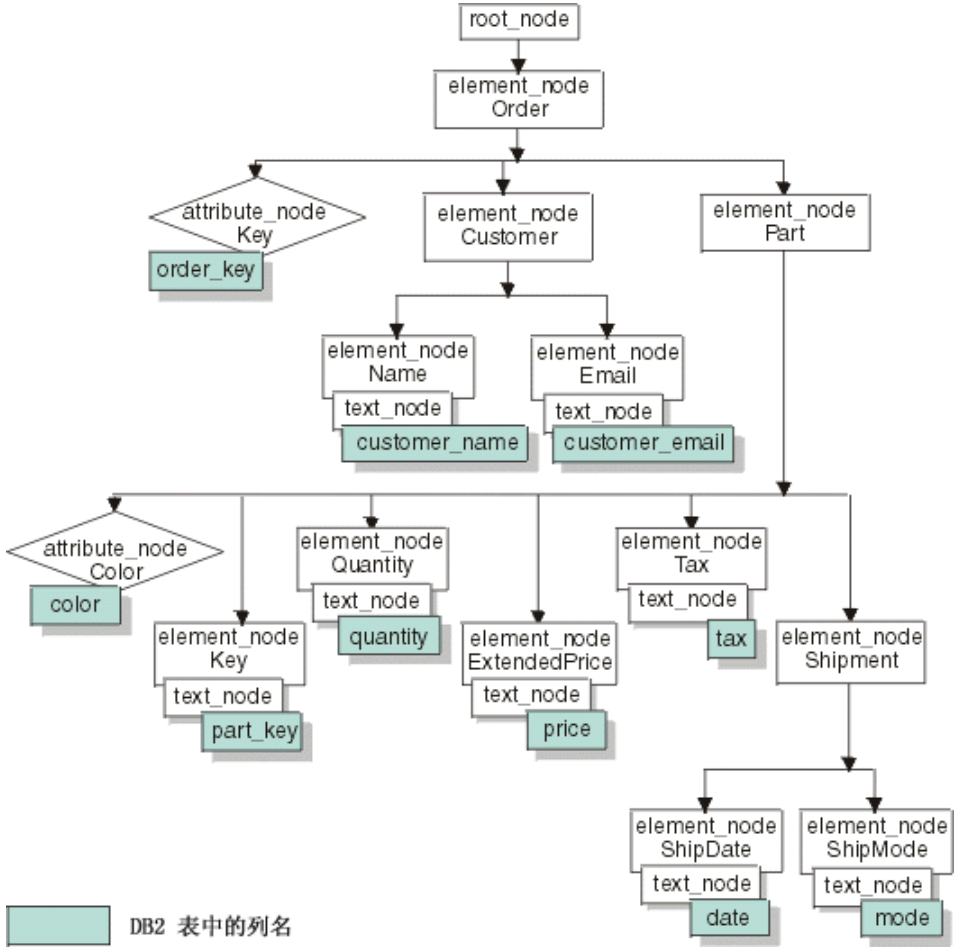


图 11. 映射至关系表列的结构化 XML 文档

XML Extender 在组合或分解位于多个关系表中的 XML 文档时，使用该映射模式。XML Extender 提供了向导，可帮助您创建 DAD 文件。但是，在创建 DAD 文件之前，必须考虑如何将 XML 数据映射至 XML 集合。

映射模式的类型: 映射模式是在 DAD 文件的 <Xcollection> 元素中指定的。XML Extender 提供了两种类型的映射模式: *SQL 映射*和 *"关系数据库" (RDB_node) 映射*。两种方法都使用 XSLT 模型来定义 XML 文档的层次结构。

SQL 映射

允许通过一个 SQL 语句和 *XSLT 数据模型*，简单而直接地从关系数据映射至 XML 文档。 SQL 映射用于组合；而不用于分解。SQL 映射是使用 DAD 文件中的 SQL_stmt 元素定义的。SQL_stmt 的内容是有效的 SQL 语句。SQL_stmt 将 SELECT 子句中的列映射至在 XML 文档中使用的 XML 元素或属性。当为组合 XML 文档而进行定义时，SQL 语句的 SELECT 子句中的列名用于定义 *attribute_node* 的值或 *text_node* 的内容。FROM 子句定义包含数据的表；而 WHERE 子句指定*连接 (join)* 并搜索*条件 (condition)*。

SQL 映射使 DB2 用户能使用 SQL 映射数据。当使用 SQL 映射时，必须能够将一个 SELECT 语句中的所有表连接起来，以形成一个查询。如果一个 SQL 语句不够，可考虑使用 RDB_node 映射。 要将所有的表紧密联系在一起，建议使用这些表间的*主键*和*外键*关系。

RDB_node 映射

定义 XML 元素的内容的位置或 XML 属性的值，从而 XML Extender 可确定存储或检索 XML 数据的位置。

RDB_node 包含了一个或多个用于表、可选列和可选条件的节点定义。这些表和列用于定义如何将 XML 数据存储在数据库中。该条件指定选择 XML 数据的标准或连接 XML 集合表的方法。

要定义映射模式，用 <Xcollection> 元素创建 DAD。第56页的图12 显示了一个 DAD 文件样本的片段，该文件使用 XML 集合 SQL 映射根据三个关系表中的数据组合一组 XML 文档。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:\dtd\dad.dtd">
<DAD>
  <dtdid>c:\dxx\samples\dad\getstart.dtd</dtdid>
  <validation>YES</validation>
  <Xcollection>
    <SQL_stmt>
       SELECT o.order_key, customer, p.part_key, quantity, price, tax, date,
              mode, comment
         FROM order_tab o, part_tab p,
               table(select substr(char(timestamp(generate_unique())),
               as ship_id, date, mode, from ship_tab) as s
        WHERE p.price > 2500.00 and s.date > "1996-06-01" AND
              p.order_key = o.order_key and s.part_key = p.part_key
    </SQL_stmt>
    <prolog>?xml version="1.0"?</prolog>
    <doctype>!DOCTYPE DAD SYSTEM "c:\dxx\samples\dtd\getstart.dtd"</doctype>
    <root_node>
   <element_node name="Order">
  <attribute_node name="key">
<column_name="order_key"/>
  </attribute_node>
  <element_node name="Customer">
  <text_node>
  <column name="customer"/>
  </text_node>
  <element_node>

...

  </element_node><!-end Part->
   </element_node><!-end Order->
    </root_node>
  </Xcollection>
</DAD>
```

图 12. SQL 映射模式

XML Extender 提供了管理 XML 集合中的数据的几个存储过程。这些存储过程支持两种类型的映射，但需要 DAD 文件遵循『映射模式需求』中所描述的规则。

映射模式需求: 下列部分描述每一类型的 XML 集合映射模式的需求。

使用 SQL 映射时的需求

在此映射模式中，必须指定 DAD <Xcollection> 元素中的 SQL_stmt 元素。 该 SQL_stmt 应包含一个 SQL 语句，该语句可将多个关系表与查询*谓词*连接在一起。 此外，还需要下列子句:

- **SELECT 子句**
  - 确保列名是唯一的。如果两个表具有同一列名，可使用 AS 关键字来为它们其中一个创建别名。

– 将同一表中的列分组在一起，并使用关系表的逻辑分层级。这意味着根据这些表映射至 XML 文档的层次结构时的重要性级别，来对它们进行分组。在 SELECT 子句中，较高级表的列应在较低级表的列之前。下例示范了表间的分层关系:

```
SELECT o.order_key, customer, p.part_key, quantity, price, tax,
       ship_id, date, mode
```

在本例中，表 ORDER_TAB 中的 order_key 和 customer 具有最高的关系级，因为它们在 XML 文档的分层树中处于较高的位置。表 SHIP_TAB 中的 ship_id、date 和 mode 处于最低的关系级。

– 使用单列候选键来开始每一级别。如果表中没有这样的键可用，该查询应使用表表达式和内部函数 generate_unique()，对该表生成一个键。在上述示例中， o.order_key 是 ORDER_TAB 的主键，而 part_key 是 PART_TAB 的主键。它们在要选择的列中，所属的组的起始处出现。因为 SHIP_TAB 表没有主键，所以需要生成一个，在本例中为 ship_id。它是作为 SHIP_TAB 表组的首列而列示的。使用 FROM 子句生成主键列，如以下示例中所示。

- **FROM 子句**
  – 使用表表达式和内部函数 generate_unique() 生成表的单键，这些表不具有主单键。例如:

```
FROM order_tab as o, part_tab as p,
        table(select substr(char(timestamp(generate_unique())),16) as
        ship_id, date, mode from ship_tab) as s
```

在本示例中， generate_unique() 函数被强制转型为 TIMESTAMP 的 CHAR 数据类型，且给予它别名 ship_id。

– 当需要使一列有别于其他列时，使用别名。例如，您可将 o 用于 ORDER_TAB，将 p 用于 PART_TAB，而将 s 用于 SHIP_TAB。

- **WHERE 子句**
  – 将主键和外键关系指定为将集合中的表紧密联系在一起的连接条件。例如:

```
WHERE p.price > 2500.00 AND s.date > "1996-06-01" AND
      p.order_key = o.order_key AND s.part_key = p.part_key
```

– 指定谓词中的任何其他搜索条件。可使用任何有效的谓词。

- **ORDER BY 子句**
  – 在 SQL_stmt 的末尾处定义 ORDER BY 子句。
  – 确保各列名与 SELECT 子句中的各列名相匹配。

– 指定列名或标识符，它们唯一地标识数据库的实体关系设计中的实体。可使用表表达式和内部函数 generate_unique 或用户定义函数 (UDF) 生成标识符。
– 维护实体自上而下的层次结构。在 ORDER BY 子句中指定的列必须是对每个实体列示的第一列。保持该次序可确保要生成的 XML 文档不包含不正确的复制。
– 不要以任何模式或表名来限定 ORDER BY 中的列。

尽管 SQL_stmt 具有上述需求，但它的功能仍非常强大，因为您可在 WHERE 子句中指定任何谓词（只要该谓词中的表达式使用表中的列）。

使用 RDB_node 映射时的需求

当使用此映射方法时，不要使用 DAD 文件的 <Xcollection> 元素中的 SQL_stmt 元素。 而应该将每个顶部节点中的 RDB_node 元素用于 *element_node* 和每个 attribute_node 及 text_node。

- **顶部 element_node 的 RDB_node**

  DAD 文件中的 *top element_node* 表示 XML 文档的根元素。指定顶部 element_node 的 RDB_node 如下所述:

  – 指定与各 XML 文档相关联的所有表。例如，以下映射指定 element_node <Order> （它是 top element_node）的 RDB_node 中的三个表:

```
<element_node name="Order">
        <RDB_node>
        <table name="order_tab"/>
            <table name="part_tab"/>
        <table name="ship_tab"/>
            <condition>
order_tab.order_key = part_tab.order_key AND
part_tab.part_key = ship_tab.part_key
            </condition>
        </RDB_node>
```

  若在集合中仅有一张表，则条件元素可为空。

  – 如果正在进行分解，或正在启用由 DAD 文件指定的 XML 集合，则必须对每个表指定一个主键。主键可由一列或多列组成，称为组合键。主键是通过将属性键添加至 RDB_node 的表元素指定的。 当提供了一个组合键时，键属性是通过由空格隔开的键列名指定的。 例如:

```
<table name="part_tab" key="part_key, price"/>
```

  当组合文档时，忽略对分解指定的信息。

– 使用 orderBy 属性来重新组合 XML 文档，这些文档包含多次返回其初始结构的元素或属性。 此属性允许指定将要用作保留文档次序的键的列名。 orderBy 属性是 DAD 文件中的表元素的一部分，且它是可选属性。

您必须明白无误地拼出表名和列名。

- **每个 attribute_node 和 text_node 的 RDB_node**

  在此映射模式中，数据驻留在每个 element_node 的 attribute_node 和 text_node 上。因此，XML Extender 需要知道从数据库中的何处查找数据。需要对每个 attribute_node 和 text_node 指定 RDB_node，以告知存储过程，从哪个表、哪列以及什么查询条件下获取数据。您必须指定表值和列值；条件值是可选的。

  – 指定包含列数据的表名。表名必须包括在顶部 element_node 的 RDB_node 内。 在本例中，对于元素 <Price> 的 text_node，该表被指定为 PART_TAB。

```
<element_node name="Price">
    <text_node>
        <RDB_node>
            <table name="part_tab"/>
            <column name="price"/>
            <condition>
                 price > 2500.00
            </condition>
        </RDB_node>
    </text_node>
</element_node>
```

  – 指定包含元素文本的数据的列名。在上一个示例中，该列被指定为 PRICE。

  – 如果想要使用查询条件来生成 XML 文档，则指定一个条件。在上述示例中，条件被指定为 `price > 2500.00`。 只有满足条件的数据才会在生成的 XML 文档中。该条件必须为有效 WHERE 子句。

  – 如果正在分解一个文档，或正在启用由 DAD 文件指定的 XML 集合，则必须对每个属性节点和 text_node 指定列类型。这确保了在启用 XML 集合期间创建新表时，每列都具有正确的数据类型。列类型是通过将属性类型添加至列元素来指定的。例如，

```
<column name="order_key" type="integer"/>
```

    当组合文档时，忽略对分解指定的信息。

使用 RDB_node 映射方法，您不需要提供 SQL 语句。 但是，将复杂的查询条件放入 RDB_node 元素可能会是更困难的事。 例如，使用 *并集*、表达式或操作的功能只比使用 SQL 至 XML 方法稍逊一筹。

分解表大小需求

通过将元素和属性值抽取到表行中，分解使用 RDB_node 映射来指定将 XML 文档分解为 DB2 表的方式。每个 XML 文档的值都被存储在一个或多个 DB2 表中。每个表自每个文档最多可分解 1024 行。

例如，如果 XML 文档被分解为五个表，则对于该特定文档，这五个表中的每一个最多可有 1024 行。如果该表包含有多个文档的多个行，则对于每个文档，该表最多可有 1024 行。如果该表有 20 个文档，则它可有 20,480 行，即每个文档 1024 行。

使用多次出现的元素（具有在 XML 结构中可多次出现的位置路径的元素）对行数会有影响。例如，包含出现了 20 次的元素 <Part> 的文档在表中可能被分解为 20 行。使用多次出现的元素时，要考虑此表大小限制。

第4章 管理 XML 数据

XML Extender 管理任务由启用数据库和 XML 的表列，以及将 XML 数据映射至 DB2 关系结构组成。 XML Extender 提供了几个管理工具供您使用，这取决于您是否想开发一个应用程序来执行管理任务，或是否只是想使用向导。您可使用下列工具以完成 XML Extender 的管理任务:

- XML Extender 管理向导
- **dxxadm** 命令
- XML Extender 管理存储过程

本章描述与管理向导相关联的管理任务和 **dxxadm** 命令。管理存储过程在第187页的『管理存储过程』中作了描述。

要完成本章中的任务，您应熟悉第42页的『管理计划』中描述的概念及计划任务。

以下几节描述 XML Extender 管理任务:

1. 『启动管理向导』
2. 第64页的『对 XML 启用数据库』
3. 第65页的『将 DTD 存储在 DTD 库中』
4. 第67页的『定义 XML 列或集合』
5. 第67页的『使用 XML 列』
6. 第77页的『使用 XML 集合』

启动管理向导

本节包含有关设置和调用 XML Extender 管理向导的信息。

设置管理向导

确保您已遵循了针对您的操作系统的自述文件中，有关管理向导的安装和配置步骤。包括确保您已经运行了绑定语句并且已在 CLASSPATH 语句中包括了必需的软件。

- 绑定语句在向导自述文件和入门样本文件中提供:

  `/dxx_install/samples/cmd/getstart_prep.cmd`

- CLASSPATH 语句看去应象如下所示（分行仅用于呈示）:

© Copyright IBM Corp. 1999, 2000

```
.;C:\java\db2java.zip;C:\java\runtime.zip;C:\java\sqlj.zip;
C:\dxx\dxxadmin\dxxadmin.jar;C:\dxx\dxxadmin\dxxadmin.cmd;
C:\dxx\dxxadmin\html\dxxahelp*.htm;C:\java\jdk\lib\classes.zip;
C:\java\swingall.jar
```

重要事项: 向导需要一个无空格的路径名。若您具有 IBM DB2 "通用数据库" V7.1 缺省安装，则 SQLLIB\java 位于 Program Files 目录下，将 Java 代码*复制*到一个较简单的路径。不要移动 Java 代码和更改 CLASSPATH; "控制中心" 要求在安装期间指定 CLASSPATH。

"XML Extender 管理" 向导使用类文件。主 "XML Extender 管理" 类文件的完整文件名是:

com.ibm.dxx.admin.Admin。

修改此文件，以便您的系统调用该向导。

- 要使用 JDK 进行调用，输入:

  ```
  java -classpath classpath com.ibm.dxx.admin.Admin
  ```

- 要使用 JRE 进行调用，输入:

  ```
  jre -classpath classpath com.ibm.dxx.admin.Admin
  ```

  其中 *classpath* 指定下列任一项:

  – %CLASSPATH% 环境变量，用于指定管理向导类文件的位置。当使用此选项时，系统 CLASSPATH 必须指向 *dxx_install*/dxxadmin 目录，该目录包含下列文件: dxxadmin.jar、xml4j.jar 和 db2java.zip。例如:

  ```
  java -classpath %CLASSPATH% com.ibm.dxx.admin.Admin
  ```

  – %CLASSPATH% 环境变量的覆盖，其指针指向 *dxx_install*/dxxadmin 目录中的文件，您可以从该目录运行 XML Extender 管理向导。例如:

  ```
  java -classpath dxxadmin.jar;xml4j.jar;db2java.zip com.ibm.dxx.admin.Admin
  url=jdbc:db2:mydb userid=db2xml password=db2xml
  driver=COM.ibm.db2.jdbc.app.DB2Driver
  ```

  您也可以选择在运行时指定下列参数:

  url 要连接的 IBM DB2 UDB 数据源的全限定 URL 路径。例如: jdbc:db2://dxx.stl.ibm.com:8080/guidb。在向导中被标记为 "地址"。

  userid 用来访问以上数据源的用户标识。例如: db2guest。

  password
  以上用户标识的口令。例如: guest。

  driver 以上 URL 的 JDBC 驱动程序名。缺省值: COM.ibm.db2.jdbc.net.DB2Driver。在向导中被标记为 "JDBC 驱动程序"。

有关这些值的更多信息，参见『调用管理向导』。

调用管理向导

遵循下列步骤以调用 XML Extender 管理向导。

1. 调用该向导。

   对于 Windows NT:

   从桌面双击 XML Extender 管理向导图标。

   对于 AIX、Sun Solaris 和 Linux:

   运行 `dxxadmin` 文件。

   “管理向导登录”窗口打开。

   调用 XML Extender 管理向导时，显示“登录”窗口。登录至想要在使用 XML 数据时使用的数据库。XML Extender 连接至当前实例。

2. 在**地址**字段中，输入您正在连接的 IBM DB2 UDB 数据源的全限定 JDBC URL。该地址具有以下语法:

   对于独立配置（建议）:

   `jdbc:db2:`*`database_name`*

   其中:

   database_name
   　您正在连接且用来存储 XML 文档的数据库。

   例如,

   `jdbc:db2:sales_db`

   对于网络配置:

   `jdbc:db2://`*`server_name`*`:`*`port_number`*`/`*`database_name`*

   其中:

   server_name
   　是 XML Extender 所在的服务器的名称。

   port_number
   　用于连接服务器的端口号。要确定端口号，从服务器的 DB2 命令行输入以下命令:

   　`db2jstrt port#`

   　Windows NT 用户可检查文件 \winnt\system32\driver\etc\services 以获取端口号。

database_name
您正在连接且用来存储 XML 文档的数据库。

例如,

```
jdbc:db2://host1.ibm.com:8080/sales_db
```

3. 在**用户标识**和**口令**字段中，输入或验证用于您正在连接的数据库的 DB2 用户标识和口令。
4. 在 **JDBC 驱动程序**字段中，使用下列值来验证指定地址的 JDBC 驱动程序名:

   对于独立配置（缺省配置和建议的配置）:

   ```
   COM.ibm.db2.jdbc.app.DB2DRIVER
   ```

   对于网络配置:

   ```
   COM.ibm.db2.jdbc.net.DB2DRIVER
   ```
5. 单击**完成**以连接至该向导并进至 LaunchPad 窗口。

LaunchPad 窗口提供了对五个管理向导的访问。借助于这些向导，您可以:

- 启用数据库
- 将 DTD 添加至 DTD 库
- 将 DAD 文件用于:
  - XML 列
  - XML 集合
- 使用 XML 列
- 使用 XML 集合

对 XML 启用数据库

要用 XML Extender 来存储或检索 DB2 中的 XML 文档，对 XML 启用数据库。XML Extender 启用与您相连的使用当前实例的数据库。

当对 XML 启用数据库时，XML Extender 会:

- 创建所有用户定义类型 (UDT) 和用户定义函数 (UDF)
- 用 XML Extender 所需的必要元数据创建并填充控制表
- 创建 db2xml 模式并指定必要的特权

  XML 函数的全名是 *schema-name.function-name*，其中 *schema-name* 是为 SQL 对象提供逻辑分组的标识符。 可在引用 UDF 或 UDT 的任何位置使用全名。当引用 UDF 或 UDT 时还可省略模式名，在这种情况下，DB2 使用函数路径以确定想要使用的函数或数据类型。

使用管理向导

使用下列步骤以启用用于 XML 数据的数据库:

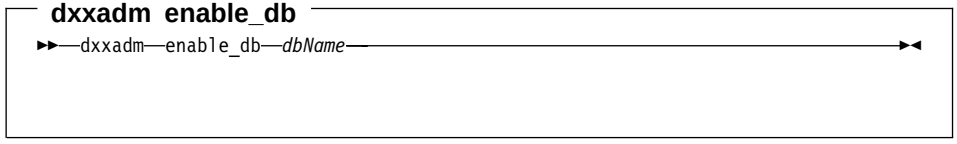
1. 设置并启动管理向导。参见第61页的『启动管理向导』以获取详情。
2. 从 LaunchPad 窗口单击**启用数据库**以启用当前数据库。
   如果已经启用了数据库，则仅可选择**禁用数据库**。
   启用数据库时，会返回至 LaunchPad 窗口。

从 DB2 命令外壳

从命令行输入 **dxxadm**，指定要启用的数据库。

语法:

dxxadm enable_db



参数:

dbName
要启用的数据库的名称。

示例: 启用现存的称为 SALES_DB 的数据库。

```
dxxadm enable_db SALES_DB
```

将 DTD 存储在 DTD 库中

可使用 DTD 来验证 XML 列或 XML 集合中的 XML 数据。 DTD 验证 XML 列，且用来定义那些用于 XML 结构搜索及集合组合和分解的 DAD 文件。

所有的 DTD 都存储在 DTD 库中，这是一个称为 DTD_REF 的 DB2 表。它具有模式名 db2xml。DTD_REF 表中的每个 DTD 都具有唯一标识。当对 XML 启用数据库时， XML Extender 创建 DTD_REF 表。

参见第44页的『计划 XML 列』和第50页的『计划 XML 集合』以了解有关使用 DTD 的更多信息。

可从 DB2 命令外壳或通过使用管理向导插入 DTD。

使用管理向导

使用下列步骤插入 DTD:

1. 设置并启动管理向导。参见第61页的『启动管理向导』以获取详情。
2. 从 LaunchPad 窗口单击**导入 DTD**，以将现存 DTD 文件导入到当前数据库的 DTD 库中。显示“导入 DTD”窗口。
3. 在 **DTD 文件名**字段中输入 DTD 文件名，或单击 **...** 以浏览现存的 DTD 文件。
4. 在 **DTD ID** 字段中输入 DTD ID 。

   DTD ID 是 DTD 的标识符，可为在本地系统上指定 DTD 位置的路径。DTD ID 必须与 <DTDID> 元素的 DAD 文件中指定的值匹配。
5. 可选择在**作者**字段中输入 DTD 作者的姓名。

   如果在 DTD 中已经指定了作者的姓名， XML Extender 会自动显示它。
6. 单击**完成**以将 DTD 插入到 DTD 库表 DB2XML.DTD_REF 中，并返回至 LaunchPad 窗口。

从 DB2 命令外壳

使用表7中的模式对 DTD_REF 表发出 SQL INSERT 语句:

表 7. DTD_REF DTD 表的模式

列名	数据类型	描述
DTDID	VARCHAR(128)	主键（唯一且不为 NULL）。主键用于标识 DTD，且在使用验证时，必须与每个 XML 文档中 DOCTYPE 行上的 SYSTEM 标识相同。当在 DAD 文件中指定主键时，DAD 文件必须遵循由 DTD 定义的模式。
CONTENT	XMLCLOB	DTD 的内容。
USAGE_COUNT	INTEGER	（使用此 DTD 来定义 DAD 的）数据库中的 XML 列数和 XML 集合数。
AUTHOR	VARCHAR(128)	DTD 的作者，供用户输入的可选信息。
CREATOR	VARCHAR(128)	进行首次插入的用户标识。
UPDATOR	VARCHAR(128)	进行最后更新的用户标识。

例如:

```
DB2 INSERT into db2xml.dtd_ref values('c:\dxx\samples\dtd\getstart.dtd',
    db2xml.XMLClobFromFile('c:\dxx\samples\dtd\getstart.dtd'), 0, 'user1',
    'user1', 'user1')
```

有关 XML 集合的重要事项: DTD ID 是一个路径，它指定 DTD 在本地系统上的位置。DTD ID 必须与 <DTDID> 元素的 DAD 文件中指定的值匹配。

定义 XML 列或集合

下列章节描述如何设置和定义用于 XML 列或集合的数据库，以及如何准备必需的数据映射模式。

这些章节为:

- 『使用 XML 列』
- 第77页的『使用 XML 集合』

使用 XML 列

要设置 XML 列，需要定义要访问 XML 数据和启用 XML 表中的 XML 数据列的 DAD 文件。创建 DAD 的一个要点是了解位置路径语法，原因是该语法用来将想要对其创建索引的元素和属性值映射至 DB2 表。参见第48页的『位置路径』以了解有关位置路径及其语法的更多信息。

创建或编辑 DAD 文件

当指定 DAD 文件时，定义需要搜索的数据的属性和键元素。XML Extender 使用此信息来创建辅助表，以便可为数据创建索引从而快速地进行检索。参见第50页的『DAD 文件』以了解有关创建 DAD 文件的计划问题。

开始之前

- 了解 XML 数据的层次结构，从而可定义键元素和属性，以用于创建索引和进行快速搜索。
- 准备 XML 文档的 DTD 并将其插入 DTD_REF 表。此步骤是验证所必需的。

使用管理向导

使用下列步骤以创建 DAD 文件:

1. 设置并启动管理向导。参见第61页的『启动管理向导』以获取详情。
2. 从 LaunchPad 窗口单击**使用 DAD 文件**，以编辑或创建 XML DAD 文件。“指定 DAD 文件”窗口打开。
3. 选择是编辑现存 DAD 文件还是创建新的 DAD 文件。
   - **要编辑现存的 DAD**:
     a. 单击 **...** 以浏览下拉菜单中的现存 DAD 文件，或在**文件名**字段中输入 DAD 文件名。
     b. 验证向导是否识别指定的 DAD 文件。

– 如果向导识别指定的 DAD 文件，则**下一步**是可选择的，且 XML 列显示在**类型**字段中。
– 如果向导不识别指定的 DAD 文件，则**下一步**是不可选择的。重新将 DAD 文件名输入**文件名**字段中，或单击**打开**以再次浏览，以查找现存的 DAD 文件。 继续进行，直到**下一步**是可选择的为止。

c. 单击**下一步**。

- **要创建新 DAD**:

a. 将**文件名**字段留为空白。
b. 从**类型**菜单，单击 **XML 列**。
c. 单击**下一步**。

4. 选择是否使用 DTD 从“选择验证”窗口验证 XML 文档。
   - 要进行验证:
     a. 单击**用 DTD 来验证 XML 文档**。
     b. 从 **DTD ID** 菜单选择要用于验证的 DTD。

     如果还未将任何 DTD 导入数据库的 DTD 库，则不能验证 XML 文档。
   - 单击**不要使用 DTD 来验证 XML 文档**继续进行，而不验证 XML 文档。
5. 单击**下一步**。
6. 选择是添加新辅助表，编辑现存的辅助表，还是从“辅助表”窗口除去现存的辅助表。
   - **要添加新的辅助表或辅助表列**:

     要添加新辅助表，定义表中的各列。对辅助表中的每一列完成下列步骤。
     a. 完成“辅助表”窗口的**详细信息**框中的字段。
        1) **表名**: 输入包含列的表的名称。 例如:

           ```
           ORDER_SIDE_TAB
           ```

        2) **列名**: 输入列的名称。例如:

           ```
           CUSTOMER_NAME
           ```

        3) **类型**: 从菜单选择列的类型。例如:

           ```
           XMLVARCHAR
           ```

        4) **长度（仅限 VARCHAR 类型）**: 输入最大 VARCHAR 字符数。 例如:

           ```
           30
           ```

        5) **路径**: 输入元素或属性的位置路径。例如:

           ```
           /ORDER/CUSTOMER/NAME
           ```

参见第48页的『位置路径』以了解位置路径语法。

6) **多次发生**: 从菜单选择**否**或**是**。

指示此元素或属性的位置路径在一个文档中是否可使用多次。

重要事项: 如果对某一列指定多次发生，则仅可在包含该列的辅助表中指定一列。

b. 单击**添加**以添加一列。

c. 继续添加、编辑或除去辅助表的各列，或单击**下一步**。

- **要编辑现存的辅助表列:**

  可通过更改现存列的定义来更新辅助表。

  a. 单击想要编辑的辅助表和列名。

  b. 编辑**详细信息**框中的字段。

  c. 单击**更改**以保存更改。

  d. 继续添加、编辑或除去每个辅助表的列，或单击**下一步**。

- **要除去现存的辅助表列:**

  a. 单击想要除去的辅助表和列。

  b. 单击**除去**。

  c. 继续添加、编辑或除去辅助表列，或单击**下一步**。

- **要除去现存的辅助表:**

  要除去整个辅助表，删除表中的每一列。

  a. 对于想要除去的表，单击每个辅助表列。

  b. 单击**除去**。

  c. 继续添加、编辑或除去辅助表列，或单击**下一步**。

7. 在 “指定 DAD” 窗口的**文件名**字段中输入修改过的 DAD 文件的输出文件名。
8. 单击**完成**以保存 DAD 文件并返回至 LaunchPad 窗口。

从 DB2 命令外壳

DAD 文件是可在任何文本编辑器中创建的 XML 文件。

使用下列步骤以创建 DAD 文件:

1. 打开文本编辑器。
2. 创建 DAD 文件头（使用下列语法）:

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "path\dtd\dad.dtd"> --> the path and file name of
       the DTD for the DAD file
```

3. 插入 <DAD></DAD> 标记。

4. 在 <DAD> 标记内，可选择指定 DTD ID 标识符，它将 DAD 文件与用于验证的 XML 文档 DTD 相关联:

```
<dtdid>path\dtd_name.dtd</dtdid> --> the path and file
       name of the DTD
       for your application
```

   DTD ID 对于验证来说是必需的，且它必须与在将 DTD 插入到 DTD 参考表 (db2xml.DTD_REF) 中时使用的 DTD ID 值相匹配。

5. 指定是否进行验证（即，使用 DTD 以确保 XML 文档是有效的 XML 文档）。例如:

```
<validation>YES</validation>   --> 指定 YES 或 NO
```

   如果指定 YES，则必须在前一步骤中已指定了 DTD ID，且已将 DTD 插入 DTD_REF 表。

6. 使用 <Xcolumn> 元素将访问方法和存储方法定义为 XML 列。

```
<Xcolumn>
</Xcolumn>
```

7. 定义每个辅助表，以及为用于结构搜索而构建索引的重要元素和属性。对每个表执行下列步骤。下列步骤使用 第241页的『DAD 文件: XML 列』中显示的样本 DAD 文件中采用的示例:

   a. 插入 <TABLE></TABLE> 标记及名称属性。

```
<table name="order_tab">
</table>
```

   b. 在 <TABLE> 标记之后，对表中的每一列插入 <COLUMN> 标记及其属性:

      - ***name***: 列的名称
      - ***type***: 列的类型
      - ***path***: 元素或属性的位置路径。参见第48页的『位置路径』以了解位置路径语法。
      - ***multi_occurrence***: 指示此元素或属性是否可在一个文档中使用多次

```
<table ...>
   <column name="order_key"
           type="integer"
           path="/Order/@key"
           multi_occurrence="NO"/>
   <column name="customer"
           type="varchar(50)"
           path="/Order/Customer/Name"
           multi_occurrence="NO"/>
</table>
```

8. 确保在最后一个列定义后有一个表示结束的 </TABLE> 标记。

9. 确保在最后一个 </TABLE> 标记后有一个表示结束的 </Xcolumn> 标记。
10. 确保在 </Xcolumn> 标记后有一个表示结束的 </DAD> 标记。

创建或改变 XML 表

要将完整的 XML 文档存储在表中，必须创建或改变表以使其包含类型为 XML 用户定义类型 (UDT) 的列。该表被称为 *XML 表*，即包含 XML 文档的表。 该表可以是一个改变过的表也可以是新表。当表包含一个 XML 类型的列时，可启用 XML 的列。

可使用管理向导或使用 DB2 命令外壳来改变具有类型为 XML 的列的现存表。

使用管理向导

1. 设置并启动管理向导。参见第61页的『启动管理向导』以获取详情。
2. 从 LaunchPad 窗口单击**使用 XML 列**。 “选择任务”窗口打开。
3. 单击**添加 XML 列**。 “添加 XML 列”窗口打开。
4. 从**表名**下拉菜单选择表名，或输入想要改变的表的名称。例如:

   ```
   SALES_DB
   ```

5. 在**列名**字段中输入要添加至表的列的名称。例如:

   ```
   ORDER
   ```

6. 从**列类型**下拉菜单中选择该列的 UDT。例如:

   ```
   XMLVARCHAR
   ```

7. 单击**完成**以添加 XML 类型的列。

从 DB2 命令外壳

在 CREATE TABLE 或 ALTER TABLE 语句的列子句中创建或改变具有类型为 XML 的列的表。

示例: 在 sales 应用程序中，您可能想要将 XML 格式化的行项指令存储在某列中，该列被称为名为 SALES_TAB 的应用程序表的 ORDER。此表还有列 INVOICE_NUM 和 SALES_PERSON。因为它是一个小指令，所以使用 XMLVARCHAR 类型来存储它。主键为 INVOICE_NUM。以下 CREATE TABLE 语句创建带有 XML 类型的列的表:

```
CREATE TABLE sales_tab(
             invoice_num  char(6) NOT NULL PRIMARY KEY,
             sales_person     varchar(20),
             order            XMLVarchar);
```

启用 XML 列

要在 DB2 数据库中存储 XML 文档，必须启用 XML 的列。启用列时即准备为该列构建索引，以便可快速搜索该列。还可使用 XML Extender 管理向导或使用 DB2 命令外壳来启用列。该列必须为 XML类型。

当 XML Extender 启用 XML 列时，它:

- 读取 DAD 文件以选择:
  - 对 DAD 文件的 DTD 来验证该 DAD 文件。
  - 从 DTD_REF 表检索 DTD ID（如果指定的话）。
  - 创建辅助表以对 XML 列创建索引。
  - 准备列以包含 XML 数据。
- 可选择创建 XML 表和辅助表的*缺省视图*（如果定义了的话）。
- 指定 ROOT 标识值（如果尚未指定的话）。

启用 XML 列之后，可

- 创建辅助表的索引
- 将 XML 文档插入到 XML 列中
- 查询、更新或搜索 XML 列中的 XML 文档。

开始之前

通过创建或改变带有 XML UDT 的列的 DB2 表，来创建 XML 表。

使用管理向导

使用下列步骤以启用 XML 列:

1. 设置并启动管理向导。参见第61页的『启动管理向导』以获取详情。
2. 从 LaunchPad 窗口单击**使用 XML 列**来查看与 XML Extender 列相关的任务。“选择任务”窗口打开。
3. 单击**启用列**，然后单击**下一步**来启用数据库中的现存表列。
4. 从**表名**字段选择包含 XML 列的表。例如

   ```
   SALES_TAB
   ```

5. 从**列名**字段选择要启用的列。 例如:

   ```
   ORDER
   ```

   该列必须存在，且必须为 XML 类型。
6. 在 **DAD 文件名**字段中输入 DAD 路径和文件名，或单击 **...** 以浏览现存的 DAD 文件 例如:

   ```
   c:\dxx\samples\dad\getstart.dad
   ```

7. 可选择在**表空间**字段中输入现存表空间的名称。

   该表空间包含 XML Extender 创建的辅助表。如果指定表空间，则辅助表是在指定的表空间中创建的。如果未指定表空间，则辅助表是在缺省表空间中创建的。

8. 可选择在**缺省视图**字段中输入缺省视图的名称。

   如果指定了缺省视图，则缺省视图在启用列时自动创建，且连接 XML 表和所有相关的辅助表。

9. 可选择在 **Root 用户标识**字段中输入应用程序表中的主键的列名。建议进行此操作。

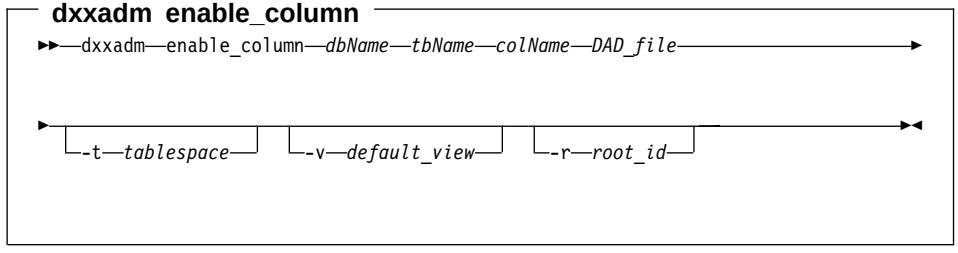
   XML Extender 将 ROOT 标识值用作唯一标识符，以将所有辅助表与应用程序表相关联。如果未指定该 ID，则 XML Extender 会将 DXXROOT_ID 列添加至应用程序表，并生成一个标识符。

10. 单击**完成**以启用 XML 列，创建辅助表，并返回至 LaunchPad 窗口。
    - 如果成功启用了该列，消息启用列成功会显示。
    - 如果未成功启用该列，会显示错误消息框。校正输入字段的值，直到该列被成功启用为止。

从 DB2 命令外壳

要启用 XML 列，输入以下命令:

语法:



参数:

dbName

数据库名称。

tbName

表的名称，该表包含要启用的列。

colName

正在启用的 XML 列的名称。

DAD_file

包含文档访问定义 (DAD) 的文件的名称。

tablespace

先前创建的表空间，它包含 XML Extender 创建的辅助表。如果未指定，则使用缺省表空间。

default_view

可选。缺省视图的名称，该视图由 XML Extender 创建并用来连接应用程序表和所有相关辅助表。

root_id 可选。应用程序表中主键的列名，及将所有辅助表与应用程序表相关联的唯一标识符。XML Extender 将 *root_id* 的值用作唯一标识符，以让所有辅助表与应用程序表相关联。建议指定 ROOT 标识。如果未指定 ROOT 标识，则 XML Extender 将 DXXROOT_ID 列添加至应用程序表，并生成标识符。

限制: 如果应用程序表已有列名 DXXROOT_ID，但此列不包含 *root_id* 的值，您必须指定 *root_id* 参数；否则将发生错误。

示例: 以下示例使用 DB2 命令外壳来启用列。DAD 文件和 XML 文档可在第239页的『附录B. 样本』中找到。

```
dxxadm enable_column SALES_DB sales_tab order getstart.dad
            -v sales_order_view -r invoice_num
```

在本示例中，列 ORDER 是在表 SALES_DB.SALES_TAB 中启用的。 DAD 文件为 `getstart.dad`，缺省视图为 sales_order_view，而 ROOT 标识为 INVOICE_NUM。

使用本示例，SALES_TAB 表具有以下模式:

列名	INVOICE_NUM	SALES_PERSON	ORDER
数据类型	CHAR(6)	VARCHAR(20)	XMLVARCHAR

下列辅助表是根据 DAD 规范创建的:

ORDER_SIDE_TAB:

列名	ORDER_KEY	CUSTOMER	INVOICE_NUM
数据类型	INTEGER	VARCHAR(50)	CHAR(6)
路径表达式	`/Order/@key`	`/Order/Customer/Name`	N/A

PART_SIDE_TAB:

列名	PART_KEY	PRICE	INVOICE_NUM
数据类型	INTEGER	DOUBLE	CHAR(6)
路径表达式	/Order/Part/@key	/Order/Part/ExtendedPrice	N/A

SHIP_SIDE_TAB:

列名	DATE	INVOICE_NUM
数据类型	DATE	CHAR(6)
路径表达式	/Order/Part/Shipment/ShipDate	N/A

所有辅助表都具有同一类型的列 INVOICE_NUM，原因是 ROOT 标识是由应用程序表中的主键 INVOICE_NUM 指定的。在启用列之后，值 INVOICE_NUM 被插入辅助表中。在启用 XML 列和 ORDER 时指定 *default_view* 参数，就创建了缺省的视图 sales_order_view。该视图使用以下语句连接上述表:

```
CREATE VIEW sales_order_view(invoice_num, sales_person, order,
                                order_key, customer, part_key, price, date)
     AS
     SELECT sales_tab.invoice_num, sales_tab.sales_person, sales_tab.order,
            order_tab.order_key, order_tab.customer,
            part_tab.part_key, part_tab.price,
            ship_tab.date
     FROM sales_tab, order_tab, part_tab, ship_tab
     WHERE sales_tab.invoice_num = order_tab.invoice_num
               AND sales_tab.invoice_num = part_tab.invoice_num
               AND sales_tab.invoice_num = ship_tab.invoice_num
```

如果在 **enable_column** 命令中指定了表空间，则在指定的表空间中创建辅助表 如果未指定表空间，则在缺省的表空间中创建辅助表。

对辅助表创建索引

启用 XML 列并创建辅助表之后，可创建辅助表的索引。辅助表包含创建 DAD 文件时指定的列中的 XML 数据。创建这些表的索引有助于改进对 XML 文档进行的查询的性能。

开始之前

- 创建 DAD 文件，它指定 XML 文档结构的辅助表。
- 使用 DAD 文件启用 XML 列；这些辅助表就是由该文件创建的。

DB2 CREATE INDEX 命令

使用 DB2 CREATE INDEX 命令。

示例:

下例对四个辅助表创建索引:

```
DB2 CREATE INDEX KEY_IDX
      ON ORDER_SIDE_TAB(ORDER_KEY)

DB2 CREATE INDEX CUSTOMER_IDX
      ON ORDER_SIDE_TAB(CUSTOMER)

DB2 CREATE INDEX PRICE_IDX
      ON PART_SIDE_TAB(PRICE)

DB2 CREATE INDEX DATE_IDX
      ON SHIP_SIDE_TAB(DATE)
```

禁用 XML 列

如果需要更新 XML 列的 DAD 文件，或如果想要删除 XML 列或包含该列的表，则禁用该列。禁用该列之后，可重新启用该列（用更新过的 DAD 文件进行）、删除该列或执行其他任务。可通过使用 XML Extender 管理向导或使用 DB2 命令外壳来禁用某列。

当 XML Extender 启用 XML 列时，它:

- 从 XML_USAGE 表删除该列的项。
- 删除与此列相关联的辅助表。

重要事项: 如果删除具有 XML 列的表，而不首先禁用该列，则 XML Extender 不能删除与 XML 列相关联的任何辅助表，原因是这样做可能会导致意外的结果。

开始之前

确保要禁用的 XML 列存在于当前 DB2 数据库之中。

使用管理向导

使用下列步骤来禁用 XML 列:

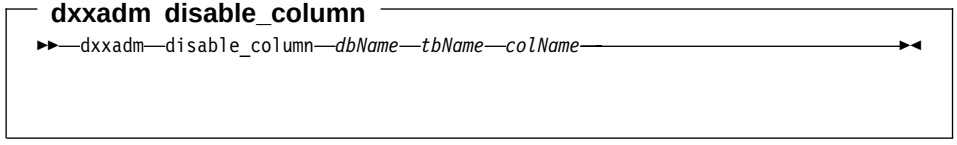
1. 设置并启动管理向导。参见第61页的『启动管理向导』以获取详情。
2. 从 LaunchPad 窗口单击**使用 XML 列**，以查看与 XML Extender 列相关的任务。“选择任务”窗口打开。
3. 单击**禁用列**，然后单击**下一步**来禁用数据库中的现存表列。
4. 从**表名**字段选择包含 XML 列的表。
5. 从**列名**字段中选择被禁用的列。

6. 单击**完成**。
   - 如果成功地禁用了该列，则显示禁用列成功消息。
   - 如果未成功禁用该列，则显示一个错误框。校正输入字段的值，直到成功禁用该列为止。

从 DB2 命令外壳

要禁用 XML 列，输入以下命令:

语法:



参数:

dbName
数据库名称。

tbName
包含要禁用列的表的名称。

colName
被禁用的 XML 列的名称。

示例: 以下示例使用 DB2 命令外壳来禁用列。 DAD 文件和 XML 文档可在第239页的『附录B. 样本』中找到。

```
dxxadm disable_column SALES_DB sales_tab order
```

在本示例中，列 ORDER 在表 SALES_DB.SALES_TAB 中被禁用。

该列被禁用时，辅助表被删除。

使用 XML 集合

设置 XML 集合需要创建映射模式并可选择使用虚拟名来启用集合，该虚拟名将 DB2 表与 DAD 文件相关联。

尽管启用 XML 集合不是必需的，但它确实提供了更好的性能。

创建或编辑 DAD 文件用于映射模式

当使用 XML 集合时创建 DAD 文件是必需的。DAD 文件定义了 XML 数据和多个关系表之间的关系。XML Extender 使用 DAD 文件以:

- 将关系数据组合成 XML 文档
- 将 XML 文档分解成关系数据

可使用以下两个方法中的任一个来在 XML 表与 DB2 表之间映射数据: SQL 映射和 RDB_node 映射:

SQL 映射

使用 SQL 语句元素来对用来存放 XML 数据的表和列指定 SQL 查询。SQL 映射仅可用于组合 XML 文档。

RDB_node 映射

使用 XML Extender 唯一的元素: "关系数据库" 节点, 亦即 RDB_node, 该节点指定表、列、条件和 XML 数据的次序。RDB_node 映射支持比 SQL 语句所能提供的映射更为复杂的映射。RDB_node 映射可用于组合和分解 XML 文档。

这两种映射方法都使用 *XPath 数据模型*, 在第51页的『DAD 文件』中描述了该模型。

开始之前

- 映射 DB2 表与 XML 文档之间的关系。此步骤应包括映射 XML 文档的层次以及指定文档中的数据是如何映射至 DB2 表的。
- 如果计划验证 XML 文档, 则将您正在组合或分解的 XML 文档的 DTD 插入到 DTD 参考表 db2xml.DTD_REF 中。

使用 SQL 映射来组合 XML 文档

在组合 XML 文档且想要使用 SQL 时使用 SQL 映射。

使用管理向导: 借助于下列步骤, 使用 XML 集合 SQL 映射来创建 DAD 文件

要使用 SQL 映射来创建 DAD 文件以进行组合:

在组合 XML 文档且想要使用 SQL 语句来在 XML 文档中定义表和列(将从这些表和列中派生数据)时, 使用 SQL 映射。

1. 设置并启动管理向导。参见第61页的『启动管理向导』以获取详情。
2. 从 LaunchPad 窗口单击**使用 DAD 文件**。显示 "指定 DAD" 窗口。
3. 选择是编辑现存 DAD 文件还是创建新的 DAD 文件。

   要创建新 DAD 文件:

a. 将**文件名**字段留为空白。

b. 从**类型**菜单中，选择 **XML 集合 SQL 映射**。

c. 单击**下一步**以打开“选择验证”窗口。

要编辑现存的 DAD 文件:

a. 在**文件名**字段中输入 DAD 文件名，或单击 **...** 以浏览现存的 DAD 文件

b. 验证向导是否识别指定的 DAD 文件。

- 如果向导识别指定的 DAD 文件，则**下一步**是可选择的，且 XML 集合 SQL 映射显示在**类型**字段中。
- 如果向导不识别指定的 DAD 文件，则**下一步**是不可选择的。重新输入 DAD 文件名，或单击 **...** 以再次浏览现存的 DAD 文件 校正输入字段的值，直到可选择**下一步**为止。

c. 单击**下一步**以打开“选择验证”窗口。

4. 在“选择验证”窗口中，选择是否使用 DTD 来验证 XML 文档。
   - 要进行验证:
     a. 单击**用 DTD 来验证 XML 文档**。
     b. 从 **DTD ID** 菜单选择要用于验证的 DTD。

     如果还未将任何 DTD 导入数据库的 DTD 库，则不能验证 XML 文档。
   - 单击**不要使用 DTD 来验证 XML 文档**继续进行，而不验证 XML 文档。
5. 单击**下一步**以打开“指定文本”窗口。
6. 在 **Prolog** 字段中输入 prolog 名，以指定要组合的 XML 文档的 prolog。

   ```
   <?xml version="1.0"?>
   ```

   如果正在编辑现存的 DAD，则 prolog 会自动显示在 **Prolog** 字段中。
7. 在“指定文本”窗口的 **Doctype** 字段中输入 XML 文档的文档类型，并指向 XML 文档的 DTD 例如:

   ```
   ! DOCTYPE DAD SYSTEM "c:\dxx\samples\dtd\getstart.dtd"
   ```

   如果正在编辑现存的 DAD，则文档类型会自动显示在 **Doctype** 字段中。
8. 单击**下一步**以打开“指定 SQL 语句”窗口。
9. 在 **SQL 语句**字段中输入有效的 SQL SELECT 语句。例如:

   ```
   SELECT o.order_key, customer_name, customer_email, p.part_key, color, quantity,
      price, tax, ship_id, date, mode from order_tab o, part_tab p,
   table (select substr(char(timestamp(generate_unique())),16)
        as ship_id, date, mode, part_key from ship_tab) s
            WHERE o.order_key = 1 and
                  p.price > 20000 and
   ```

```
        p.order_key = o.order_key and
        s.part_key = p.part_key
ORDER BY order_key, part_key, ship_id
```

如果正在编辑现存的 DAD，则 SQL 语句会自动显示在 **SQL 语句**字段中。

10. 单击**测试 SQL** 以测试 SQL 语句的有效性。
    - 如果 SQL 语句有效，则**样本结果**字段中会显示样本结果。
    - 如果 SQL 语句无效，则**样本结果**字段中会显示错误消息。错误消息会指示您校正 SQL SELECT 语句并再次尝试。
11. 单击**下一步**以打开“SQL 映射”窗口。
12. 通过在“SQL 映射”窗口左边的字段中单击元素或属性节点来选择要自其进行映射的元素或属性节点。

    将 XML 文档中的元素和属性映射至对应于 DB2 数据的元素和属性节点。 这些节点提供从 XML 数据至 DB2 数据的路径。

    - **要添加根节点**:
      a. 选择**根**图标。
      b. 单击**新元素**以定义新节点。
      c. 在**详细信息**框中，将**节点类型**指定为**元素**。
      d. 在**节点名**字段中输入顶级节点的名称。
      e. 单击**添加**以创建新节点。

      您已经创建了根节点或元素，它是映象中所有其他元素或属性节点的父代。 现在可将子元素和属性添加至此节点。
    - **要添加子元素或属性节点**:
      a. 单击左边字段中的父代节点，以添加子元素或属性。

      如果未选择父代节点，则不可选择**新元素**。
      b. 单击**新元素**。
      c. 从**详细信息**框中的**节点类型**菜单选择节点类型。

      节点类型菜单仅显示在映象中的该位置上有效的节点类型:

      元素 表示在与 XML 文档相关联的 DTD 中定义的 XML 元素。用来将 XML 元素与 DB2 表中的列相关联。元素节点可具有属性节点、子元素节点或文本节点。底级节点具有文本节点，且在树视图中具有与该底级节点相关联的列名。

      属性 表示在与 XML 文档相关联的 DTD 中定义的 XML 属性。它用来将 XML 属性与 DB2 表中的列相关联。属性节点可具有文本节点，且在树视图中具有与该属性节点相关联的列名。

文本 对有内容要映射至关系表的元素或属性节点指定文本内容。文本节点在树视图中具有与其相关联的列名。

表 对要映射至关系表的元素或属性值指定表名。

列 对要映射至关系表的元素或属性值指定列名。

条件 对该列指定条件。

d. 在**详细信息**框中的**节点名**字段中输入节点名。 例如:

```
Order
```

e. 如果将底级元素的**属性**、**元素**或**文本**指定为了“节点”类型，则从**详细信息**框的**列**字段中选择一列。例如:

```
Customer_Name
```

限制: 不能使用管理向导来创建新列。 如果将**列**指定为节点类型，则您仅可选择已经存在于 DB2 数据库中的列。

f. 单击**添加**以添加新节点。

稍后，可通过单击左边字段中的某节点并在**详细信息**框中对其进行任何必要的修改，来修改该节点。单击**更改**来更新该元素。

还可通过突出显示重复添加过程的节点来将子元素或属性添加至该节点。

g. 继续编辑 SQL 映射，或单击**下一步**，以打开“指定 DAD”窗口。

- **要除去节点**:

a. 单击左边的字段中的某个节点。

b. 单击**除去**。

c. 继续编辑 SQL 映射，或单击**下一步**，以打开“指定 DAD”窗口。

注意，如果除去了底级节点，则另一元素将成为底级节点，且可能需要对其定义列名。

13. 在“指定 DAD”窗口的**文件名**字段中输入修改过的 DAD 文件的输出文件名。
14. 单击**完成**以返回至 LaunchPad 窗口。

从 DB2 命令外壳: 在组合 XML 文档且想要使用 SQL 时使用 SQL 映射表示法。

DAD 文件是可使用任何文本编辑器创建的 XML 文件。下列步骤显示样本附录第240页的『文档访问定义文件』中的分段。请参考这些样本以获取更多综合信息和上下文。

1. 打开文本编辑器。
2. 创建 DAD 头:

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "path\dad.dtd"> --> the path and file name of the DTD
       for the DAD
```

3. 插入 <DAD></DAD> 标记。
4. 在 <DAD> 标记后面，指定将 DAD 文件与 XML 文档 DTD 相关联的 DTD ID。

```
<dtdid>path\dtd_name.dtd> --> the path and file name
       of the DTD for your application
```

5. 指定是否进行验证（即，使用 DTD 以确保 XML 文档是有效的 XML 文档）。例如:

```
<validation>NO</validation>   --> 指定 YES 或 NO
```

6. 使用 <Xcollection> 元素将访问方法和存储方法定义为 XML 集合。访问和存储方法定义 XML 文档将把派生自数据的内容存储在 DB2 表中。

```
<Xcollection>
</Xcollection>
```

7. 指定一个或多个 SQL 语句来从 DB2 表查询数据或将数据插入到 DB2 表中。有关准则，参见第56页的『映射模式需求』。例如，指定单个 SQL 查询，如下例中所示:

```
 <SQL_stmt>
  SELECT o.order_key, customer_name, customer_email, p.part_key, color, quantity,
  price, tax, ship_id, date, mode from order_tab o, part_tab p,
  table (select substr(char(timestamp(generate_unique())),16)
    as ship_id, date, mode, part_key from ship_tab) s
        WHERE o.order_key = 1 and
              p.price > 20000 and
              p.order_key = o.order_key and
              s.part_key = p.part_key
        ORDER BY order_key, part_key, ship_id
 </SQL_stmt>
```

8. 添加以下 prolog 信息:

```
<prolog>?xml version="1.0"?</prolog>
```

   需要与以上文本完全一致。

9. 添加 <doctype></doctype> 标记。例如:

```
<doctype>! DOCTYPE Order SYSTEM "c:\dxx\samples\dtd\getstart.dtd"</doctype>
```

10. 使用 <root_node></root_node> 标记来定义根节点。在 root_node 内部，指定构成 XML 文档的元素和属性。
11. 将 XML 文档中的元素和属性映射至对应于 DB2 数据的元素和属性节点。 这些节点提供从 XML 数据至 DB2 数据的路径。
    a. 对 XML 文档中映射至 DB2 表中的列的每一元素定义 <element_node>。

```
<element_node name="name"></element_node>
```

元素节点（element_node）可具有下列节点。

- attribute_node
- child element_node
- text_node

b. 对 XML 文档中的每个属性定义 <attribute_node>，该文档中的每个属性都映射至 DB2 表中的一列。参见本节开头的示例 DTD 以了解 SQL 映射，并参见第233页的『附录A. 用于 DAD 文件的 DTD』（它提供了 DAD 文件的全部语法）中 DAD 文件的 DTD。

例如，您需要用于元素 <Order> 的属性键。键值存储在列 PART_KEY 中。

DAD 文件: 在 DAD 文件中，创建键的属性节点并指示要存储值 1 的表。

```
<attribute_node name="key">
    <column name="part_key"/>
</attribute_node>
```

组合的 XML 文档: 键值取自 PART_KEY 列。

```
<Order key="1">
```

12. 为具有将派生自 DB2 表的内容的每个元素或属性创建 <text_node>。该文本节点具有 <column> 元素，该元素指定提供该内容的列。

例如，您可能具有列 XML 元素 <Tax>，该元素的值将取自列 TAX:

DAD 元素:

```
<element_node name="Tax">
   <text_node>
         <column name="tax"/>
   </text_node>
</element_node>
```

列名必须在 DAD 文件开头的 SQL 语句中。

组合的 XML 文档:

```
<Tax>0.02</Tax>
```

值 0.02 将派生自列 TAX。

13. 确保在最后一个 </element_node> 标记后有一个表示结束的 </root_node> 标记。
14. 确保在 </root_node> 标记后有一个表示结束的 </Xcollection> 标记。
15. 确保在 </Xcollection> 标记后有一个表示结束的 </DAD> 标记。

使用 RDB_node 映射来组合 XML 文档

使用 RDB_node 映射来组合使用类似于 XML 的结构的 XML 文档。

此方法使用 <RDB_node> 来对元素或属性节点指定 DB2 表、列和条件。<RDB_node> 使用下列元素:

- <table>: 定义对应于元素的表
- <column>: 定义包含对应元素的列
- <condition>: 可选择指定列的条件

在 <RDB_node> 中使用的子元素取决于该节点的内容，并使用下列规则:

如果节点类型是:	使用 RDB 子元素:		
	表	列	条件 [1]
根元素	是	否	是
属性	是	是	可选
文本	是	是	可选

(1) 需要有多个表

使用管理向导: 要创建用于组合的 DAD，使用 RDB_node 映射:

1. 设置并启动管理向导。参见第61页的『启动管理向导』以获取详情。
2. 从 LaunchPad 窗口单击**使用 DAD 文件**。显示“指定 DAD ”窗口。
3. 选择是编辑现存 DAD 文件还是创建新 DAD。

   要编辑现存的 DAD:

   a. 在**文件名**字段中输入 DAD 文件名，或单击 **...** 以浏览现存的 DAD
   b. 验证向导是否识别指定的 DAD 文件。
      - 如果向导识别指定的 DAD 文件，则**下一步**是可选择的，且 XML 集合 RDB 节点映射显示在**类型**字段中。
      - 如果向导不识别指定的 DAD 文件，则**下一步**是不可选择的。在**文件名**字段中重新输入 DAD 文件名，或单击 **...** 以再次浏览现存的 DAD 文件 继续这些步骤，直到**下一步**是可选的为止。
   c. 单击**下一步**以打开“选择验证”窗口。

   要创建新 DAD:

   a. 将**文件名**字段留为空白。
   b. 从**类型**菜单选择 XML 集合 RDB_node 映射。
   c. 单击**下一步**以打开“选择验证”窗口。

4. 在“选择验证”窗口中，选择是否使用 DTD 来验证 XML 文档。
   - 要进行验证:
     a. 单击**用 DTD 来验证 XML 文档**。
     b. 从 **DTD ID** 菜单选择要用于验证的 DTD。

     如果还未将任何 DTD 导入数据库的 DTD 库，则不能验证 XML 文档。
   - 单击**不要使用 DTD 来验证 XML 文档**继续进行，而不验证 XML 文档。
5. 单击**下一步**以打开“指定文本”窗口。
6. 在“指定文本”窗口的 **Prolog** 字段中输入 prolog 名

   ```
   <?xml version="1.0"?>
   ```

   如果正在编辑现存的 DAD，则 prolog 会自动显示在 **Prolog** 字段中。
7. 在“指定文本”窗口的 **Doctype** 字段中输入 XML 文档的文档类型。

   如果正在编辑现存的 DAD，则文档类型会自动显示在 **Doctype** 字段中。
8. 单击**下一步**以打开“RDB 映射”窗口。
9. 通过在“RDB 映射”窗口左边的字段中单击元素或属性节点来选择要自其进行映射的元素或属性节点。

   将 XML 文档中的元素和属性映射至对应于 DB2 数据的元素和属性节点。 这些节点提供从 XML 数据至 DB2 数据的路径。
10. **要添加根节点**:
    a. 选择**根**图标。
    b. 单击**新元素**以定义新节点。
    c. 在**详细信息**框中，将**节点类型**指定为**元素**。
    d. 在**节点名**字段中输入顶级节点的名称。
    e. 单击**添加**以创建新节点。

       您已经创建了根节点或元素，它是映象中所有其他元素或属性节点的父代。 根节点具有表子元素和连接条件。
    f. 对作为集合一部分的每个表添加表节点。
       1) 突出显示根节点名并选择**新元素**。
       2) 在**详细信息**框中，将**节点类型**指定为**表**。
       3) 从**表名**中选择表的名称。该表必须已存在。
       4) 单击**添加**以添加表节点。
       5) 对每个表重复这些步骤。
    g. 为表节点添加连接关系。
       1) 突出显示根节点名并选择**新元素**。

2) 在**详细信息**框中，将**节点类型**指定为**条件**。

3) 在**条件**字段中，使用下列语法来输入连接条件:

```
table_name.table_column = table_name.table_column AND
table_name.table_column = table_name.table_column ...
```

4) 单击**添加**以添加该条件。

11. **要添加元素或属性节点**:

a. 单击左边字段中的父代节点，以添加子元素或属性。

b. 单击**新元素**。如果未选择父代节点，则不可选择**新元素**。

c. 从**详细信息**框中的**节点类型**菜单选择节点类型

节点类型菜单仅显示在映射中的该点处有效的节点类型 **元素**或**属性**。

d. 在**节点名**字段中指定节点名。

e. 单击**添加**以添加新节点。

f. **要将元素或属性节点的内容映射至关系表**:

1) 指定文本节点。

a) 单击父代节点。

b) 单击**新元素**。

c) 在**节点类型**字段中，选择**文本**。

d) 选择**添加**以添加该节点。

2) 添加表节点。

a) 选择刚创建的文本节点并单击**新元素**。

b) 在**节点类型**字段中，选择**表**并对该元素指定表名。

c) 单击**添加**以添加该节点。

3) 添加列节点。

a) 再次选择该文本节点并单击**新元素**。

b) 在**节点类型**字段中，选择**列**并对该元素指定列名。

c) 单击**添加**以添加该节点。

限制: 不能使用管理向导来创建新列。 如果将 “列” 指定为节点类型，则只可选择 DB2 数据库中已存在的列。

4) 可选择对列添加条件。

a) 再次选择该文本节点并单击**新元素**。

b) 在**节点类型**字段中，选择**条件**且以下列语法选择条件:

operator LIKE|<|>|= *value*

c) 单击**添加**以添加该节点。

g. 继续编辑 RDB 映射，或单击**下一步**以打开“指定 DAD”窗口。

12. **要除去节点**:
    a. 单击左边的字段中的某个节点。
    b. 单击**除去**。
    c. 继续编辑 RDB_node 映射，或单击**下一步**以打开“指定 DAD”窗口。
13. 在“指定 DAD”窗口的**文件名**字段中，输入修改过的 DAD 的输出文件名。
14. 单击**完成**以除去该节点并返回至 LaunchPad 窗口。

从 DB2 命令外壳: DAD 文件是可使用任何文本编辑器创建的 XML 文件。下列步骤显示样本附录第240页的『文档访问定义文件』中的分段。请参考这些样本以获取更多综合信息和上下文。

1. 打开文本编辑器。
2. 创建 DAD 头:

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "path\dad.dtd"> --> the path and file name of the DTD
       for the DAD
```

3. 插入 <DAD></DAD> 标记。
4. 在 <DAD> 标记后面，指定将 DAD 文件与 XML 文档 DTD 相关联的 DTD ID。

```
<dtdid>path\dtd_name.dtd> --> the path and file name of the DTD
       for your application
```

5. 指定是否进行验证（即，使用 DTD 以确保 XML 文档是有效的 XML 文档）。例如:

```
<validation>NO</validation>   --> 指定 YES 或 NO
```

6. 使用 <Xcollection> 元素将访问方法和存储方法定义为 XML 集合。访问方法和存储方法定义 XML 数据存储在 DB2 表的集合中。

```
<Xcollection>
</Xcollection>
```

7. 添加以下 prolog 信息:

```
<prolog>?xml version="1.0"?</prolog>
```

   需要与以上文本完全一致。

8. 添加 <doctype></doctype> 标记。例如:

```
<doctype>! DOCTYPE Order SYSTEM "c:\dxx\samples\dtd\getstart.dtd"</doctype>
```

9. 使用 <root_node> 定义根节点。在 root_node 内部，指定构成 XML 文档的元素和属性。

10. 将 XML 文档中的元素和属性映射至对应于 DB2 数据的元素和属性节点。 这些节点提供从 XML 数据至 DB2 数据的路径。

    a. 定义根节点元素节点。 此元素节点包含:

       - RDB_node，它指定表节点（且以一个连接条件指定集合）
       - 子元素
       - 属性

       要指定节点和条件:

       1) 创建 RDB_node 元素: 例如:

          ```
          <RDB_node>
          </RDB_node>
          ```

       2) 对包含要包括在 XML 文档中的数据的每个表定义 <table_node>。例如，如果有三个表 ORDER_TAB、PART_TAB 和 SHIP_TAB 包含将置于文档中的列数据，则为每个表创建一个表节点。例如:

          ```
          <RDB_node>
          <table name="ORDER_TAB">
          <table name="PART_TAB">
          <table name="SHIP_TAB">
          ```
          ```
          </RDB_node>
          ```

       3) 可选择在您计划启用此集合时，对每个表指定一个键列。键属性不是进行组合所必需的，但当您启用集合时，所使用的 DAD 文件必需既支持组合又支持分解。例如:

          ```
          <RDB_node>
          <table name="ORDER_TAB" key="order_key">
          <table name="PART_TAB" key="part_key">
          <table name="SHIP_TAB" key="date mode">
          </RDB_node>
          ```

       4) 为集合中的表定义连接条件。语法为

          ```
          expression = expression AND
          expression = expression
          ```

          例如:

          ```
          <RDB_node>
          <table name="ORDER_TAB">
          <table name="PART_TAB">
          <table name="SHIP_TAB">
          <condition>
             order_tab.order_key = part_tab.order_key AND
             part_tab.part_key = ship_tab.part_key
          </condition>
          </RDB_node>
          ```

    b. 对 XML 文档中映射至 DB2 表中的列的每个元素定义 <element_node> 标记。例如:

```
<element_node name="name">
</element_node>
```

元素节点的类型可为下列其中一种元素类型:

- <text_node>: 用于指定元素包含 DB2 表的内容; 该元素没有子元素
- <attribute_node>: 用于指定属性。属性节点在下一步骤中定义。

该文本节点包含要将内容映射至 DB2 表和列名的 <RDB_node>。

RDB_node 用于具有要映射至 DB2 表的内容的底级元素。 RDB_node 具有下列子元素。

- <table>: 定义对应于元素的表
- <column>: 定义包含对应元素的列并指定带有类型属性的列类型
- <condition>: 可选择指定列的条件

例如, 您可能具有一个 XML 元素 <Tax>, 它映射至一个称为 TAX 的列:

XML 文档:

```
<Tax>0.02</Tax>
```

在这种情况下, 您希望值 0.02 成为 TAX 列中的一个值。

```
<element_node name="Tax">
   <text_node>
      <RDB_node>
         <table name="part_tab"/>
         <column name="tax"/>
      </RDB_node>
   </text_node>
</element_node>
```

在本例中, <RDB_node> 指定 <Tax> 元素的值是一个文本值, 数据存储在 TAX 列中的 PART_TAB 表中。参见第240页的『文档访问定义文件』中的示例 DAD 文件以了解 RDB_node 映射, 并参见第233页的『附录A. 用于 DAD 文件的 DTD』(它提供了 DAD 文件的全部语法)中 DAD 文件的 DTD。

c. 可选择在您计划启用此集合时, 对每个 <column> 元素添加一个类型属性。类型属性通常不是进行组合所必需的, 但当您启用集合时, 所使用的 DAD 文件必需既支持组合又支持分解。例如:

```
<column name="tax" type="real"/>
```

d. 对 XML 文档中的每个属性定义 <attribute_node>, 该文档中的每个属性都映射至 DB2 表中的一列。例如:

```
<attribute_node name="key">
</attribute_node>
```

该属性节点具有要将属性值映射至 DB2 表和列的 <RDB_node>。<RDB_node> 具有下列子元素。

- <table>: 定义对应于元素的表
- <column>: 定义包含对应元素的列
- <condition>: 可选择指定列的条件

例如，您可能想要将属性键用于元素 <Order>。键值要存储在列 PART_KEY 中。在 DAD 文件中，创建键的 <attribute_node>，并指示要将值存储至其中的表。

DAD 文件

```
<attribute_node name="key">
      <RDB_node>
    <table name="part_tab">
    <column name="part_key"/>
      <RDB_node>
</attribute_node>
```

组合的 XML 文档:

```
<Order key="1">
```

11. 确保在最后一个 </element_node> 标记后有一个表示结束的 </root_node> 标记。
12. 确保在 </root_node> 标记后有一个表示结束的 </Xcollection> 标记。
13. 确保在 </Xcollection> 标记后有一个表示结束的 </DAD> 标记。 tag.

指定 XML 文档的样式表

当组合文档时，XML Extender 还使用 <stylesheet> 元素支持样式表的处理指令。处理指令必须在 <Xcollection> 根元素中，并通过为 XML 文档结构定义 <doctype> 和 <prolog> 来定位它。例如:

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:\dtd\dad.dtd">
<DAD>
<SQL_stmt>
       ...
</SQL_stmt>
<Xcollection>
...
<prolog>...</prolog>
<doctype>...</doctype>
<stylesheet>?xml-stylesheet type="text/css" href="order.css"?</stylesheet>
<root_node>...</root_node>
```

```
...

</Xcollection>...
</DAD>
```

使用 RDB_node 映射分解 XML 文档

使用 RDB_node 映射来分解 XML 文档。此方法使用 <RDB_node> 来对元素或属性节点指定 DB2 表、列和条件。 <RDB_node> 使用下列元素:

- <table>: 定义对应于元素的表
- <column>: 定义包含对应元素的列
- <condition>: 可选择指定列的条件

在 <RDB_node> 中使用的子元素取决于该节点的内容，并使用下列规则:

如果节点类型是:	使用 RDB 子元素:		
	表	列	条件 [1]
根元素	是	否	是
属性	是	是	可选
文本	是	是	可选

(1) 需要有多个表

使用管理向导: **要创建用于分解的 DAD**:

1. 设置并启动管理向导。参见第61页的『启动管理向导』以获取详情。
2. 从 LaunchPad 窗口单击**使用 DAD 文件**。显示“指定 DAD ”窗口。
3. 选择是编辑现存 DAD 文件还是创建新 DAD。

   要编辑现存的 DAD:

   a. 在**文件名**字段中输入 DAD 文件名，或单击 **...** 以浏览现存的 DAD
   b. 验证向导是否识别指定的 DAD 文件。
      - 如果向导识别指定的 DAD 文件，则**下一步**是可选择的，且 XML 集合 RDB 节点映射会显示在**类型**字段中。
      - 如果向导不识别指定的 DAD 文件，则**下一步**是不可选择的。在**文件名**字段中重新输入 DAD 文件名，或单击 **...** 以再次浏览现存的 DAD 文件 继续这些步骤，直到**下一步**是可选的为止。
   c. 单击**下一步**以打开“选择验证”窗口。

   要创建新 DAD:

   a. 将**文件名**字段留为空白。
   b. 从**类型**菜单选择 XML 集合 RDB_node 映射。

c. 单击**下一步**以打开“选择验证”窗口。

4. 在“选择验证”窗口中，选择是否使用 DTD 来验证 XML 文档。
   - 要进行验证:
     a. 单击**用 DTD 来验证 XML 文档**。
     b. 从 **DTD ID** 菜单选择要用于验证的 DTD。

     如果还未将任何 DTD 导入数据库的 DTD 库，则不能验证 XML 文档。
   - 单击**不要使用 DTD 来验证 XML 文档**继续进行，而不验证 XML 文档。
5. 单击**下一步**以打开“指定文本”窗口。
6. 如果只是在分解 XML 文档，则忽略 **Prolog** 字段。如果将 DAD 文件同时用于组合和分解，则在“指定文本”窗口的 **Prolog** 字段中输入 prolog 名。如果正将 XML 文档分解为 DB2 数据，则不需要 prolog。

   ```
   <?xml version="1.0"?>
   ```

   如果正在编辑现存的 DAD，则 prolog 会自动显示在 **Prolog** 字段中。
7. 如果仅是分解 XML 文档，则忽略 **Doctype** 字段。如果将 DAD 文件同时用于组合和分解，则在 **Doctype** 字段中输入 XML 文档的文档类型

   如果正在编辑现存的 DAD，则文档类型会自动显示在 **Doctype** 字段中。
8. 单击**下一步**以打开“RDB 映射”窗口。
9. 通过在“RDB 映射”窗口左边的字段中单击元素或属性节点来选择要自其进行映射的元素或属性节点。

   将 XML 文档中的元素和属性映射至对应于 DB2 数据的元素和属性节点。 这些节点提供从 XML 数据至 DB2 数据的路径。
10. **要添加根节点**:
    a. 选择**根**图标。
    b. 单击**新元素**以定义新节点。
    c. 在**详细信息**框中，将**节点类型**指定为**元素**。
    d. 在**节点名**字段中输入顶级节点的名称。
    e. 单击**添加**以创建新节点。

       您已经创建了根节点或元素，它是映象中所有其他元素或属性节点的父代。 根节点具有表子元素和连接条件。
    f. 对作为集合一部分的每个表添加表节点。
       1) 突出显示根节点名并选择**新元素**。
       2) 在**详细信息**框中，将**节点类型**指定为**表**。
       3) 从**表名**中选择表的名称。该表必须已存在。

4) 对**表键**字段中的表指定键列。

5) 单击**添加**以添加表节点。

6) 对每个表重复这些步骤。

g. 为表节点添加连接关系。

1) 突出显示根节点名并选择**新元素**。

2) 在**详细信息**框中，将**节点类型**指定为**条件**。

3) 在**条件**字段中，使用下列语法来输入连接条件:

```
table_name.table_column = table_name.table_column AND
table_name.table_column = table_name.table_column ...
```

4) 单击**添加**以添加该条件。

现在可将子元素和属性添加至此节点。

11. **要添加元素或属性节点**:

a. 单击左边字段中的父代节点，以添加子元素或属性。

如果未选择父节点，则**新建**是不可选择的。

b. 单击**新元素**。

c. 从**详细信息**框中的**节点类型**菜单选择节点类型

节点类型菜单仅显示在映射中的该点处有效的节点类型 **元素**或**属性**。

d. 在**节点名**字段中指定节点名。

e. 单击**添加**以添加新节点。

f. **要将元素或属性节点的内容映射至关系表**:

1) 指定文本节点。

a) 单击父代节点。

b) 单击**新元素**。

c) 在**节点类型**字段中，选择**文本**。

d) 选择**添加**以添加该节点。

2) 添加表节点。

a) 选择刚创建的文本节点并单击**新元素**。

b) 在**节点类型**字段中，选择**表**并对该元素指定表名。

c) 单击**添加**以添加该节点。

3) 添加列节点。

a) 再次选择该文本节点并单击**新元素**。

b) 在**节点类型**字段中，选择**列**并对该元素指定列名。

c) 对**类型**字段中的列指定基本数据类型，以指定该列必须为哪种类型才能存储未标记的数据。

d) 单击**添加**以添加该节点。

限制: 不能使用管理向导来创建新列。 如果将 “列” 指定为节点类型，则只可选择 DB2 数据库中已存在的列。

4) 可选择对列添加条件。

a) 再次选择该文本节点并单击**新元素**。

b) 在**节点类型**字段中，选择**条件**且以下列语法选择条件:

operator LIKE|<|>|= *value*

c) 单击**添加**以添加该节点。

可通过选择节点，更改**详细信息**框中的字段，并单击**更改**来修改这些节点。

g. 继续编辑 RDB 映射，或单击**下一步**以打开 “指定 DAD” 窗口。

12. **要除去节点**:

a. 单击左边的字段中的某个节点。

b. 单击**除去**。

c. 继续编辑 RDB_node 映射，或单击**下一步** 以打开 “指定 DAD” 窗口。

13. 在 “指定 DAD” 窗口的**文件名**字段中，输入修改过的 DAD 的输出文件名。

14. 单击**完成**以除去该节点并返回至 LaunchPad 窗口。

从 DB2 命令外壳: DAD 文件是可使用任何文本编辑器创建的 XML 文件。下列步骤显示样本附录第240页的『文档访问定义文件』中的分段。请参考这些样本以获取更多综合信息和上下文。

1. 打开文本编辑器。
2. 创建 DAD 头:

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "path\dad.dtd"> --> the path and file name of the DTD
       for the DAD
```

3. 插入 <DAD></DAD> 标记。
4. 在 <DAD> 标记后面，指定将 DAD 文件与 XML 文档 DTD 相关联的 DTD ID。

```
<dtdid>path\dtd_name.dtd> --> the path and file name of the DTD
       for your application
```

5. 指定是否进行验证（即，使用 DTD 以确保 XML 文档是有效的 XML 文档）。例如:

```
<validation>NO</validation>   --> 指定 YES 或 NO
```

6. 使用 <Xcollection> 元素将访问方法和存储方法定义为 XML 集合。访问方法和存储方法定义 XML 数据存储在 DB2 表的集合中。

```
<Xcollection>
</Xcollection>
```

7. 添加以下 prolog 信息:

```
<prolog>?xml version="1.0"?</prolog>
```

需要与以上文本完全一致。

8. 添加 <doctype></doctype> 标记。例如:

```
<doctype>! DOCTYPE Order SYSTEM "c:\dxx\samples\dtd\getstart.dtd"</doctype>
```

9. 使用 <root_node></root_node> 标记来定义 root_node。在 root_node 内部，指定构成 XML 文档的元素和属性。
10. 在 <root_node> 标记之后，将 XML 文档中的元素和属性映射至对应于 DB2 数据的元素和属性节点。 这些节点提供从 XML 数据至 DB2 数据的路径。
    a. 定义顶级根元素节点。此元素节点包含:
       - 表节点，且有一连接条件指定集合。
       - 子元素
       - 属性

       要指定节点和条件:

       1) 创建 RDB_node 元素: 例如:

```
<RDB_node>
</RDB_node>
```

       2) 对包含要包括在 XML 文档中的数据的每个表定义 <table_node>。例如，如果有三个表 ORDER_TAB、PART_TAB 和 SHIP_TAB 包含将置于文档中的列数据，则为每个表创建一个表节点。例如:

```
<RDB_node>
<table name="ORDER_TAB">
<table name="PART_TAB">
<table name="SHIP_TAB"></RDB_node>
```

       3) 为集合中的表定义连接条件。语法为

```
expression = expression AND
expression = expression ...
```

       例如:

```
<RDB_node>
<table name="ORDER_TAB">
<table name="PART_TAB">
<table name="SHIP_TAB">
<condition>
```

```
   order_tab.order_key = part_tab.order_key AND
   part_tab.part_key = ship_tab.part_key
</condition>
</RDB_node>
```

4) 对每个表指定一个主键。主键由一列或多列组成，称为组合键。要指定主键，应将属性键添加至 RDB_node 的表元素。以下示例对根 element_node Order 的 RDB_node 中的每一个表定义主键:

```
<element_node name="Order">
    <RDB_node>
         <table name="order_tab" key="order_key"/>
         <table name="part_tab" key="part_key price"/>
         <table name="ship_tab" key="date mode"/>
         <condition>
             order_tab.order_key = part_tab.order_key AND
             part_tab.part_key = ship_tab.part_key
         </condition>
    <RDB_node>
```

当组合 XML 文档时忽略对分解指定的信息。

键属性是进行分解以及启用集合时所必需的，因为所使用的 DAD 文件必需既支持组合又支持分解。

b. 对 XML 文档中映射至 DB2 表中的列的每个元素定义 <element_node> 标记。例如:

```
<element_node name="name">
</element_node>
```

元素节点的类型可为下列其中一种元素类型:

- <text_node>: 用于指定元素包含 DB2 表的内容: 在此情况下，该元素没有子元素。
- <attribute_node>: 用于指定属性; 属性节点在下一步骤中定义
- 子元素

该文本节点包含要将内容映射至 DB2 表和列名的 RDB_node。

RDB_node 用于具有要映射至 DB2 表的内容的底级元素。 RDB_node 具有下列子元素。

- <table>: 定义对应于元素的表
- <column>: 定义包含对应元素的列
- <condition>: 可选择指定列的条件

例如，您可能具有这样的 XML 元素 <Tax>，您想要将其未作标记的内容存储在称为 TAX 的列中:

XML 文档:

```
<Tax>0.02</Tax>
```

在这种情况下，要将值 0.02 存储在列 TAX 中。

在 DAD 文件中，指定 <RDB_node> 以将 XML 元素映射至 DB2 表和列。

DAD 文件:

```
<element_node name="Tax">
   <text_node>
      <RDB_node>
         <table name="part_tab"/>
         <column name="tax"/>
      </RDB_node>
   </text_node>
</element_node>
```

<RDB_node> 指定 <Tax> 元素的值为文本值，而数据存储在 TAX 列中的 PART_TAB 表中。

c. 对 XML 文档中的每个属性定义 <attribute_node>，该文档中的每个属性都映射至 DB2 表中的一列。例如:

```
<attribute_node name="key">
</attribute_node>
```

该属性节点具有要将属性值映射至 DB2 表和列的 RDB_node。 RDB_node 具有下列子元素。

- <table>: 定义对应于元素的表
- <column>: 定义包含对应元素的列
- <condition>: 可选择指定列的条件

例如，您可能具有元素 <Order> 的属性键。键值要存储在列 PART_KEY 中。

XML 文档:

```
<Order key="1">
```

在 DAD 文件中，创建键的属性节点并指示值 1 要存储在其中的表。

DAD 文件:

```
<attribute_node name="key">
  <RDB_node>
    <table name="part_tab">
    <column name="part_key"/>
  <RDB_node>
</attribute_node>
```

11. 对每一个 attribute_node 和 text_node 的 RDB_node 指定列类型。这确保了将存储未作标记的数据的每一列都有正确的数据类型。要指定列类型，应将属性类型添加至列元素。以下示例将列类型定义为 INTEGER:

```
<attribute_node name="key">
    <RDB_node>
        <table name="order_tab"/>
        <column name="order_key" type="integer"/>
    </RDB_node>
</attribute_node>
```

12. 确保在最后一个 </element_node> 标记后有一个表示结束的 </root_node> 标记。
13. 确保在 </root_node> 标记后有一个表示结束的 </Xcollection> 标记。
14. 确保在 </Xcollection> 标记后有一个表示结束的 </DAD> 标记。

启用 XML 集合

启用 XML 集合对 DAD 文件进行了语法分析，以标识与 XML 文档相关的表和列，并记录了 XML_USAGE 表中的控制信息。对于下列操作，启用 XML 集合是可选的:

- 分解 XML 文档并将数据存储在新 DB2 表中
- 将多个 DB2 表中的现存数据组合成一个 XML 文档

如果将同一 DAD 文件用于组合和分解，则可对组合和分解都启用集合。

可通过 XML Extender 管理向导，将 **dxxadm** 命令与 enable_collection 选项配合使用，或可使用 XML Extender 存储过程 dxxEnableCollection() 来启用 XML 集合。

使用管理向导

使用以下步骤来启用 XML 集合。

1. 设置并启动管理向导。参见第61页的『启动管理向导』以获取详情。
2. 从 LaunchPad 窗口单击**使用 XML 集合**。显示“选择任务”窗口。
3. 单击**启用集合**并单击**下一步**。显示“启用集合”窗口。
4. 在下拉菜单的**集合名**字段中选择想要启用的集合的名称。
5. 在 **DAD 文件名**字段中输入 DAD 文件名，或单击 **...** 以浏览现存的 DAD 文件。
6. 可选择在**表空间**字段中输入先前创建的表空间的名称。

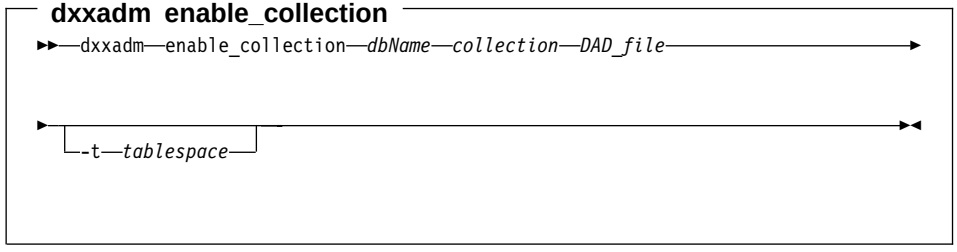
   该表空间将包含为进行分解而生成的新 DB2 表。
7. 单击**完成**以启用该集合并返回至 LaunchPad 窗口。
   - 如果成功启用了该集合，消息启用集合成功会显示。

- 如果未成功启用该集合，则显示一条错误消息。继续以上步骤，直到成功启用该集合为止。

从 DB2 命令外壳

要启用 XML 集合，输入 **dxxadm** 命令:

语法:



参数:

dbName
数据库名称。

collection
XML 集合的名称。此值用作 XML 集合存储过程的参数。

DAD_file
包含文档访问定义 (DAD) 的文件的名称。

tablespace
现存的表空间，它包含为进行分解而生成的新 DB2 表。 如果未指定，则使用缺省表空间。

示例: 以下示例使用 DB2 命令外壳启用数据库 SALES_DB 中称为 sale_ord 的集合。 DAD 文件使用 SQL 映射，且可在第241页的『DAD 文件: XML 集合 - SQL 映射』中找到。

```
dxxadm enable_collection SALES_DB sales_ord getstart.dad
```

在启用 XML 集合之后，可使用 XML Extender 存储过程来组合或分解 XML 文档。

禁用 XML 集合

禁用 XML 集合将除去 XML_USAGE 表中将表和列标识为集合一部分的记录。这不会删除任何数据表。在想要更新 DAD 且需要重新启用集合或要删除集合时，应禁用集合。

可通过 XML Extender 管理向导，将 **dxxadm** 命令与 disable_collection 选项配合使用，或使用 XML Extender 存储过程 dxxDisableCollection() 来禁用 XML 集合。

使用管理向导

使用以下步骤来禁用 XML 集合。

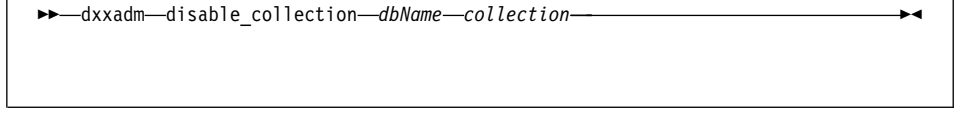
1. 设置并启动管理向导。参见第61页的『启动管理向导』以获取详情。
2. 从 LaunchPad 窗口单击**使用 XML 集合**，以查看与 XML Extender 集合相关的任务。显示“选择任务”窗口。
3. 单击**禁用 XML 集合**，然后单击**下一步**以禁用 XML 集合 显示“禁用集合”窗口。
4. 在**集合名**字段中输入想要禁用的集合的名称。
5. 单击**完成**以禁用该集合并返回至 LaunchPad 窗口。
   - 如果成功禁用了该集合，消息“禁用集合成功”会显示。
   - 如果未成功禁用该集合，会显示错误消息框。继续以上步骤，直到成功禁用该集合为止。

从 DB2 命令外壳

要禁用 XML 集合，输入 **dxxadm** 命令:

语法:

dxxadm disable_collection



参数:

dbName

数据库名称。

collection

XML 集合的名称。此值用作 XML 集合存储过程的参数。

示例:

```
dxxadm disable_collection SALES_DB sales_ord
```

对 XML 禁用数据库

在想要清除 XML Extender 环境并删除 XML Extender UDT、UDF、存储过程及管理支持表时，禁用数据库。XML Extender 禁用您与其相连的数据库（使用当前实例）。

对 XML 禁用数据库时，XML Extender 对该数据库进行下列操作:

- 删除所有用户定义类型 (UDT) 和用户定义函数 (UDF)
- 删除带有 XML Extender 的元数据的控制表
- 删除 db2xml 模式。

开始之前

禁用要禁用的数据库中的所有 XML 列或集合。

使用管理向导

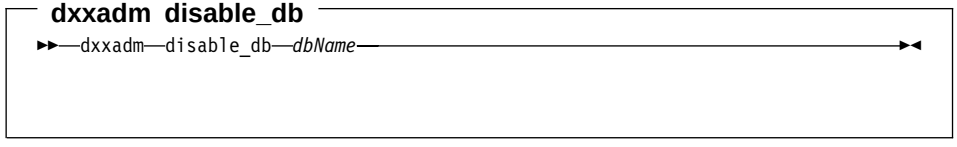
使用下列步骤来对 XML 数据禁用数据库:

1. 设置并启动管理向导。参见第61页的『启动管理向导』以获取详情。
2. 从 LaunchPad 窗口单击**禁用数据库**，以禁用当前数据库。
   如果当前未启用数据库，则仅**启用数据库**是可选择的。
   数据库被禁用时，您会返回至 LaunchPad 窗口。

从 DB2 命令外壳

从命令行输入 **dxxadm**，指定将要禁用的数据库。

语法:



参数:

dbName
要禁用的数据库的名称。

示例: 禁用称为 SALES_DB 的现存数据库。

```
dxxadm disable_db SALES_DB
```

第3部分 程序设计

此部分描述了用于管理 XML 数据的程序设计技术。

© Copyright IBM Corp. 1999, 2000

第5章 管理 XML 列数据

当使用 XML 列时，将整个 XML 文档作为列数据来存储。此访问和存储方法允许您完整地保存 XML 文档，同时使您能够对文档进行索引和搜索，从文档中检索数据，以及更新文档。 XML 列在 DB2 中包含了本机格式的 XML 文档作为列数据。在为 XML 启用了数据库之后，下列用户定义类型 (UDT) 可供您使用:

XMLCLOB
在 DB2 中作为字符大对象 (CLOB) 来存储的 XML 文档内容

XMLVARCHAR
在 DB2 中作为 VARCHAR 来存储的 XML 文档内容

XMLFile
存储在本地文件系统的文件中的 XML 文档

可以创建或改变将 XML UDT 用作列数据类型的应用程序表 这些表被称为 XML 表。 要了解如何为 XML 创建或改变表，参见第71页的『创建或改变 XML 表』。

为 XML 启用一列之后，就可以开始管理 XML 列的内容。 创建 XML 列之后，可以执行下列管理任务:

- 在 DB2 中存储 XML 文档
- 从 DB2 中检索 XML 数据或文档
- 更新 XML 文档
- 删除 XML 数据或文档

要执行这些任务，可以使用两种方法:

- *缺省强制转型函数*，它将 SQL 基本类型转换为 XML UDT
- XML Extender 提供的用户定义函数 (UDF)

本书对每种任务都描述了这两种方法。

UDT 和 UDF 名

DB2 函数的全名是: *schema-name.function-name*，其中，*schema-name* 是一个标识符，它提供了对 SQL 对象的逻辑分组。 XML Extender UDF 的模式名是 db2xml。 db2xml 模式名也是 XML Extender UDT 的限定符。在本书中，仅对函数名进行引用。

© Copyright IBM Corp. 1999, 2000

函数路径是模式名的有序列表。DB2 使用列表中的模式名的次序来解析对函数和 UDT 的引用。可通过指定 SQL 语句 SET CURRENT FUNCTION PATH 来指定函数路径。这将在 CURRENT FUNCTION PATH 专用寄存器中设置函数路径。

对于 XML Extender，最好将 db2xml 模式添加到函数路径中。因此使您可在输入 XML Extender UDF 和 UDT 名称时，不必添加前缀 db2xml。下列示例显示如何将 db2xml 模式添加到函数路径中:

```
SET CURRENT FUNCTION PATH = db2xml, CURRENT FUNCTION PATH
```

重要事项: 若您作为 db2xml 来登录，则不要将 db2xml 作为函数路径中的第一个模式来添加；当您作为 db2xml 来登录时，db2xml 将被自动设置为第一个模式。这将产生一个错误条件，因为函数路径将以两个 db2xml 模式开头。

存储数据

通过使用 XML Extender，可以将完整的 XML 文档插入 XML 列中。若定义辅助表，则 XML Extender 将自动更新这些表。当直接存储 XML 文档时，XML Extender 将基本类型作为 XML 类型来存储。

任务概述:

1. 确保已经创建或更新了 DAD 文件。
2. 确定当存储文档时要使用哪种数据类型。
3. 选择一种用来存储 DB2 表中的数据的方法（强制转型函数或 UDF）。
4. 指定 SQL INSERT 语句，该语句指定要包含 XML 文档的 XML 表和列。

XML Extender 提供了用来存储 XML 文档的两种方法: 缺省强制转型函数和存储器 UDF。表8显示何时使用每种方法。

表 8. XML Extender 存储器函数

基本类型	在 DB2 中存储为...		
	XMLVARCHAR	**XMLCLOB**	**XMLFILE**
VARCHAR	XMLVARCHAR()	N/A	XMLFileFromVarchar()
CLOB	N/A	XMLCLOB()	XMLFileFromCLOB()
FILE	XMLVarcharFromFile()	XMLCLOBFromFile()	XMLFILE

使用缺省强制转型函数

对于每个 UDT，都存在*缺省强制转型函数*，用于将 SQL 基本类型强制转型为 UDT。可以在 VALUES 子句中使用 XML Extender 提供的强制转型函数来插入数据。第107页的表9显示了提供的强制转型函数:

表 9. XML Extender 缺省强制转型函数

在 SELECT 子句中使用的强制转型	返回类型	描述
XMLVARCHAR(VARCHAR)	XMLVARCHAR	从内存缓冲区输入 VARCHAR。
XMLCLOB(CLOB)	XMLCLOB	从内存缓冲区输入 CLOB 或 CLOB 定位器
XMLFILE(VARCHAR)	XMLFILE	仅存储文件名

示例: 以下语句将强制转型的 VARCHAR 类型插入 XMLVARCHAR 类型:

```
INSERT INTO sales_tab
VALUES('123456', 'Sriram Srinivasan', db2xml.XMLVarchar(:xml_buff))
```

使用存储器 UDF:

对于每个 XML Extender UDT, 都存在一个存储器 UDF, 用来将数据从不同于其基本类型的资源中导入 DB2 中。例如, 若您想将 XML 文件文档作为 XMLCLOB 来导入 DB2 中, 则您可以使用函数 XMLCLOBFromFile()。

表10显示由 XML Extender 提供的存储器函数。

表 10. XML Extender 存储器 UDF

存储器用户定义的函数	返回类型	描述
XMLVarcharFromFile()	XMLVARCHAR	从服务器上的文件中读取 XML 文档, 并返回 XMLVARCHAR 类型的值。
XMLCLOBFromFile()	XMLCLOB	从服务器上的文件中读取 XML 文档, 并返回 XMLCLOB 类型的值。
XMLFileFromVarchar()	XMLFILE	从内存中读取 VARCHAR 类型的 XML 文档, 将它写至外部文件, 并返回 XMLFILE 类型的值, 它是文件名。
XMLFileFromCLOB()	XMLFILE	从内存中读取 CLOB 或 CLOB 定位器类型的 XML 文档, 将它写至外部文件, 并返回 XMLFILE 类型的值, 它是文件名。

示例: 下列语句通过使用 XMLCLOBFromFile() 函数作为 XMLCLOB，将记录存储到 XML 表中。

```
EXEC SQL INSERT INTO sales_tab(ID, NAME, ORDER)
    VALUES( '1234', 'Sriram Srinivasan,
      XMLCLOBFromFile('c:\dxx\samples\cmd\getstart.xml'))
```

上述示例将 XML 对象从文件 `c:\dxx\samples\cmd\getstart.xml` 导入至表 SALES_TAB 中的列 ORDER 中。

检索数据

通过使用 XML Extender，可以检索整个文档，也可以检索元素和属性的内容。 当直接检索 XML 列时，XML Extender 将 UDT 作为列类型返回。 有关检索数据的详情，参见下列各节:

- 『检索整个文档』
- 第110页的『检索元素内容和属性值』

XML Extender 提供了用来检索数据的两种方法: 缺省强制转型函数和 Content() 过载 UDF。表11显示了何时使用每种方法。

表 11. XML Extender 检索函数

XML 类型	从 DB2 中检索...		
	VARCHAR	CLOB	FILE
XMLVARCHAR	VARCHAR	N/A	Content()
XMLCLOB	N/A	XMLCLOB	Content()
XMLFILE	N/A	Content()	FILE

检索整个文档

任务概述:

1. 确保您已经在 XML 表中存储了 XML 文档，并确定您想检索哪些数据。
2. 选择一种用来检索 DB2 表中的数据的方法（强制转型函数或 UDF）。
3. 若使用过载的 Content() UDF 来确定哪种数据类型是与正在检索的数据相关联的，以及要导出哪种数据类型。
4. 指定 SQL 查询，该查询指定要从哪些 XML 表和列来检索 XML 文档。

XML Extender 提供了检索数据的两种方法:

使用缺省强制转型函数

使用 DB2 为 UDT 提供的缺省强制转型函数，将 XML UDT 转换为 SQL

基本类型，然后对它进行操作。可以在 SELECT 语句中使用 XML Extender 提供的强制转型函数来检索数据。 表12显示了提供的强制转型函数:

表 12. XML Extender 缺省强制转型函数

在 select 子句中使用强制转型	返回类型	描述
varchar(XMLVARCHAR)	VARCHAR	采用 VARCHAR 的 XML 文档
clob(XMLCLOB)	CLOB	采用 CLOB 的 XML 文档
varchar(XMLFile)	VARCHAR	采用 VARCHAR 的 XML 文件名

示例: 下列示例将检索 XMLVARCHAR，并将它作为 VARCHAR 数据类型存储在内存中:

```
EXEC SQL SELECT db2xml.varchar(order) from sales_tab
```

使用 Content() 过载 UDF

使用 Content() UDF 将文档内容从外部存储器检索到内存中，或者将文档从内部存储器导出到 DB2 服务器上的*外部文件*中。

例如，您可能具有存储为 XMLFILE 的 XML 文档 且您想要在内存中处理它，则可使用 Content() UDF，该函数可将 XMLFILE 数据类型用作输入，并返回 CLOB。

根据指定的数据类型不同，Content() UDF 将执行两种不同的检索函数。它:

从外部存储器检索文档，并将它放入内存中

当某个 XML 文档存储为外部文件时，可以使用 Content()，将该 XML 文档检索到内存缓冲区或 CLOB 定位器中. 使用以下函数语法，其中，*xmlobj* 是要查询的 XML 列:

XMLFILE 到 CLOB: 从文件中检索数据，并将它导出到 CLOB 定位器中。

```
Content(xmlobj XMLFile)
```

从内部存储器中检索文档，并将该文档导出到外部文件中

还可使用 Content() 来检索作为 XMLCLOB 数据类型存储在 DB2 中的 XML 文档，并将其导出至数据库服务器文件系统上的文件。它返回类型为 VARCHAR 的文件的名称。使用以下函数语法，其中，*xmlobj* 是要查询的 XML 列，而 *filename* 是外部文件。 *XML 类型*可以是 XMLVARCHAR 或 XMLCLOB数据类型。

*XML 类型*到外部文件: 检索作为 XML 数据类型存储的 XML 内容，并将它导出到外部文件中。

```
Content(xmlobj XML type, filename varchar(512))
```

其中:

xmlobj 是要检索 XML 内容的 XML 列的名称; *xmlobj* 可以是类型 XMLVARCHAR 或 XMLCLOB。

filename
是要用来存储 XML 数据的文件名。

在下面的示例中，带有*嵌入式 SQL* 的一个小型 C 语言程序段说明了 XML 文档是如何从文件中检索到内存中的。此示例假定 BOOK 列是 XMLFILE 类型。

```
EXEC SQL BEGIN DECLARE SECTION;
             SQL TYPE IS CLOB_LOCATOR xml_buff;
          EXEC SQL END DECLARE SECTION;
          EXEC SQL CONNECT TO SALES_DB
          EXEC SQL DECLARE c1 CURSOR FOR
              SELECT Content(order) from sales_tab
                        EXEC SQL OPEN c1;
          do {
             EXEC SQL FETCH c1 INTO :xml_buff;
             if (SQLCODE != 0) {
                break;
                }
             else {
               /* do whatever you need to do with the XML doc in buffer */
               }
             }
          EXEC SQL CLOSE c1;
          EXEC SQL CONNECT RESET;
```

检索元素内容和属性值

可以从一个或多个 XML 文档来检索（抽取）*元素*的内容或*属性*值（单个文档搜索或集合文档搜索）。 XML Extender 提供了用户定义的抽取函数，您可以在 SQL SELECT 子句中为每种 SQL 数据类型指定该函数。

检索元素的内容和属性值对于开发应用程序是很有用的，因为您可以将 XML 数据作为关系数据来访问。例如，在表 SALES_TAB 的 ORDER 列中可能存储了 1000 个 XML 文档。可以检索已经订购了项目的所有客户的姓名，即通过使用下列在 SELECT 子句中带有抽取 UDF 的 SQL 语句来检索此信息:

```
SELECT extractVarchar(Order, '/Order/Customer/Name') from sales_order_view
   WHERE price > 2500.00
```

在此示例中，抽取 UDF 将从 ORDER 列中检索作为 VARCHAR 数据类型的元素 <customer>。位置路径为 /Order/Customer/Name（参见第48页的『位置路径』以了解位置路径语法）。另外，通过使用 WHERE 子句减少返回值的个数，该子句指定只有带有子元素 <ExtendedPrice> 的 <customer> 元素的内容才具有大于 2500.00 的值。

要抽取元素内容或属性值： 通过将下列语法用作表或*标量函数*，使用第112页的表 13中所列示的抽取 UDF:

extract*retrieved_datatype*(*xmlobj*, *path*)

其中:

retrieved_datatype

是从抽取函数返回的数据类型；它可以是下列类型之一:

- INTEGER
- SMALLINT
- DOUBLE
- REAL
- CHAR
- VARCHAR
- CLOB
- DATE
- TIME
- TIMESTAMP
- FILE

xmlobj 是要从其中抽取元素或属性的 XML 列的名称。此列必须定义为下列 XML 用户定义的类型之一:

- XMLVARCHAR
- XMLCLOB as LOCATOR
- XMLFILE

path 为 XML 文档中的元素或属性的位置路径（如 /Order/Customer/Name）。参见第48页的『位置路径』以了解位置路径语法。

重要事项: 注意抽取 UDF 支持具有谓词的位置路径，但这些谓词只能带属性而不能带元素。例如，下列谓词受支持:

```
'/Order/Part[@color="black "]/ExtendedPrice'
```

下列谓词不受支持:

```
'/Order/Part/Shipment/[Shipdate < "11/25/00"]'
```

表13显示了抽取函数，采用标量格式和表格式:

表 13. XML Extender 抽取函数

标量函数	表函数	返回的列名（表函数）	返回类型
extractInteger()	extractIntegers()	returnedInteger	INTEGER
extractSmallint()	extractSmallints()	returnedSmallint	SMALLINT
extractDouble()	extractDoubles()	returnedDouble	DOUBLE
extractReal()	extractReals()	returnedReal	REAL
extractChar()	extractChars()	returnedChar	CHAR
extractVarchar()	extractVarchars()	returnedVarchar	VARCHAR
extractCLOB()	extractCLOBs()	returnedCLOB	CLOB
extractDate()	extractDates()	returnedDate	DATE
extractTime()	extractTimes()	returnedTime	TIME
extractTimestamp()	extractTimestamps()	returnedTimestamp	TIMESTAMP

标量函数示例:

在下例中，当键的属性值 = ″1″ 时返回一个值。该值是在将整数自动转换为 DECIMAL 类型时抽取的。

```
CREATE TABLE t1(key decimal(3,2));
INSERT into t1 values
SELECT * from table(db2xml.extractInteger(db2xml.XMLFile
   ('c:\dxx\samples\xml\getstart.xml'), '/Order/[@key="1"]'));
SELECT * from t1;
```

表函数示例:

在以下示例中，以 INTEGER 的形式抽取销售订单的每个键值

```
SELECT * from table(db2xml.extractIntegers(db2xml.XMLFile
   ('c:\dxx\samples\xml\getstart.xml'), '/Order/@key')) as x;
```

更新 XML 数据

利用 XML Extender，可以通过替换 XML 列数据来更新整个 XML 文档，或者可以更新指定的元素或属性的值。

任务概述:

1. 确保您已经在 XML 表中存储了 XML 文档，并确定您想检索哪些数据。

2. 选择一种用来更新 DB2 表中的数据的方法（强制转型函数或 UDF）。
3. 指定 SQL 查询，该查询指定要更新的 XML 表和列。

重要事项: 当更新为 XML 启用的一列时， XML Extender 将自动更新辅助表以反映所作的更改。但是，不要直接更新这些表，而不通过更改相应的 XML 元素或属性值来更新存储在 XML 列中的原始 XML 文档。 直接更新将导致数据不一致问题。

要更新 XML 文档:

使用下列方法之一:

使用缺省强制转型函数

对于每个用户定义类型 (UDT)，都存在缺省强制转型函数，用于将 SQL 基本类型强制转型为 UDT。可以使用 XML Extender 提供的强制转型函数来更新 XML 文档。 第107页的表9显示了所提供的强制转型函数，并假定创建的 ORDER列属于 XML Extender 提供的另一个 UDT。

示例: 根据强制转型的 VARCHAR 类型来更新 XMLVARCHAR 类型，假定 `xml_buf` 是定义为 VARCHAR 类型的主变量。

```
UPDATE sales_tab VALUES('123456', 'Sriram Srinivasan',
  db2xml.XMLVarchar(:xml_buff))
```

使用存储器 UDF

对于每个 XML Extender UDT，都存在一个存储器 UDF，用来将数据从不同于其基本类型的资源中导入 DB2 中。可以使用存储器 UDF 来更新整个 XML 文档，方法是替换整个文档。

示例: 下列示例通过使用 XMLVarcharFromFile() 函数来更新 XML 文档:

```
UPDATE sales_tab
       set order = XMLVarcharFromFile('c:\dxx\samples\cmd\getstart.xml')
   WHERE sales_person = 'Sriram Srinivasan'
```

上述示例将 XML 对象从文件 `c:\dxx\samples\cmd\getstart.xml` 更新至表 SALES_TAB 中的列 ORDER 中。

有关 XML Extender 提供的存储器函数的列表，参见第107页的表10。

要更新 XML 文档的特定元素和属性:

使用 Update() UDF 在文档中一次更新一个值，而不是更新整个文档。通过使用 UDF，可以指定要替换的位置路径所表示的元素或属性的位置路径和值。（参见第48页的『位置路径』以了解位置路径语法。）不需要编辑 XML 文档：XML Extender 将替您更改。

“更新”UDF 更新整个 XML 文件并基于 XML 语法分析程序的信息重新构造该文件。参见第181页的『“更新”函数如何处理 XML 文档』，以了解“更新”UDF 在更新文档和示例文档之前和之后处理这些文档的方式。

语法:

```
Update(xmlobj, path, value)
```

其中:

xmlobj 是要更新其元素或属性的值的 XML 列名。

path 是要更新的元素或属性的位置路径。参见第48页的『位置路径』以了解位置路径语法。参见第183页的『多次出现』以了解多次出现的注意事项。

value 是要更新的值。

示例: 通过使用 Update() UDF，下列语句将 <Customer> 元素的值更新为字符串 IBM:

```
UPDATE sales_tab
       set order = Update(order, '/Order/Customer/Name', 'IBM')
   WHERE sales_person = 'Sriram Srinivasan'
```

多次出现:

当在 Update() UDF 中提供了位置路径时，则会用提供的值更新具有匹配路径的每个元素或属性的内容。这意味着若文档具有多次出现的位置路径，则“更新”函数用 *value* 参数中提供的值替换现存值。

搜索 XML 文档

搜索 XML 数据与检索 XML 数据相似：两种方法都是检索数据以进一步进行处理，但是它们通过使用 WHERE 子句来将谓词定义为检索标准。

根据应用程序的需要，XML Extender 提供了几种方法来搜索 XML 列中的 XML 文档。 它具有搜索文档结构的能力，并根据元素内容和属性值来返回结果。 可以搜索 XML 列的视图及其辅助表，直接搜索辅助表将获得更好的性能，或者将抽取 UDF 配合 WHERE 子句使用。另外，可以使用 DB2 Text Extender，搜索结构化内容内的列数据，以查找文本字符串。

利用 XML Extender，您可以使用对辅助表列的索引来进行高速搜索，辅助表列中包含了从 XML 文档中抽取的 XML 元素内容或属性值。通过指定元素的数据类型或属性，您可以搜索 SQL 一般数据类型或执行范围搜索。例如，在采购单示例中，可以搜索价格高于 2500.00 的所有订单。

另外，可以使用 DB2 UDB Text Extender 来执行结构化文本搜索或全文本搜索。例如，您可以具有一列 RESUME，该列中包含 XML 格式的简历。您可能想要具有 Java 技巧的所有申请人的名称。可以使用 DB2 Text Extender 来搜索 XML 文档，以获得 <skill> 元素中包含字符串 JAVA 的所有简历。

下列各节描述搜索方法:

- 『按结构搜索 XML 文档』
- 第117页的『使用 Text Extender 来进行结构化文本搜索』

按结构搜索 XML 文档

通过使用 XML Extender 搜索功能，可以根据文档结构（也就是根据元素和属性）来搜索一列中的 XML 数据。要搜索列数据，可以按几种方式来使用 SELECT 语句，并根据与文档元素和属性的匹配情况，返回一个*结果集*。可以使用下列方法来搜索列数据:

- 利用直接查询辅助表来进行搜索
- 从*连接的视图*中搜索
- 利用抽取 UDF 来搜索
- 搜索多次出现的元素或属性

下列各节中描述了这些方法，并在下列方案中提供了示例。 应用程序表 SALES_TAB 中有一个名为 ORDER 的 XML 列。 此列有三个辅助表: ORDER_SIDE_TAB、PART_SIDE_TAB 和 SHIP_SIDE_TAB。 当启用 ORDER 列时指定了缺省视图 sales_order_view，并使用以下 CREATE VIEW 语句来连接这些表:

```
CREATE VIEW sales_order_view(invoice_num, sales_person, order,
                             order_key, customer, part_key, price, date)
     AS
     SELECT sales_tab.invoice_num, sales_tab.sales_person, sales_tab.order,
            order_side_tab.order_key, order_side_tab.customer,
            part_side_tab.part_key, ship_side_tab.date
     FROM sales_tab, order_side_tab, part_side_tab, ship_side_tab
     WHERE sales_tab.invoice_num = order_side_tab.invoice_num
                AND sales_tab.invoice_num = part_side_tab.invoice_num
                AND sales_tab.invoice_num = ship_side_tab.invoice_num
```

利用直接查询辅助表来进行搜索

当对辅助表进行索引时，对于结构化搜索，利用子查询搜索来直接查询将获得最佳性能。可以使用查询或子查询来正确搜索辅助表。

示例: 下列语句使用查询和子查询来直接搜索辅助表:

```
SELECT sales_person from sales_tab
       WHERE invoice_num in
          (SELECT invoice_num from part_side_tab
           WHERE price > 2500.00)
```

在此示例中，invoice_num 是 SALES_TAB 表中的主键。

从连接的视图中搜索

可以让 XML Extender 创建一个缺省视图，该缺省视图通过使用唯一的标识来连接应用程序表和辅助表。可以使用此缺省视图或连接应用程序表和辅助表的任何视图，来搜索列数据以及查询辅助表。此方法为应用程序表及其辅助表提供了单个虚拟视图。然而，创建的辅助表越多，查询的成本就越高。

提示: 当创建您自己的视图时，可以使用 *root_id* 或 DXXROOT_ID（它是由 XML Extender 创建的）来连接表。

示例: 下列语句将搜索一个视图:

```
SELECT sales_person from sales_order_view
   WHERE price > 2500.00
```

该 SQL 语句将从连接视图 sales_order_view 表中返回 sales_person 的值，这些值具有价格高于 2500.00 的行式项目订单。

利用抽取 UDF 来搜索

当您还没有为应用程序表创建索引或辅助表时，还可以使用 XML Extender 的抽取 UDF 来搜索元素和属性。使用抽取 UDF 来扫描 XML 数据的成本相当高，所以只应将它配合 WHERE 子句使用，该子句可以限制搜索中所包括的 XML 文档数。

示例: 下列语句利用抽取 XML Extender UDF 来进行搜索:

```
SELECT sales_person from sales_tab
     WHERE extractVarchar(order, '/Order/Customer/Name')
    like '%IBM%'
AND invoice_num > 100
```

在此示例中，抽取 UDF 以值 IBM 来抽取 </Order/Customer/Name> 元素。

搜索多次出现的元素或属性

当搜索多次出现的元素或属性时，可使用 DISTINCT 子句来防止重复的值。

示例: 下列语句利用 DISTINCT 子句来进行搜索:

```
SELECT sales_person from sales_tab
       WHERE invoice_num in
              (SELECT DISTINCT invoice_num from part_side_tab
          WHERE price > 2500.00)
```

在此示例中， DAD 文件指定 /Order/Part/Price 多次出现，并为它创建辅助表 PART_SIDE_TAB。PART_SIDE_TAB 表中可能有多个行具有相同的 invoice_num。通过使用 DISTINCT 以只返回唯一值。

使用 Text Extender 来进行结构化文本搜索

当搜索 XML 文档结构时，XML Extender 搜索已转换为一般数据类型的元素和属性值，但是不搜索文本。 可以使用 DB2 UDB Text Extender，对为 XML 启用的列进行结构化文本搜索或全文本搜索。 在 DB2 UDB 版本 6.1 或更高版本中，Text Extender 支持 XML 文档搜索。Text Extender 在 Windows 操作系统、AIX 和 Sun Solaris 中可用。

结构化文本搜索

搜索基于 XML 文档的树形结构的文本字符串。 例如，若您的文档结构为 /Order/Customer/Name，而您想在 <Customer> 子元素中搜索字符串『IBM』，则可以使用结构化文本搜索。 该文档可能还在 <Comment> 子元素中具有字符串 IBM，或将 IBM 作为产品的部件名。 仅在指定的元素中对字符串进行结构化文本搜索。 在此示例中，只能找到 </Order/Customer/Name> 子元素中具有 IBM 的文档；而不会返回在其他元素中具有 IBM、但是在 </Order/Customer/Name> 子元素中没有 IBM 的文档。

全文本搜索

在文档结构的任何地方搜索文本字符串，而不管是元素还是属性。 使用前面的示例，将返回具有字符串 IBM 的所有文档，而不管字符串 IBM 在何处出现。

要使用 Text Extender 搜索，必须安装 DB2 Text Extender 并按照下面所描述的那样来启用数据库和表。要了解如何使用 Text Extender 搜索，参见 *DB2 通用数据库 Text Extender 管理和程序设计*中有关利用 Text Extender 的 UDF 来进行搜索的章节。

为 Text Extender 启用 XML 列

假定您具有启用了 XML 的数据库，使用下列步骤来启用 Text Extender，以便搜索启用了 XML 的列的内容。为了提供示例，将数据库命名为 SALES_DB，将表命名为 ORDER，且 XML 列名为 XVARCHAR 和 XCLOB:

1. 要了解如何安装 Text Extender，参见 Extenders CD 上的 `install.txt` 文件。
2. 从下列位置之一来输入 **txstart** 命令:
   - 在 UNIX 操作系统上，从实例拥有者的命令提示符处输入该命令。
   - 在 Windows NT 上，从指定了 DB2INSTANCE 的命令窗口中输入该命令。
3. 打开 Text Extender 命令行窗口。此步骤假定您具有名称为 SALES_DB 的数据库以及名称为 ORDER 的表，该表具有名称为 XVARCHAR 和 XCLOB 的两个 XML 列 您可能需要在 `dxx\samples\c` 中运行样本程序。
4. 与数据库连接。在 **db2tx** 命令提示处，输入:

   ```
   'connect to SALES_DB'
   ```

5. 为 Text Extender 启用数据库。

   从 **db2tx** 命令提示处，输入:

   ```
   'enable database'
   ```

6. 为 Text Extender 启用 XML 表中的列，定义 XML 文档的数据类型、语言、代码页和关于该列的其他信息。
   - 对于 VARCHAR 列 XVARCHAR，输入:

     ```
     'enable text column order xvarchar function db2xml.varchartovarchar handle
     varcharhandle ccsid 850 language us_english format xml indextype precise
         indexproperty sections_enabled
     documentmodel (Order) updateindex update'
     ```

   - 对于 CLOB 列 XCLOB，输入:

     ```
     'enable text column order xclob function db2xml.clob handle clobhandle
      ccsid 850 language us_english indextype precise updateindex update'
     ```

7. 检查索引的状态。
   - 对于列 XVARCHAR，输入: `get index status order handle varcharhandle`
   - 对于列 XCLOB，输入: `get index status order handle clobhandle`
8. 在称为 `desmodel.ini` 的文档模型 INI 文件中定义 XML 文档模型。对于 UNIX，此文件位于: `/db2tx/txins000`，对于 Windows NT，此文件位于 `\instance\\db2tx\txins000`，以及初始化文件的各部分中。例如，对于 `textmodel.ini`:

   ```
   ; 文档模型列表
   [MODELS]
   modelname=Order

   ; an 'Order' document model definition
   ```

```
; left side = section name identifier
; right side = section name tag

[Order]
Order = /Order
Order/Customer/Name = /Order/Customer/Name
Order/Customer/Email = /Order/Customer/Email
Order/Part/@color = /Order/Part/@color
Order/Part/Shipment/ShipMode = /Order/Part/Shipment/ShipMode
```

使用 Text Extender 来搜索文本

Text Extender 的搜索功能可以很好地处理 XML Extender 文档结构化搜索。建议的方法是创建一个查询，该查询对文档元素或属性进行搜索，并使用 Text Extender 来搜索元素内容或属性值。

示例: 下列语句使用 Text Extender 来搜索 XML 文档文本。在 DB2 命令窗口中，输入:

```
'connect to SALES_DB'
'select xvarchar from order where db2tx.contains(varcharhandle,
 'model Order section(Order/Customer/Name) "Motors"')=1'
'select xclob from order where db2tx.contains(clobhandle,
 'model Order section(Order/Customer/Name) "Motors"')=1'
```

Text Extender Contains() UDF 搜索。

此示例不包含使用 Text Extender 来搜索列数据所需要的所有步骤。要了解 Text Extender 搜索概念和功能，参见 *DB2 通用数据库 Text Extender 管理和程序设计* 中有关利用 Text Extender 的 UDF 来进行搜索的章节。

删除 XML 文档

使用 SQL DELETE 语句来从 XML 列中删除 XML 文档行。可以指定 WHERE 子句来细化要删除哪些文档。

示例: 下列语句将把含有大于 2500.00 的 <ExtendedPrice> 值的所有文档删除掉。

```
DELETE from sales_tab
       WHERE invoice_num in
          (SELECT invoice_num from part_side_tab
           WHERE price > 2500.00)
```

从 JDBC 调用函数时的限制

当在函数中使用参数标志符时， JDBC 限制要求必须将函数的参数标志符强制转型为返回的数据将插入的列的数据类型。函数选择逻辑不了解该自变量可能导致的数据类型，它不能解析该引用。

因此，JDBC 不能解析下列代码:

```
db2xml.XMLdefault_casting_function(length)
```

可以使用 CAST 规范来为参数标志符提供类型，如 VARCHAR，然后可以执行函数选择逻辑:

```
db2xml.XMLdefault_casting_function(CAST(? AS cast_type(length))
```

示例 1: 在以下示例中，参数标志符被强制转型为 VARCHAR。正在传送的参数是一个 XML 文档，该文档被强制转型为 VARCHAR(1000) 并插入到列 ORDER 中。

```
String query = "insert into sales_tab(invoice_num, sales_person, order) values
    (?,?,db2xml.XMLVarchar(cast (? as varchar(1000))))";
```

示例 2: 在以下示例中，参数标志符被强制转型为 VARCHAR。正在传送的参数是一个文件名，其内容被转换为 VARCHAR 并插入到列 ORDER 中。

```
String query = "insert into sales_tab(invoice_num, sales_person, order) values
    (?,?,db2xml.XMLVarcharfromFILE(cast (? as varchar(1000))))";
```

第6章 管理 XML 集合数据

XML 集合是一组关系表，这些表中包含被映射至 XML 文档的数据。此访问和存储方法允许您从现存数据中组合 XML 文档，分解 XML 文档，并将 XML 用作交换方法。

关系表可以是分解 XML 文档时 XML Extender 所生成的新表，也可以是现存表，这些现存表中的数据将与 XML Extender 一起使用，以便为应用程序生成 XML 文档。 这些表中的列数据不包含 XML 标记；它包含分别与元素和属性相关联的内容和值。 存储过程用作存储、检索、更新、搜索和删除 XML 集合数据的访问和存储方法。

XML 集合存储过程所使用的参数限制在第257页的『附录D. XML Extender 限制』中进行介绍。

您可以增加存储过程结果的 CLOB 大小，如第186页的『增加 CLOB 限制』中所述。

有关管理 XML 集合的信息，参见下列各节:

- 『将 DB2 数据组合 XML 文档』
- 第129页的『将 XML 文档分解为 DB2 数据』

将 DB2 数据组合 XML 文档

组合就是从 XML 集合中的关系数据中生成一组 XML 文档。可以使用存储过程来组合 XML 文档。要使用这些存储过程，必须创建 DAD 文件，该文件指定了 XML 文档与 DB2 表结构之间的映射。存储过程使用 DAD 文件来组合 XML 文档。要了解如何创建 DAD 文件，参见第50页的『计划 XML 集合』。

开始之前

- 将 XML 文档的结构映射至包含了元素内容和属性值的关系表。
- 选择映射方法: SQL 映射或 RDB_node 映射。
- 准备 DAD 文件。有关详情，参见第50页的『计划 XML 集合』。
- 可选择启用 XML 集合。

© Copyright IBM Corp. 1999, 2000

组合 XML 文档

XML Extender 提供了两个存储过程 dxxGenXML() 和 dxxRetrieveXML()，以组合 XML 文档。

dxxGenXML()

此存储过程用于以下应用程序：那些偶尔进行更新的应用程序，或不想要因管理 XML 数据而带来额外开销的应用程序。存储过程 dxxGenXML() 不需要已启用的集合；而是使用 DAD 文件。

存储过程 dxxGenXML() 通过使用存储在 XML 集合表中的数据来构造 XML 文档，这些 XML 集合表是通过 DAD 文件中的 <Xcollection> 元素指定的。此存储过程将每个 XML 文档作为一行来插入*结果表*中。还可以打开结果表上的游标，并取装结果集。结果表应该由应用程序来创建，并且始终具有 VARCHAR、CLOB、XMLVARCHAR 或 XMLCLOB 类型的一列。

另外，若将 DAD 文件中的验证元素指定为“是”，则 XML Extender 将 INTEGER 类型的列 DXX_VALID 添加至结果表，并且为有效的 XML 文档插入值 1，为无效的 XML 文档插入值 0。

存储过程 dxxGenXML() 还允许您指定在结果表中要生成的最大行数。这将缩短处理时间。该存储过程将返回该表中的实际行数，以及任何返回码和信息。

对应于分解的存储过程是 dxxShredXML()；它也将 DAD 作为输入参数并且不需要启用 XML 集合。

要组合 XML 集合：dxxGenXML()

使用以下存储过程说明，在应用程序中嵌入存储过程调用：

```
dxxGenXML(CLOB(100K)     DAD,             /* input */
          char(resultTabName) resultTabName,   /* input */
          integer        overrideType,    /* input */
          varchar(1024)  override,        /* input */
          integer        maxRows,         /* input */
          integer        numRows,         /* output */
          long           returnCode,      /* output */
          varchar(1024)  returnMsg)       /* output */
```

有关完整的语法和示例，参见第195页的『dxxGenXML()』。

示例：下列示例将组合一个 XML 文档：

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
```

```
SQL TYPE is CLOB(100K) dad;              /* DAD */
SQL TYPE is CLOB_FILE  dadFile;          /* dad file */
char                   result_tab[32];   /* name of the result table */
char                   override[2];      /* override, will set to NULL*/
short                  overrideType;     /* defined in dxx.h */
short                  max_row;          /* maximum number of rows */
short                  num_row;          /* actual number of rows */
long                   returnCode;       /* return error code */
char                   returnMsg[1024];  /* error message text */
short                  dad_ind;
short                  rtab_ind;
short                  ovtype_ind;
short                  ov_inde;
short                  maxrow_ind;
short                  numrow_ind;
short                  returnCode_ind;
short                  returnMsg_ind;

EXEC SQL END DECLARE SECTION;

/* create table */
EXEC CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

/* read data from a file to a CLOB */
strcpy(dadfile.name,"c:\dxx\samples\dad\getstart_xcollection.dad");
dadfile.name_length = strlen("c:\dxx\samples\dad\getstart_xcollection.dad")
dadfile.file_options = SQL_FILE_READ;
EXEC SQL VALUES (:dadfile) INTO :dad;
strcpy(result_tab,"xml_order_tab");
override[0] = '\0';
overrideType = NO_OVERRIDE;
max_row = 500;
num_row = 0;
returnCode = 0;
msg_txt[0] = '\0';
collection_ind = 0;
dad_ind = 0;
rtab_ind = 0;
ov_ind = -1;
ovtype_ind = 0;
maxrow_ind = 0;
numrow_ind = -1;
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL dxxGenXML(:dad:dad_ind;
     :result_tab:rtab_ind,
     :overrideType:ovtype_ind,:override:ov_ind,
     :max_row:maxrow_ind,:num_row:numrow_ind,
     :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

因为在 DAD 文件中指定的 SQL 查询生成了 250 个 XML 文档，所以调用存储过程后的结果表中包含 250 行。

dxxRetrieveXML()

此存储过程可用于进行定期更新的应用程序。因为是重复同一任务，所以提高性能是很重要的。启用 XML 集合，并在存储过程中使用该集合名可提高性能。

除了存储过程 dxxRetrieveXML() 采用的是已启用的 XML 集合的名称，而不是 DAD 文件的名称以外，它与存储过程 dxxGenXML() 所起的作用相同。当启用 XML 集合后，DAD 文件被存储在 XML_USAGE 表中。因此，XML Extender 将检索 DAD 文件，从此以后，dxxRetrieveXML() 与 dxxGenXML() 就是完全相同的。

dxxRetrieveXML() 允许将同一 DAD 文件同时用于组合和分解。此存储过程还可用于检索分解的 XML 文档。

对应于分解的存储过程是 dxxInsertXML()；它也采用已启用的 XML 集合的名称。

要组合 XML 集合: dxxRetrieveXML()

使用以下存储过程说明，在应用程序中嵌入存储过程调用:

```
dxxRetrieveXML(char(collectionName) collectionName,  /* input */
               char(resultTabName) resultTabName,   /* input */
               integer         overrideType,    /* input */
               varchar(1024)   override,        /* input */
               integer         maxRows,         /* input */
               integer         numRows,         /* output */
               long            returnCode,      /* output */
               varchar(1024)   returnMsg)       /* output */
```

有关完整的语法和示例，参见第199页的『dxxRetrieveXML()』。

示例: 以下示例是对 dxxRetrieveXML() 的一个调用。 它假定结果表是用 XML_ORDER_TAB 的名称来创建的，并且结果表中有一列的类型是 XMLVARCHAR。

```
#include "dxx.h"
#include "dxxrc.h"

            EXEC SQL INCLUDE SQLCA;
            EXEC SQL BEGIN DECLARE SECTION;
              char    collection[32];   /* dad buffer */
              char    result_tab[32];   /* name of the result table */
              char    override[2];      /* override, will set to NULL*/
              short   overrideType;     /* defined in dxx.h */
              short   max_row;          /* maximum number of rows */
              short   num_row;          /* actual number of rows */
              long    returnCode;       /* return error code */
              char    returnMsg[1024];  /* error message text */
              short   dadbuf_ind;
              short   rtab_ind;
```

```
short   ovtype_ind;
short   ov_inde;
short   maxrow_ind;
short   numrow_ind;
short   returnCode_ind;
short   returnMsg_ind;

EXEC SQL END DECLARE SECTION;

/* create table */
EXEC SQL CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

/* initialize host variable and indicators */
strcpy(collection,"sales_ord");
strcpy(result_tab,"xml_order_tab");
override[0] = '\0';
overrideType = NO_OVERRIDE;
max_row = 500;
num_row = 0;
returnCode = 0;
msg_txt[0] = '\0';
collection_ind = 0;
rtab_ind = 0;
ov_ind = -1;
ovtype_ind = 0;
maxrow_ind = 0;
numrow_ind = -1;
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL dxxRetrieve(:collection:collection_ind;
  :result_tab:rtab_ind,
       :overrideType:ovtype_ind,:override:ov_ind,
       :max_row:maxrow_ind,:num_row:numrow_ind,
       :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

动态覆盖 DAD 文件中的值

对于动态查询，可以使用两个可选参数来覆盖 DAD 文件中的条件: *override* 和 *overrideType*。根据 *overrideType* 中的输入，应用程序可以覆盖 SQL 映射的 <SQL_stmt> 标记值，或者覆盖 DAD 中 RDB_node 映射的 RDB_nodes 中的条件。

这些参数具有下列值和规则:

overrideType

此参数是标记 *override* 参数的类型所必需的输入参数 (IN)。 *overrideType* 具有下列值:

NO_OVERRIDE

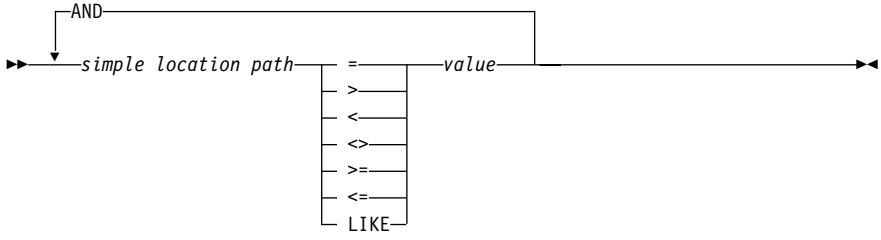
指定不要覆盖 DAD 文件中的条件。

SQL_OVERRIDE
指定要使用 SQL 语句来覆盖 DAD 文件中的条件。

XML_OVERRIDE
指定要使用基于 XPath 的条件来覆盖 DAD 文件中的条件。

override
此参数是一个可选输入参数 (IN)，它为 DAD 文件指定覆盖条件。输入值语法对应于在 *overrideType* 上指定的值。

- 若指定 NO_OVERRIDE，则输入值为 NULL 字符串。
- 若指定 SQL_OVERRIDE，则输入值为一个有效的 SQL 语句。**要求:** 若使用 SQL_OVERRIDE 和 SQL 语句，则必须在 DAD 文件中使用 SQL 映射模式。输入的 SQL 语句将覆盖在 DAD 文件中由 <SQL_stmt> 元素指定的 SQL 语句。
- 若使用 XML_OVERRIDE，则输入值是包含了一个或多个表达式的字符串。**要求:** 若使用 XML_OVERRIDE 和一个表达式，则必须在 DAD 文件中使用 RDB_node 映射模式。输入 XML 表达式将覆盖 DAD 文件中所指定的 RDB_node 条件。该表达式使用下列语法:



其中:

simple location path
是一个简单位置路径，使用由 XPATH 定义的语法；要了解有关语法，参见第49页的表5。

运算符
可使用空格来将运算符与表达式的其他部分隔开。

值 一个数值或以单引号引起来的字符串。

可选择在运算的两边空出空格；但 LIKE 运算符两边的空格是必需的。

当指定了 XML_OVERRIDE 值时，与简单位置路径匹配的 text_node 或 attribute_node 中的 RDB_node 的条件，将被指定的表达式覆盖。

XML_OVERRIDE 与 XPath 不完全兼容。简单位置路径仅用来标识映射至列的元素或属性。

示例:

下列示例显示如何使用 SQL_OVERRIDE 和 XML_OVERRIDE 来进行动态覆盖。本书中的大多数存储过程示例都使用 NO_OVERRIDE。

示例: 一个使用 SQL_OVERRIDE 的存储过程。

```
include "dxx.h"
include "dxxrc.h"

       EXEC SQL INCLUDE SQLCA;
       EXEC SQL BEGIN DECLARE SECTION;
         char    collection[32];  /* dad buffer */
         char    result_tab[32];    /* name of the result table */
         char    override[256];   /* override, SQL_stmt */
         short   overrideType;      /* defined in dxx.h */
         short   max_row;         /* maximum number of rows */
         short   num_row;         /* actual number of rows */
         long    returnCode;      /* return error code */
         char    returnMsg[1024];   /* error message text */
         short   rtab_ind;
         short   ovtype_ind;
         short   ov_inde;
         short   maxrow_ind;
         short   numrow_ind;
         short   returnCode_ind;
         short   returnMsg_ind;

         EXEC SQL END DECLARE SECTION;

         /* create table */
         EXEC CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

         /* initialize host variable and indicators */
         strcpy(collection,"sales_ord");
         strcpy(result_tab,"xml_order_tab");
         sprintf(override,"%s %s %s %s %s %s %s",
                   "SELECT o.order_key, customer, p.part_key, quantity, price,",
                   "tax, ship_id, date, mode ",
                   "FROM order_tab o, part_tab p,",
                   "table(select substr(char(timestamp(generate_unique())),16",
                   "as ship_id,date,mode from ship_tab)as s",
                   "WHERE p.price > 50.00 and s.date >'1998-12-01' AND",
                   "p.order_key = o.order_key and s.part_key = p.part_key");
        overrideType = SQL_OVERRIDE;
         max_row = 500;
         num_row = 0;
         returnCode = 0;
         msg_txt[0] = '\0';
         collection_ind = 0;
```

```
rtab_ind = 0;
ov_ind = 0;
ovtype_ind = 0;
maxrow_ind = 0;
numrow_ind = -1;
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL dxxRetrieve(:collection:collection_ind;
       :result_tab:rtab_ind,
       :overrideType:ovtype_ind,:override:ov_ind,
       :max_row:maxrow_ind,:num_row:numrow_ind,
       :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

在此示例中，DAD 文件中的 <xcollection> 元素必须具有一个 <SQL_stmt> 元素。通过将价格更改为高于 50.00，并将日期更改为大于 1998-12-01， *override* 参数会覆盖 <SQL_stmt> 的值。

示例: 一个使用 XML_OVERRIDE 的存储过程。

```
include "dxx.h"
include "dxxrc.h"

       EXEC SQL INCLUDE SQLCA;
       EXEC SQL BEGIN DECLARE SECTION;
         char    collection[32];  /* dad buffer */
         char    result_tab[32];  /* name of the result table */
         char    override[256];   /* override, SQL_stmt */
         short   overrideType;    /* defined in dxx.h */
         short   max_row;         /* maximum number of rows */
         short   num_row;         /* actual number of rows */
         long    returnCode;      /* return error code */
         char    returnMsg[1024]; /* error message text */
         short   dadbuf_ind;
         short   rtab_ind;
         short   ovtype_ind;
         short   ov_inde;
         short   maxrow_ind;
         short   numrow_ind;
         short   returnCode_ind;
         short   returnMsg_ind;

         EXEC SQL END DECLARE SECTION;

         /* create table */
         EXEC CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

         /* initialize host variable and indicators */
         strcpy(collection,"sales_ord");
         strcpy(result_tab,"xml_order_tab");
         sprintf(override,"%s %s",
                  "/Order/Part/Price > 50.00 AND ",
                  "Order/Part/Shipment/ShipDate > '1998-12-01'");
```

```
overrideType = XML_OVERRIDE;
 max_row = 500;
 num_row = 0;
 returnCode = 0;
 msg_txt[0] = '\0';
 collection_ind = 0;
 rtab_ind = 0;
 ov_ind = 0;
 ovtype_ind = 0;
 maxrow_ind = 0;
 numrow_ind = -1;
 returnCode_ind = -1;
 returnMsg_ind = -1;

 /* Call the store procedure */
 EXEC SQL CALL dxxRetrieve(:collection:collection_ind;
        :result_tab:rtab_ind,
        :overrideType:ovtype_ind,:override:ov_ind,
        :max_row:maxrow_ind,:num_row:numrow_ind,
        :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

在此示例中，DAD 文件中的 <collection> 元素具有根 element_node 的 RDB_node。*override* 值是基于 XML 内容的。 XML Extender 将简单位置路径转换为所映射的 DB2 列。

将 XML 文档分解为 DB2 数据

分解 XML 文档就是将 XML 文档内的数据分类，然后将它存储在关系表中。XML Extender 提供了将 XML 数据从源 XML 文档分解为关系表的存储过程。要使用这些存储过程，必须创建 DAD 文件，该文件指定了 XML 文档与 DB2 表结构之间的映射。这些存储过程使用 DAD 文件来分解 XML 文档。 要了解如何创建 DAD 文件，参见第50页的『计划 XML 集合』。

启用 XML 集合以进行分解

大多数情况下，在使用存储过程之前，需要启用 XML 集合。 在下列情况下，需要启用 XML 集合:

- 在将 XML 文档分解到新表中时，必须启用 XML 集合，因为 XML 集合中的所有表都是在启用该集合时，由 XML Extender 创建的。
- 保持要出现多次的元素和属性的顺序很重要。XML Extender 只为启用集合时所创建的表保持多次出现的元素或属性的顺序。将 XML 文档分解为现存关系表时，不能保证会保留该顺序。

当数据库中已经存在表时，若想自发地传送 DAD 文件，则不需要启用 XML 集合。

分解表大小限制

通过将元素和属性值抽取到表行中，分解使用 RDB_node 映射来指定将 XML 文档分解为 DB2 表的方式。每个 XML 文档的值都被存储在一个或多个 DB2 表中。每个表自每个文档最多可分解 1024 行。

例如，如果 XML 文档被分解为五个表，则对于该特定文档，这五个表中的每一个最多可有 1024 行。如果该表包含有多个文档的多个行，则对于每个文档，该表最多可有 1024 行。如果该表有 20 个文档，则它可有 20,480 行，即每个文档 1024 行。

使用多次出现的元素（具有在 XML 结构中可多次出现的位置路径的元素）对行数会有影响。 例如，包含出现了 20 次的元素 <Part> 的文档在表中可能被分解为 20 行。使用多次出现的元素时，要考虑此表大小限制。

开始之前

- 将 XML 文档的结构映射至包含了元素内容和属性值的关系表。
- 使用 RDB_node 映射来准备 DAD 文件。有关详情，参见第50页的『计划 XML 集合』。
- 可选择启用 XML 集合。

分解 XML 文档

XML Extender 提供了两个存储过程 dxxShredXML() 和 dxxInsertXML，用来分解 XML 文档。

dxxShredXML()

此存储过程用于以下应用程序：那些偶尔进行更新的应用程序，或不想要因管理 XML 数据而带来额外开销的应用程序。存储过程 dxxShredXML() 不需要启用集合；而是使用 DAD 文件。

存储过程 dxxShredXML() 采用两个输入参数： DAD 文件和要分解的 XML 文档；它返回两个输出参数：返回码和返回信息。

存储过程 dxxShredXML() 根据输入 DAD 文件中的 <Xcollection> 规范，将 XML 文档插入 XML 集合中。假定 DAD 文件的 <Xcollection> 中所使用的表已经存在，并假定表中的各列满足 DAD 映射中所指定的数据类型。若这种假定不成立，则会返回错误消息。然后，存储过程 dxxShredXML() 将分解 XML 文档，并将未标记的 XML 数据插入在 DAD 文件中所指定的表中。

对应于组合的存储过程是 dxxGenXML()；它也将 DAD 作为输入参数并且不需要启用 XML 集合。

要分解 XML 集合: dxxShredXML()

使用以下存储过程说明，在应用程序中嵌入存储过程调用:

```
dxxShredXML(CLOB(100K)    DAD,            /* input */
            CLOB(1M)       xmlobj,           /* input */
            long           returnCode,       /* output */
            varchar(1024)  returnMsg)        /* output */
```

有关完整的语法和示例，参见第204页的『dxxShredXML()』。

示例: 以下是调用 dxxShredXML() 的一个示例:

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
     EXEC SQL BEGIN DECLARE SECTION;
     SQL TYPE is CLOB      dad;             /* DAD*/
     SQL TYPE is CLOB_FILE dadFile;         /* DAD file*/
     SQL TYPE is CLOB      xmlDoc;          /* input XML document */
     SQL TYPE is CLOB_FILE xmlFile;          /* input XML file */
     long                  returnCode;       /* error code */
     char                  returnMsg[1024];  /* error message text */
     short                 dad_ind;
     short                 xmlDoc_ind;
     short                 returnCode_ind;
     short                 returnMsg_ind;
     EXEC SQL END DECLARE SECTION;

     /* initialize host variable and indicators */
    strcpy(dadFile.name,"c:\dxx\samples\dad\getstart_xcollection.dad");
    dadFile.name_length=strlen("c:\dxx\samples\dad\getstart_xcollection.dad");
    dadFile.file_option=SQL_FILE_READ;
    strcpy(xmlFile.name,"c:\dxx\samples\cmd\getstart_xcollection.xml");
    xmlFile.name_length=strlen("c:\dxx\samples\cmd\getstart_xcollection.xml");
    xmlFile.file_option=SQL_FILE_READ;
    SQL EXEC VALUES (:dadFile) INTO :dad;
    SQL EXEC VALUES (:xmlFile) INTO :xmlDoc;
    returnCode = 0;
    returnMsg[0] = '\0';
    dad_ind = 0;
    xmlDoc_ind = 0;
    returnCode_ind = -1;
    returnMsg_ind = -1;

     /* Call the store procedure */
    EXEC SQL CALL db2xml.dxxShredXML(:dad:dad_ind;
                  :xmlDoc:xmlDoc_ind,
                  :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

dxxInsertXML()

此存储过程可用于进行定期更新的应用程序。因为是重复同一任务，所以提高性能是很重要的。启用 XML 集合，并在存储过程中使用该集合名可

提高性能。 除了 dxxInsertXML() 将已启用的 XML 集合作为其第一个输入参数之外，存储过程 dxxInsertXML() 与 dxxShredXML() 所起的作用相同。

存储过程 dxxInsertXML() 将 XML 文档插入已启用的 XML 集合中，该集合与 DAD 文件是相关联的。DAD 文件中包含集合表和映射的规范。检查或创建集合表是根据 <Xcollection> 中的规范来进行的。然后，存储过程 dxxInsertXML() 根据映射来分解 XML 文档，并将未标记的 XML 数据插入已命名的 XML 集合的表中。

对应于组合的存储过程是 dxxRetrieveXML()；它也采用已启用的 XML 集合的名称。

要分解 XML 集合: dxxInsertXML()

使用以下存储过程说明，在应用程序中嵌入存储过程调用:

```
dxxInsertXML(char(collectionName) collectionName,   /* input */
             CLOB(1M)       xmlobj,           /* input */
             long           returnCode,       /* output */
             varchar(1024)  returnMsg)        /* output */
```

有关完整的语法和示例，参见第206页的『dxxInsertXML()』。

示例: 以下是调用 dxxInsertXML() 的一个示例:

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
        EXEC SQL BEGIN DECLARE SECTION;
         char                  collection[64];   /* name of an XML collection */
         SQL TYPE is CLOB_FILE xmlFile;          /* input XML file */
         SQL TYPE is CLOB      xmlDoc;           /* input XML doc */
         long                  returnCode;       /* error code */
         char                  returnMsg[1024];  /* error message text */
         short                 collection_ind;
         short                 xmlDoc_ind;
         short                 returnCode_ind;
         short                 returnMsg_ind;
         EXEC SQL END DECLARE SECTION;

         /* initialize host variable and indicators */
         strcpy(collection,"sales_ord")
         strcpy(xmlobj.name,"c:\dxx\samples\cmd\getstart_xcollection.xml");
         xmlobj.name_length=strlen("c:\dxx\samples\cmd\getstart_xcollection.xml")
         xmlobj.file_option=SQL_FILE_READ;
         SQL EXEC VALUES (:xmlFile) INTO (:xmlDoc);
         returnCode = 0;
         returnMsg[0] = '\0';
         collection_ind = 0;
         xmlobj_ind = 0;
         returnCode_ind = -1;
```

```
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL db2xml.dxxInsertXML(:collection:collection_ind;
            :xmlDoc:xmlDoc_ind,
            :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

访问 XML 集合

可以更新、删除、搜索和检索 XML 集合。但是，应记住使用 XML 集合的目的是要存储或检索数据库表中未标记的纯数据。现存数据库表中的数据与任何入局 XML 文档都没有任何关系；更新、删除和搜索操作实际上是对这些表进行常规的 SQL 访问。若数据是从入局 XML 文档分解得来的，则没有任何原始 XML 文档会继续存在。

XML Extender 提供了从 XML 集合视图对数据执行操作的能力。通过使用 UPDATE 和 DELETE SQL 语句，就可以修改用来组合 XML 文档的数据，因此，更新 XML 集合。

注意事项:

- 要更新文档，就不要删除包含了表的主键的行，该行是其他集合表的外键行。当删除了主键行和外键行时，文档就被删除了。
- 要替换或删除元素和属性值，您可以删除和插入较低级表中的行，而不删除该文档。
- 要删除文档，则删除组合 DAD 中所指定的顶部 element_node 的那些行。

更新 XML 集合中的数据

XML Extender 允许您更新存储在 XML 集合表中的未标记的数据。更新 XML 集合表值时，就同时更新了 XML 元素的文本或 XML 属性的值。另外，更新时可以删除多次出现的元素或属性中数据的实例。

从 SQL 角度来看，更改元素或属性的值是一个更新操作，删除元素或属性的实例是一个删除操作。从 XML 角度来看，只要根 element_node 的元素文本或属性值存在，则 XML 文档仍然存在，因此，这是一个更新操作。

要求: 要更新 XML 集合中的数据，必须遵守下列规则。

- 当现存表具有此关系时，指定集合表中主外键的关系。若不具有此关系，则必须确保有可以连接的列。
- 包括在 DAD 文件中所指定的连接条件:
  - 对于 SQL 映射，在 <SQL_stmt> 元素中
  - 对于 RDB_node 映射，在根元素节点的 RDB_node 中

更新元素和属性值

在 XML 集合中，元素文本和属性值都被映射到数据库表的各列中。 不管列数据是先前已存在还是从入局 XML 文档中分解得来的，都可以使用正常的 SQL 更新技术来替换该列数据。

要更新元素或属性值，在 SQL UPDATE 语句中指定 WHERE 子句，在该语句中包含在 DAD 文件中所指定的连接条件。

例如:

```
UPDATE SHIP_TAB
    set MODE = 'BOAT'
  WHERE MODE='AIR' AND PART_KEY in
    (SELECT PART_KEY from PART_TAB WHERE ORDER_KEY=68)
```

在 SHIP_TAB 表中，<ShipMode> 元素值从 AIR 更新为 BOAT，其中，键为 68。

删除元素和属性实例

要通过除去多次出现的元素或属性，来更新组合的 XML 文档，则使用 WHERE 子句，来删除包含了与这些元素或属性值相对应的字段值的那一行。只要您不删除包含了顶部 element_node 的值的那一行，就将删除元素值视为对 XML 文档的更新。

例如，在以下 DELETE 语句中，通过指定 <shipment> 元素的其中一个子元素的唯一值，来删除该元素。

```
DELETE from SHIP_TAB
   WHERE DATE='1999-04-12'
```

指定一个 DATE 值将删除与此值匹配的行。组合的文档最初包含两个 <shipment> 元素，但是现在只包含一个元素。

删除 XML 集合中的 XML 文档

可以删除从集合中组合的 XML 文档。这意味着如果您具有组合多个 XML 文档的 XML 集合，则可以删除这些组合的文档中的其中一个文档。

要删除文档，则删除组合顶部 element_node 的表中的行，该顶部 element_node 是在 DAD 文件中指定的。此表中包含顶级集合表的主键以及较低级表的外键。

例如，下列 DELETE 语句指定了主键列的值。

```
DELETE from order_tab
   WHERE order_key=1
```

ORDER_KEY 是表 ORDER_TAB 中的主键，并且在组合 XML 文档时，ORDER_KEY 是顶部 element_node。删除此行就会删除在组合期间所生成的一个 XML 文档。因此，从 XML 角度来看，已从 XML 集合中删除了一个 XML 文档。

从 XML 集合中检索 XML 文档

从 XML 集合中检索 XML 文档类似于从集合来组合文档。

要检索 XML 文档，使用存储过程 dxxRetrieveXML()。参阅第199页的『dxxRetrieveXML()』中的语法和示例。

DAD 文件注意事项: 当将 XML 文档分解到 XML 集合中时，可能会丢失多次出现的元素和属性值的次序，除非在 DAD 文件中指定了该次序。要保持此次序，您应该使用 RDB_node 映射模式。此映射模式使您可对表指定 orderBy 属性，该表包含它的 RDB_node 中的根元素。

搜索 XML 集合

本节描述根据下列目标来搜索 XML 集合:

- **使用搜索条件来生成 XML 文档**:

  此任务实际上是使用条件来组合的。可以使用下列搜索条件来指定搜索条件:

  – 在 DAD 文件的 text_node 和 attribute_node 中指定条件。

  – 当使用 dxxGenXML() 和 dxxRetrieveXML() 存储过程时，指定 *overwrite* 参数。

    例如，若已经使用 DAD 文件 `order.dad` 启用了 XML 集合 sales_ord，但是您现在想使用从 Web 中派生的格式数据来覆盖价格，则可以覆盖 <SQL_stmt> DAD 元素的值，如下所示:

```
EXEC SQL INCLUDE SQLCA;
        EXEC SQL BEGIN DECLARE SECTION;
       ...

         EXEC SQL END DECLARE SECTION;

         float    price_value;

         /* create table */
              EXEC SQL CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

         /* initialize host variable and indicators */
         strcpy(collection,"sales_ord");
         strcpy(result_tab,"xml_order_tab");
         overrideType = SQL_OVERRIDE;
         max_row = 20;
         num_row = 0;
```

```
returnCode = 0;
msg_txt[0] = '\0';
override_ind = 0;
overrideType_ind = 0;
rtab_ind = 0;
maxrow_ind = 0;
numrow_ind = -1;
returnCode_ind = -1;
returnMsg_ind = -1;

/* get the price_value from some place, such as form data */
price_value = 1000.00         /* for example*/

/* specify the overwrite */
sprintf(overwrite,
       "SELECT o.order_key, customer, p.part_key, quantity, price,
              tax, ship_id, date, mode
FROM order_tab o, part_tab p,
           table(select substr(char(timestamp(generate_unique())),16)
           as ship_id, date, mode from ship_tab)as s
       WHERE p.price > %d and s.date >'1996-06-01' AND
           p.order_key = o.order_key and s.part_key = p.part_key",
           price_value);

/* Call the store procedure */
EXEC SQL CALL db2xml.dxxRetrieve(:collection:collection_ind,
       :result_tab:rtab_ind,
       :overrideType:overrideType_ind,:overwrite:overwrite_ind,
       :max_row:maxrow_ind,:num_row:numrow_ind,
             :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

order.dad 中的条件 price > 2500.00 被 price > ? 覆盖，其中 ? 基于输入变量 *price_value*。

- **搜索分解的 XML 数据**:

  可以使用常规的 SQL 查询操作来搜索集合表。您可以连接集合表，或者使用子查询，然后对文本列执行结构化文本搜索。利用结构化搜索的结果，可以应用该数据来检索或生成指定的 XML 文档。

第4部分 参考

此部分提供了关于 XML Extender 管理命令的语法信息、用户定义的数据类型 (UDT)、用户定义函数 (UDF) 以及存储过程。还提供了消息文本，以便于确定问题。

© Copyright IBM Corp. 1999, 2000

第7章 XML Extender 管理命令: dxxadm

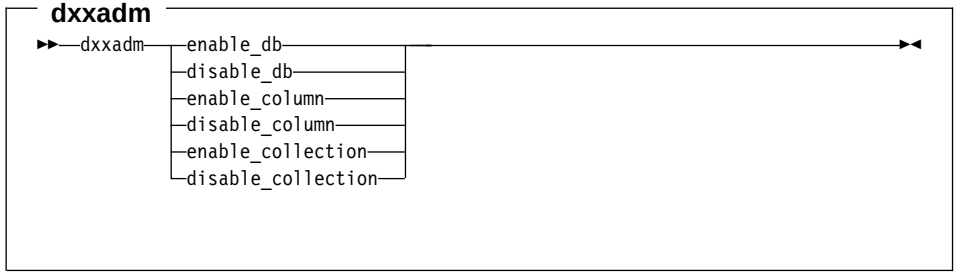
XML Extender 提供了管理命令 **dxxadm** 来完成下列管理任务: 通过使用各种命令选项调用 **dxxadm** 来执行下列 XML Extender 管理任务:

- 对 XML Extender 启用或禁用数据库
- 启用或禁用 XML 列
- 启用或禁用 XML 集合

还可使用 XML Extender 管理向导或存储过程来执行每个管理任务。

高级语法

下列语法图解提供了 **dxxadm** 命令的高级语法。以下部分提供了每一选项的描述。



管理命令选项

以下章节描述系统程序员可用的每个 **dxxadm** 命令选项:

- 第140页的『enable_db』
- 第141页的『disable_db』
- 第142页的『enable_column』
- 第144页的『disable_column』
- 第146页的『enable_collection』
- 第147页的『disable_collection』

© Copyright IBM Corp. 1999, 2000

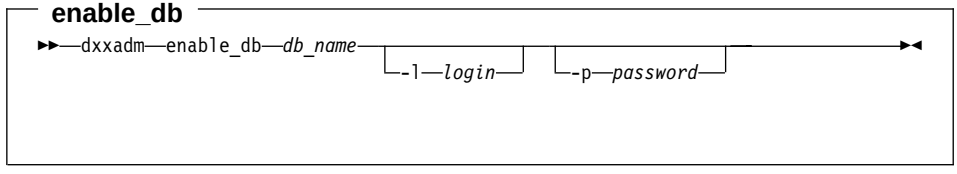
enable_db

目的

连接并启用数据库以便 XML Extender 可使用它。当启用了该数据库时，XML Extender 会创建下列对象:

- XML Extender 用户定义类型 (UDT)。
- XML Extender 用户定义函数 (UDF)。
- XML Extender DTD 参考表 DTD_REF，它存储 DTD 和关于每个 DTD 的信息。有关 DTD_REF 表的完整描述，参见第209页的『DTD 参考表』。
- XML Extender 用法表 XML_USAGE，它存储有关对 XML 和每个集合启用的每一列的公共信息。 有关 XML_USAGE 表的完整描述，参见第210页的『XML 用法表』。

格式



参数

表 14. enable_db 参数

参数	描述
db_name	XML 数据所在的数据库的名称。
-l *login*	用户标识（可选），用于在指定该项时连接数据库。 如果未指定该项，则使用当前用户标识。
-p *password*	口令（可选），用于在指定该项时连接数据库。 如果未指定该项，则使用当前口令。

例

以下示例启用数据库 SALES_DB。

```
dxxadm enable_db SALES_DB
```

disable_db

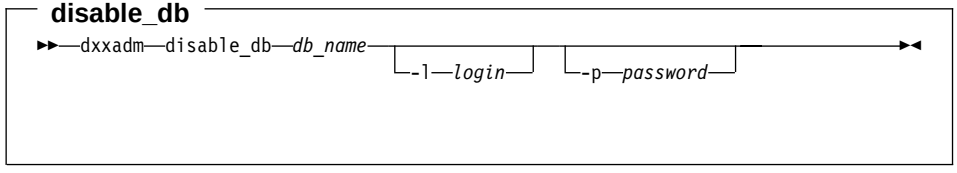
目的

连接并禁用启用了 XML 的数据库。当禁用数据库时， XML Extender 将不再能使用它。当 XML Extender 禁用该数据库时，它删除下列对象:

- XML Extender 用户定义类型 (UDT)。
- XML Extender 用户定义函数 (UDF)。
- XML Extender DTD 参考表 DTD_REF，它存储 DTD 和关于每个 DTD 的信息。有关 DTD_REF 表的完整描述，参见第209页的『DTD 参考表』。
- XML Extender 用法表 XML_USAGE，它存储有关对 XML 和每个集合启用的每一列的公共信息。 有关 XML_USAGE 表的完整描述，参见第210页的『XML 用法表』。

重要事项: 必须在尝试禁用数据库之前，禁用所有的 XML 列 XML Extender 不能禁用包含对 XML 启用的列或集合的数据库。

格式



参数

表 15. disable_db 参数

参数	描述
db_name	XML 数据所在的数据库的名称
-l *login*	用户标识（可选），用于在指定该项时连接数据库。 如果未指定该项，则使用当前用户标识。
-p *password*	口令（可选），用于在指定该项时连接数据库。 如果未指定该项，则使用当前口令。

例

以下示例禁用数据库 SALES_DB。

```
dxxadm disable_db SALES_DB
```

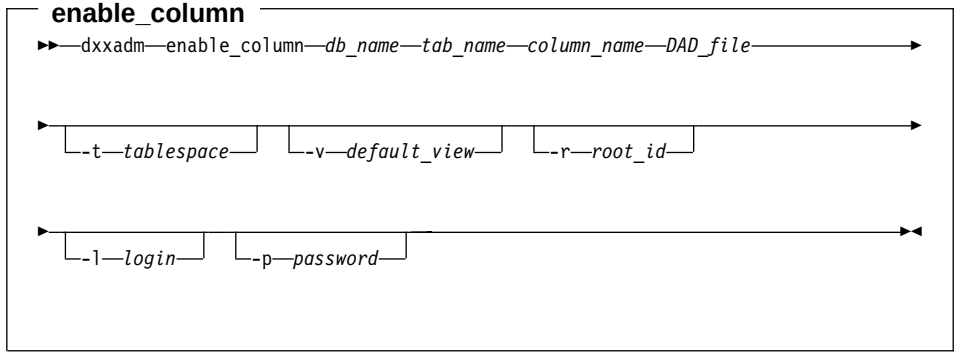
enable_column

目的

连接数据库并启用 XML 列，以使它可包含 XML Extender UDT。当启用一列时，XML Extender 完成下列任务:

- 确定 XML 表是否具有主键；若没有，则 XML Extender 将改变 XML 表，并添加称为 DXXROOT_ID 的一列。
- 使用一列（其中有 XML 表中的每一行的唯一标识符），来创建在 DAD 文件中指定的辅助表。 此列可以是用户指定的 *root_id* 或者是由 XML Extender 命名的 DXXROOT_ID。
- 有选择性地使用您指定的名称，来创建 XML 表及其辅助表的缺省视图。

格式



参数

表 16. enable_column 参数

参数	描述
db_name	XML 数据所在的数据库的名称。
tab_name	XML 列所在的表的名称。
column_name	XML 列的名称。
DAD_file	DAD 文件的名称，该文件将 XML 文档映射至 XML 列和辅助表。
-t *tablespace*	表空间，它是可选的且包含与 XML 列相关的辅助表。如果未指定该项，则使用缺省表空间。

表 16. enable_column 参数 (续)

参数	描述
-v *default_view*	缺省视图的名称（可选），该视图连接 XML 列和辅助表。
-r *root_id*	XML 列中的主键名称，它要用作辅助表的 root_id。root_id 是可选的。
-l *login*	用户标识（可选），用于在指定该项时连接数据库。 如果未指定该项，则使用当前用户标识。
-p *password*	口令（可选），用于在指定该项时连接数据库。 如果未指定该项，则使用当前口令。

例

以下示例启用一个 XML 列。

```
dxxadm enable_column SALES_DB SALES_TAB ORDER -v sales_order_view -r INVOICE_NUMBER
```

disable_column

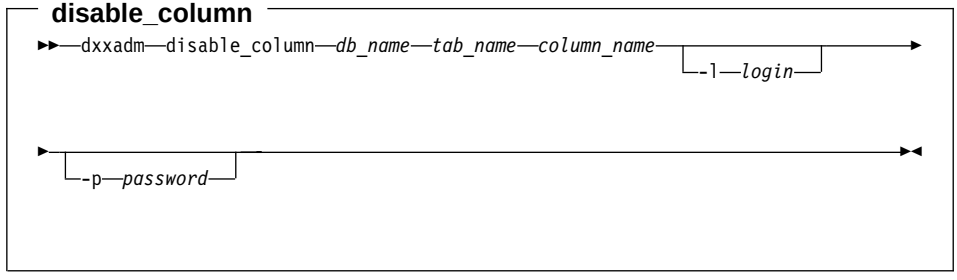
目的

连接数据库并禁用启用了 XML 的列。当禁用该列时，它不再包含 XML 数据类型。 当启用了 XML 的列被禁用时，执行下列操作:

- 从 XML_USAGE 表删除 XML 列用法项。
- 在 DTD_REF 表中减去 USAGE_COUNT。
- 删除与此列相关的所有触发器。
- 删除与此列相关的所有辅助表。

重要事项: 必须在删除 XML 表之前禁用 XML 列。 如果删除 XML 表而未禁用其 XML 列，则 XML Extender 保存它所创建的两个辅助表及 XML_USAGE 表中的 XML 列项。

格式



参数

表 17. disable_column 参数

参数	描述
db_name	数据所在的数据库的名称。
tab_name	XML 列所在的表的名称。
column_name	XML 列的名称。
-l *login*	用户标识（可选），用于在指定该项时连接数据库。 如果未指定该项，则使用当前用户标识。
-p *password*	口令（可选），用于在指定该项时连接数据库。 如果未指定该项，则使用当前口令。

例

以下示例禁用一个启用了 XML 的列。

```
dxxadm disable_column SALES_DB SALES_TAB ORDER
```

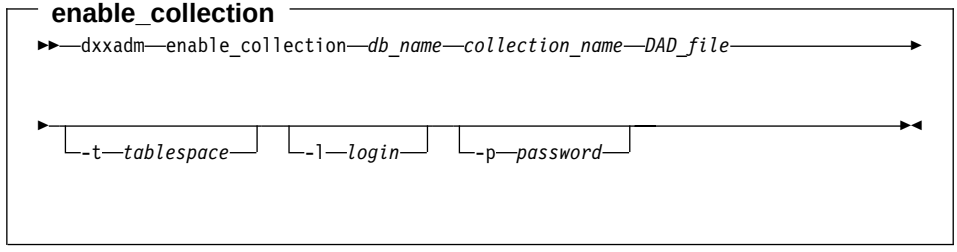
enable_collection

目的

连接数据库并按指定的 DAD 启用 XML 集合。当启用集合时， XML Extender 执行下列任务:

- 在 XML_USAGE 表中创建 XML 集合用法项。
- 对于 RDB_node 映射，如果数据库中不存在 DAD 中指定的集合表，则创建它们。

格式



参数

表 18. enable_collection 参数

参数	描述
db_name	数据所在的数据库的名称。
collection_name	XML 集合的名称。
DAD_file	DAD 文件的名称，该文件将 XML 文档映射至集合中的关系表。
-t *tablespace*	表空间的名称，它是可选的并与集合相关联。 如果未指定该项，则使用缺省表空间。
-l *login*	用户标识（可选），用于在指定该项时连接数据库。 如果未指定该项，则使用当前用户标识。
-p *password*	口令（可选），用于在指定该项时连接数据库。 如果未指定该项，则使用当前口令。

例

以下示例启用一个 XML 集合。

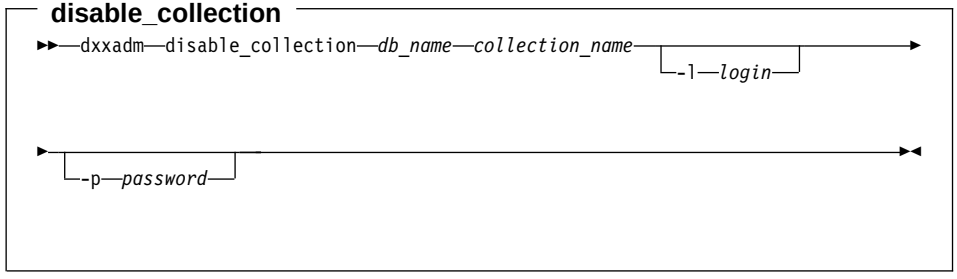
```
dxxadm enable_collection SALES_DB sales_ord getstart_xcollection.dad -t orderspace
```

disable_collection

目的

连接数据库并禁用启用了 XML 的集合。不能再在组合 (dxxRetrieveXML) 存储过程和分解 (dxxInsertXML) 存储过程中使用集合名。 当禁用 XML 集合时，从 XML_USAGE 表删除相关集合项。 注意，禁用集合不会删除在 enable_collection 步骤期间创建的集合表。

格式



参数

表 19. disable_collection 参数

参数	描述
db_name	数据所在的数据库的名称。
collection_name	XML 集合的名称。
-l *login*	用户标识（可选），用于在指定该项时连接数据库。 如果未指定该项，则使用当前用户标识。
-p *password*	口令（可选），用于在指定该项时连接数据库。 如果未指定该项，则使用当前口令。

例

下列示例禁用一个 XML 集合。

```
dxxadm disable_collection SALES_DB sales_ord
```

第8章 XML Extender 用户定义类型

XML Extender 用户定义类型 (UDT) 是用于 XML 列和 XML 集合的数据类型。所有 UDT 都具有模式名 db2xml。 XML Extender 创建 UDT 以用于存储和检索 XML 文档。 表20包含 UDT 的概述。

表 20. XML Extender UDT

用户定义类型列	源数据类型	用法说明
XMLVARCHAR	VARCHAR(*varchar_len*)	将整个 XML 文档作为 VARCHAR 存储在 DB2 中。
XMLCLOB	CLOB(*clob_len*)	将整个 XML 文档作为字符大对象 (CLOB) 存储在 DB2 中。
XMLFILE	VARCHAR(512)	指定本地文件服务器的文件名。如果对 XML 列指定 XMLFILE, 则 XML Extender 将 XML 文档存储在外部服务器文件中。Text Extender 不能用 XMLFILE 来启用。您有责任确保文件内容和 DB2, 以及被创建用来创建索引的辅助表间的完整性。

其中 *varchar_len* 和 *clob_len* 是特定于操作系统的。

对于 DB2 UDB, *varchar_len* = 3K 而 *clob_len* = 2G。

这些 UDT 仅用于指定应用程序列的类型; 它们不应用于 XML Extender 创建的辅助表。

© Copyright IBM Corp. 1999, 2000

第9章 XML Extender 用户定义函数

XML Extender 提供用于存储、检索、搜索和更新 XML 文档，以及抽取 XML 元素或属性的函数。将 XML 用户定义函数 (UDF) 用于 XML 列，但不将其用于 XML 集合。所有 UDF 具有模式名 db2xml，可在 UDF 的前面忽略它。

XML Extender 函数的四种类型为: 存储器函数、检索函数、抽取函数和更新函数。

存储器函数

存储器函数将 XML 文档插入 DB2 数据库。有关语法和示例，参见第152页的『存储器函数』。

检索函数

检索函数从 DB2 数据库中的 XML 列检索 XML 文档。有关的语法和示例，参见第157页的『检索函数』。

抽取函数

抽取函数从 XML 文档抽取元素内容或属性值，并将其转换为由函数名指定的数据类型。XML Extender 提供了一组抽取函数，以用于各种 SQL 数据类型。有关语法和示例，参见第164页的『抽取函数』。

更新函数

Update() 函数修改元素内容或属性值，并使用由位置路径指定的更新值来返回 XML 文档的副本。Update() 函数允许应用程序员指定要更新的元素或属性。有关语法和示例，参见第180页的『更新函数』。

表21提供 XML Extender 函数的摘要。

表 21. XML Extender 用户定义函数

类型	函数
存储器函数	XMLVarcharFromFile()
	XMLCLOBFromFile()
	XMLFileFromVarchar()
	XMLFileFromCLOB()
检索函数	Content(): 从 XMLFile 检索至 CLOB
	Content(): 从 XMLVarchar 检索至外部服务器文件
	Content(): 从 XMLCLOB 检索至外部服务器文件

© Copyright IBM Corp. 1999, 2000

表 21. XML Extender 用户定义函数 (续)

类型	函数
抽取函数	extractInteger() 和 extractIntegers()
	extractSmallint() 和 extractSmallints()
	extractDouble() 和 extractDoubles()
	extractReal() 和 extractReals()
	extractChar() 和 extractChars()
	extractVarchar() 和 extractVarchars()
	extractCLOB() 和 extractCLOBs()
	extractDate() 和 extractDates()
	extractTime() 和 extractTimes()
	extractTimestamp() 和 extractTimestamps()
更新函数	Update()

当在 UDF 中使用参数标志符时，JDBC 限制要求必须将 UDF 的参数标志符强制转型为返回的数据将插入的列的数据类型。要了解如何强制转型参数标志符，参见第120页的『从 JDBC 调用函数时的限制』。

存储器函数

使用存储器函数将 XML 文档插入 DB2 数据库。您可直接在 INSERT 或 SELECT 语句中使用 UDT 的缺省强制转型函数， 如第106页的『存储数据』中所述。另外， XML Extender 提供了 UDF 以从不同于 UDT 基本数据类型的源访问 XML 文档，并将它们转换为期望的 UDT。

XML Extender 提供了下列存储器函数:

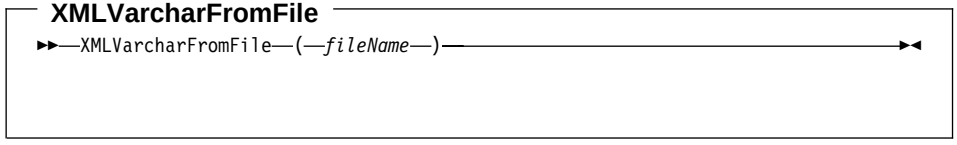
- 第153页的『XMLVarcharFromFile()』
- 第154页的『XMLCLOBFromFile()』
- 第155页的『XMLFileFromVarchar()』
- 第156页的『XMLFileFromCLOB()』

XMLVarcharFromFile()

目的

从服务器文件读取 XML 文档，并以 XMLVARCHAR 类型的形式返回该文档。

语法



参数

表 22. XMLVarcharFromFile 参数

参数	数据类型	描述
fileName	VARCHAR(512)	全限定服务器文件名。

返回类型

XMLVARCHAR

示例

以下示例从服务器文件读取 XML 文档，并以 XMLVARCHAR 类型的形式将其插入到 XML 列中。

```
EXEC SQL INSERT INTO sales_tab(ID, NAME, ORDER)
   VALUES('1234', 'Sriram Srinivasan',
           XMLVarcharFromFile('c:\dxx\samples\cmd\getstart.xml'))
```

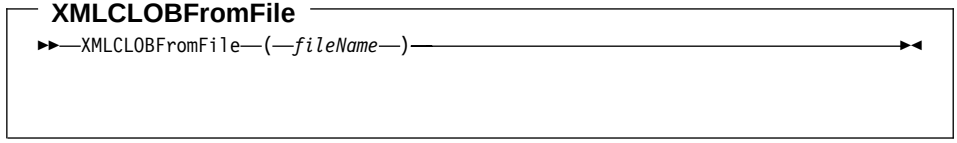
在本示例中，记录被插入 SALES_TAB 表。函数 XMLVarcharFromFile() 将 XML 文档从一个文件导入DB2，并将其作为 XMLVARCHAR 存储。

XMLCLOBFromFile()

目的

从服务器文件读取 XML 文档，并以 XMLCLOB 类型的形式返回该文档。

语法



参数

表 23. XMLCLOBFromFile 参数

参数	数据类型	描述
fileName	VARCHAR(512)	全限定服务器文件名。

返回类型

作为 LOCATOR 的 XMLCLOB

示例

以下示例从服务器文件读取 XML 文档，并以 XMLCLOB 类型的形式将其插入到 XML 列中。

```
EXEC SQL INSERT INTO sales_tab(ID, NAME, ORDER)
   VALUES('1234', 'Sriram Srinivasan',
           XMLCLOBFromFile('c:\dxx\samples\cmd\getstart.xml'))
```

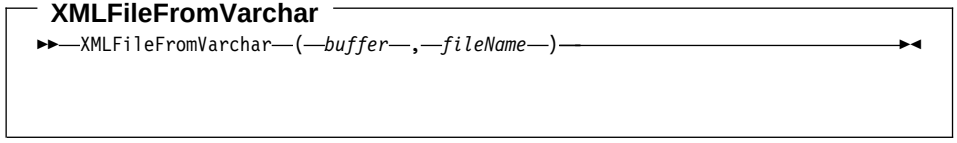
SALES_TAB 表中的列 ORDER 被定义为 XMLCLOB 类型。上述示例显示如何将列 ORDER 插入 SALES_TAB 表。

XMLFileFromVarchar()

目的

从内存以 VARCHAR 的形式读取 XML 文档，将其写至外部服务器文件，并以 XMLFILE 类型的形式返回文件名和路径。

语法



参数

表 24. XMLFileFromVarchar 参数

参数	数据类型	描述
buffer	VARCHAR(3K)	内存缓冲区。
fileName	VARCHAR(512)	全限定服务器文件名。

返回类型

XMLFILE

示例

以下示例从内存以 VARCHAR 格式读取 XML 文档，将其写至外部服务器文件，并以 XMLFILE 类型将文件名和路径插入 XML 列。

```
EXEC SQL BEGIN DECLARE SECTION;
      struct { short len; char data[3000]; } xml_buff;
EXEC SQL END DECLARE SECTION;

EXEC SQL INSERT INTO sales_tab(ID, NAME, ORDER)
   VALUES('1234', 'Sriram Srinivasan',
       XMLFileFromVarchar(:xml_buf, 'c:\dxx\samples\cmd\getstart.xml'))
```

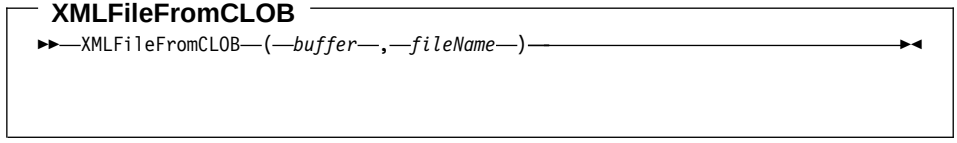
SALES_TAB 表中的列 ORDER 被定义为 XMLFILE 类型。上述示例显示如果您在缓冲区中有一个 XML 文档，则可将其存储在服务器文件中。

XMLFileFromCLOB()

目的

以 CLOB 定位器的形式读取 XML 文档，将其写至外部服务器文件，并以 XMLFILE 类型的形式返回文件名和路径。

语法



参数

表 25. XMLFileFromCLOB() 参数

参数	数据类型	描述
buffer	作为 LOCATOR 的 CLOB	包含 XML 文档的缓冲区。
fileName	VARCHAR(512)	全限定服务器文件名。

返回类型

XMLFILE

示例

以下示例以 CLOB 定位器的形式读取 XML 文档，将其写至外部服务器文件，并以 XMLFILE 类型的形式将文件名和路径插入 XML 列。

```
EXEC SQL BEGIN DECLARE SECTION;
     SQL TYPE IS CLOB_LOCATOR xml_buff;
EXEC SQL END DECLARE SECTION;

EXEC SQL INSERT INTO sales_tab(ID, NAME, ORDER)
   VALUES('1234', 'Sriram Srinivasan',
           XMLFileFromCLOB(:xml_buf, 'c:\dxx\samples\cmd\getstart.xml'))
```

SALES_TAB 表中的列 ORDER 被定义为 XMLFILE 类型。如果您在缓冲区中有一个 XML 文档，则可将其存储在服务器文件中。

检索函数

您可使用缺省强制转型函数来将 XML UDT 转换为基本数据类型， 如第108页的『检索整个文档』中所述。 XML Extender 还提供了*过载函数* Content()，用于进行检索。

Content() 函数提供下列类型的检索:

- **从服务器的外部存储器检索至客户机的主变量。**

  当 XML 文档作为外部服务器文件存储时，可使用 Content() 将其检索至内存缓冲区. 为此目的可以使用 第158页的『Content(): 从 XMLFILE 检索至 CLOB』。

- **从内部存储器检索至外部服务器文件**

  还可使用 Content() 来检索存储在 DB2 内的 XML 文档，并将其存储到 DB2 服务器文件系统上的服务器文件。下列 Content() 函数用于在外部服务器文件上存储信息:

  – 第160页的『Content(): 从 XMLVARCHAR 检索至外部服务器文件』
  – 第162页的『Content(): 从 XMLCLOB 检索至外部服务器文件』

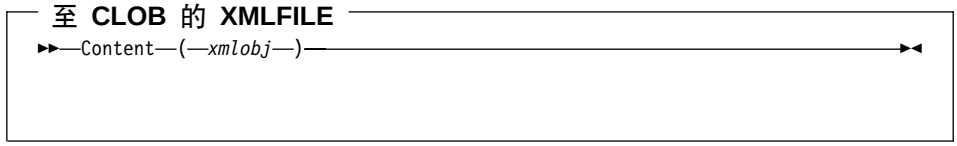
以下部分中的示例假定您正在使用 DB2 命令外壳，您不需要在每个命令的起始处输入 'DB2'。

Content(): 从 XMLFILE 检索至 CLOB

目的

从服务器文件检索数据并将其存储在 CLOB LOCATOR 中。

语法



参数

表 26. 至 CLOB 参数的 XMLFILE

参数	数据类型	描述
xmlobj	XMLFILE	XML 文档。

返回类型

作为 LOCATOR 的 CLOB (*clob_len*)

clob_len 对于 DB2 UDB 为 2G。

示例

下列示例从服务器文件检索数据，并将其存储在 CLOB 定位器中。

```
EXEC SQL BEGIN DECLARE SECTION;
     SQL TYPE IS CLOB_LOCATOR xml_buff;
EXEC SQL END DECLARE SECTION;

EXEC SQL CONNECT TO SALES_DB

EXEC SQL DECLARE c1 CURSOR FOR

     SELECT Content(order) from sales_tab
     WHERE sales_person = 'Sriram Srinivasan'

EXEC SQL OPEN c1;

do {
   EXEC SQL FETCH c1 INTO :xml_buff;
   if (SQLCODE != 0) {
      break;
      }
    else {
     /* do with the XML doc in buffer */
```

```
    }
  }

EXEC SQL CLOSE c1;

EXEC SQL CONNECT RESET;
```

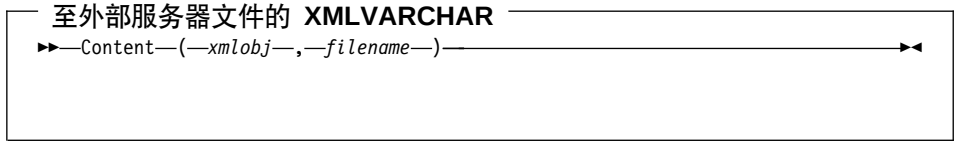
SALES_TAB 表中的列 ORDER 为 XMLFILE 类型，所以 Content() UDF 从服务器文件检索数据并将其存储在 CLOB 定位器中。

Content(): 从 XMLVARCHAR 检索至外部服务器文件

目的

检索作为 XMLVARCHAR 类型存储的 XML 内容，并将其存储在外部服务器文件中。

语法



重要事项: 如果带有指定名称的文件已存在，则内容函数会覆盖其内容。

注:

参数

表 27. 至外部服务器文件参数的 XMLVarchar

参数	数据类型	描述
xmlobj	XMLVARCHAR	XML 文档。
filename	VARCHAR(512)	全限定服务器文件名。

返回类型

VARCHAR(512)

示例

下例检索作为 XMLVARCHAR 类型存储的 XML 内容，并将其存储在外部服务器文件中。

```
CREATE table app1 (id int NOT NULL, order db2xml.XMLVarchar);
INSERT into app1 values (1, '<?xml version="1.0"?>
  <!DOCTYPE SYSTEM c:\dxx\samples\dtd\getstart.dtd"->
    <Order key="1">
      <Customer>
          <Name>American Motors</Name>
          <Email>parts@am.com</Email>
      </Customer>
      <Part color="black">
          <key>68</key>
          <Quantity>36</Quantity>
          <ExtendedPrice>34850.16</ExtendedPrice>
          <Tax>6.000000e-02</Tax>
```

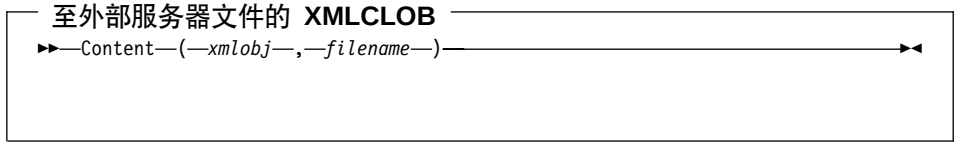
```
            <Shipment>
                <ShipDate>1998-08-19</ShipDate>
                <ShipMode>AIR </ShipMode>
            </Shipment>
            <Shipment>
                <ShipDate>1998-08-19</ShipDate>
                <ShipMode>BOAT </ShipMode>
            </Shipment>
    </Order>');

SELECT db2xml.Content(order, 'c:\dxx\samples\cmd\getstart_.dad')
  from app1 where ID=1;
```

Content(): 从 XMLCLOB 检索至外部服务器文件

目的

检索作为 XMLCLOB 类型存储的 XML 内容，并将其存储在外部服务器文件中。

语法



重要事项: 如果带有指定名称的文件已存在，则内容函数会覆盖其内容。

参数

表 28. 至外部服务器文件参数的 XMLCLOB

参数	数据类型	描述
xmlobj	作为 LOCATOR 的 XMLCLOB	XML 文档。
filename	VARCHAR(512)	全限定服务器文件名。

返回类型

VARCHAR(512)

示例

下例检索作为 XMLCLOB 类型存储的 XML 内容，并将其存储在外部服务器文件中。

```
CREATE table app1 (id int NOT NULL, order db2xml.XMLCLOB not logged);

INSERT into app1 values (1, '<?xml version="1.0"?>
  <!DOCTYPE SYSTEM c:\dxx\samples\dtd\getstart.dtd"->
    <Order key="1">
      <Customer>
          <Name>American Motors</Name>
          <Email>parts@am.com</Email>
      </Customer>
      <Part color="black">
          <key>68</key>
          <Quantity>36</Quantity>
          <ExtendedPrice>34850.16</ExtendedPrice>
          <Tax>6.000000e-02</Tax>
          <Shipment>
              <ShipDate>1998-08-19</ShipDate>
```

```
            <ShipMode>AIR </ShipMode>
        </Shipment>
        <Shipment>
            <ShipDate>1998-08-19</ShipDate>
            <ShipMode>BOAT </ShipMode>
        </Shipment>
    </Order>');

SELECT db2xml.Content(order, 'c:\dxx\samples\cmd\getstart.xml')
from app1 where ID=1;
```

抽取函数

抽取函数从 XML 文档抽取元素内容或属性值，并返回请求的 SQL 数据类型。XML Extender 提供了一组抽取函数，以用于各种 SQL 数据类型。抽取函数采用两个输入参数。第一个参数为 XML Extender UDT，它可以是 XML UDT 中的一个。第二个参数是指定 XML 元素或属性的位置路径。每个抽取函数都返回由位置路径指定的值或内容。

因为某些元素或属性值可出现多次，所以抽取函数会返回标量或表值；后者称为表函数。

XML Extender 提供下列抽取函数:

- 第165页的『extractInteger() 和 extractIntegers()』
- 第167页的『extractSmallint() 和 extractSmallints()』
- 第168页的『extractDouble() 和 extractDoubles()』
- 第170页的『extractReal() 和 extractReals()』
- 第171页的『extractChar() 和 extractChars()』
- 第172页的『extractVarchar() 和 extractVarchars()』
- 第174页的『extractCLOB() 和 extractCLOBs()』
- 第175页的『extractDate() 和 extractDates()』
- 第176页的『extractTime() 和 extractTimes()』
- 第178页的『extractTimestamp() 和 extractTimestamps()』

以下部分中的示例假定您正在使用 DB2 命令外壳，且您不需要在每个命令的起始处输入 'DB2'。

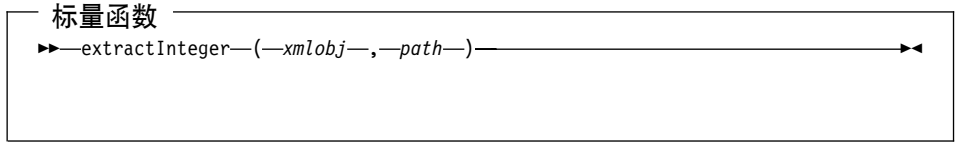
extractInteger() 和 extractIntegers()

目的

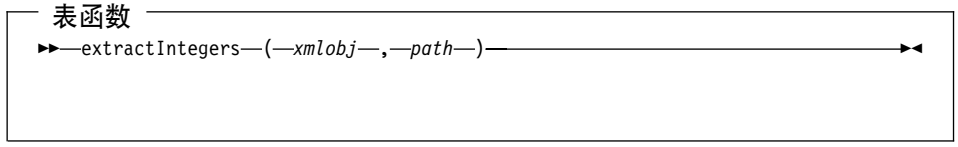
从 XML 文档抽取元素内容或属性值，并返回 INTEGER 类型的数据。

语法

标量函数



表函数



参数

表 29. extractInteger 函数参数和 extractIntegers 函数参数

参数	数据类型	描述
xmlobj	XMLVARCHAR、XMLFILE 或 XMLCLOB	列名。
path	VARCHAR	元素或属性的位置路径。

返回类型

INTEGER

返回列名（表函数）

returnedInteger

示例

标量函数示例:

在下例中，当键的属性值 = ″1″ 时返回一个值。该值是在整数自动转换为 DECIMAL 类型时抽取的。

```
CREATE TABLE t1(key decimal(3,2));
INSERT into t1 values
SELECT * from table(db2xml.extractInteger(db2xml.XMLFile
   ('c:\dxx\samples\xml\getstart.xml'), '/Order/[@key="1"]'));
SELECT * from t1;
```

表函数示例:

在以下示例中，销售订单的每个键值是以 INTEGER 形式抽取的

```
SELECT * from table(db2xml.extractIntegers(db2xml.XMLFile
   ('c:\dxx\samples\xml\getstart.xml'), '/Order/@key')) as x;
```

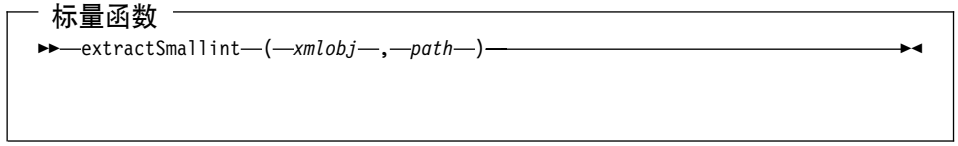
extractSmallint() 和 extractSmallints()

目的

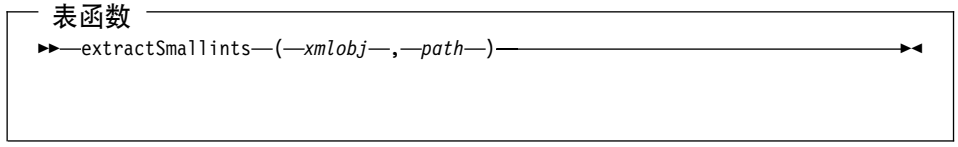
从 XML 文档抽取元素内容或属性值，并返回 SMALLINT 类型的数据。

语法

标量函数



表函数



参数

表 30. extractSmallint 函数参数和 extractSmallints 函数参数

参数	数据类型	描述
xmlobj	XMLVARCHAR、XMLFILE 或 XMLCLOB	列名。
path	VARCHAR	元素或属性的位置路径。

返回类型

SMALLINT

返回列名（表函数）

returnedSmallint

示例

下例中，所有销售订单中的 Quantity 值是以 SMALLINT 的形式抽取的

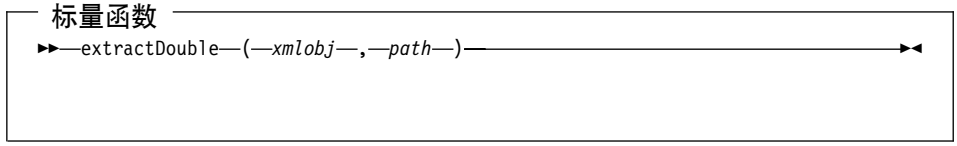
```
SELECT * from table(db2xml.extractSmallints(db2xml.File
   ('c:\dxx\samples\xml\getstart.xml'), '/Order/Part/Quantity')) as x;
```

extractDouble() 和 extractDoubles()

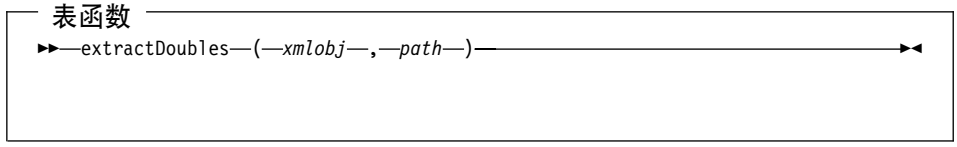
目的
从 XML 文档抽取元素内容或属性值，并返回 DOUBLE 类型的数据。

语法

标量函数



表函数



参数

表 31. extractDouble 函数参数和 extractDoubles 函数参数

参数	数据类型	描述
xmlobj	XMLVARCHAR、XMLFILE 或 XMLCLOB	列名。
path	VARCHAR	元素或属性的位置路径。

返回类型
DOUBLE

返回列名（表函数）
returnedDouble

示例
标量函数示例:

下例自动将订单中的价格从 DOUBLE 类型转换为 DECIMAL 类型。

```
CREATE TABLE t1(price, DECIMAL(5,2));
INSERT into t1 values (db2xml.extractDouble(db2xml.XMLFile
   ('c:\dxx\samples\xml\getstart.xml'),
    '/Order/Part[@color="black "]/ExtendedPrice'));
SELECT * from t1;
```

表函数示例:

下例中，销售订单每个部分中的 ExtendedPrice 值是以 DOUBLE 的形式抽取的。

```
SELECT * from table(db2xml.extractDoubles(db2xml.XMLFile
   ('c:\dxx\samples\xml\getstart.xml'), '/Order/Part/ExtendedPrice')) as x;
```

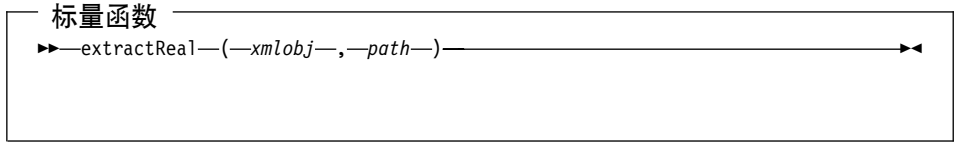
extractReal() 和 extractReals()

目的

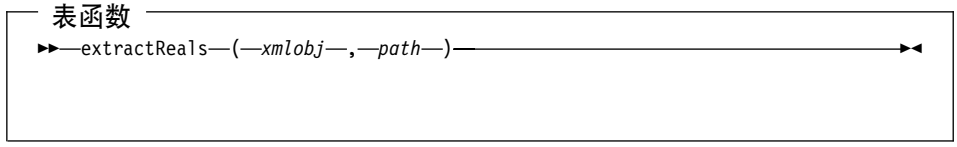
从 XML 文档抽取元素内容或属性值，并返回 REAL 类型的数据。

语法

标量函数



表函数



参数

表 32. extractReal 函数参数和 extractReals 函数参数

参数	数据类型	描述
xmlobj	XMLVARCHAR、XMLFILE 或 XMLCLOB	列名。
path	VARCHAR	元素或属性的位置路径。

返回类型

REAL

返回列名（表函数）

returnedReal

示例

在以下示例中，Tax 的每个值是以 REAL 形式抽取的。

```
SELECT * from table(db2xml.extractReals(db2xml.XMLFile
   ('c:\dxx\samples\xml\getstart.xml'), '/Order/Part/Tax')) as x;
```

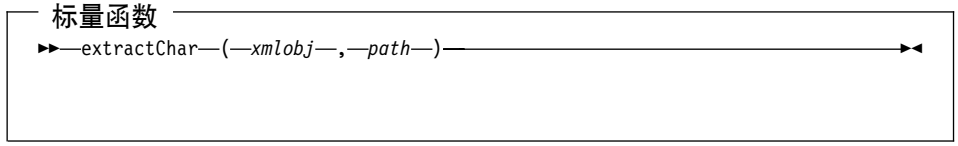
extractChar() 和 extractChars()

目的

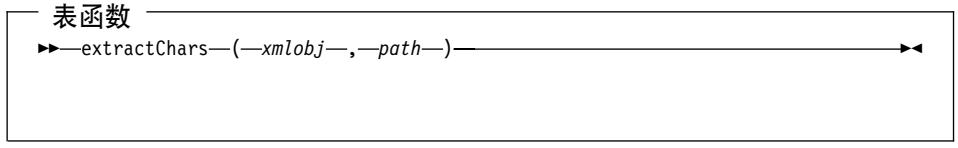
从 XML 文档抽取元素内容或属性值，并返回 CHAR 类型的数据。

语法

标量函数



表函数



参数

表 33. extractChar 函数参数和 extractChars 函数参数

参数	数据类型	描述
xmlobj	XMLVARCHAR、XMLFILE 或 XMLCLOB	列名。
path	VARCHAR	元素或属性的位置路径。

返回类型

CHAR

返回列名（表函数）

returnedChar

示例

下例中，Color 值是以 CHAR 的形式抽取的。

```
SELECT * from table(db2xml.extractChars(Order,
   ('c:\dxx\samples\xml\getstart.xml'), '/Order/Part/@Color')) as x;
```

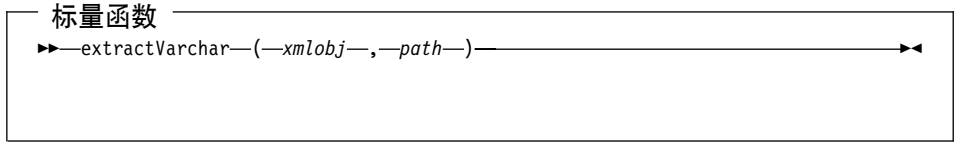
extractVarchar() 和 extractVarchars()

目的

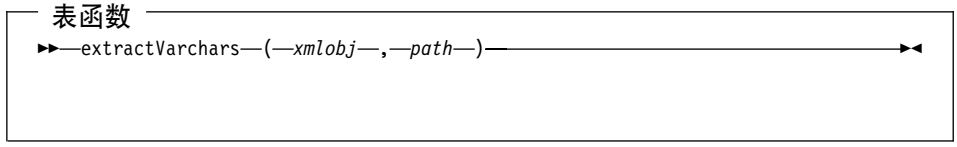
从 XML 文档抽取元素内容或属性值，并返回 VARCHAR 类型的数据。

语法

标量函数



表函数



参数

表 34. extractVarchar 函数参数和 extractVarchars 函数参数

参数	数据类型	描述
xmlobj	XMLVARCHAR、XMLFILE 或 XMLCLOB	列名。
path	VARCHAR	元素或属性的位置路径。

返回类型

VARCHAR(4K)

返回列名（表函数）

returnedVarchar

示例

在带有超过 1000 个 XML 文档（这些文档存储在 SALES_TAB 表内的列 ORDER 中）的数据库中，您可能想要查找已订购了 ExtendedPrice 高于 2500.00 的项目的所有客户。以下 SQL 语句在 SELECT 子句中使用抽取 UDF:

```
SELECT extractVarchar(Order, '/Order/Customer/Name') from sales_order_view
   WHERE price > 2500.00
```

UDF extractVarchar() 将列 ORDER 作为输入，并将位置路径 /Order/Customer/Name 作为选择标识符。UDF 返回客户的姓名。 使用 WHERE 子句，抽取函数仅估计那些 ExtendedPrice 超过 2500.00 的订购。

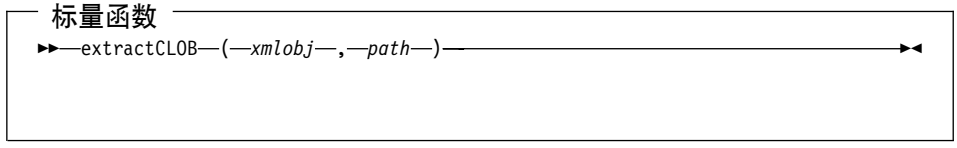
extractCLOB() 和 extractCLOBs()

目的

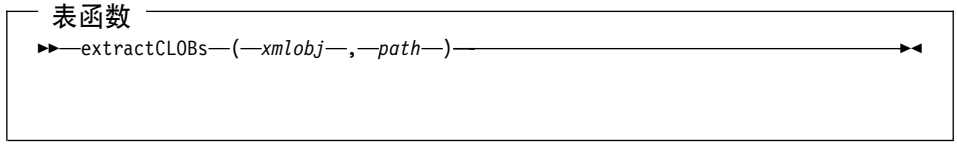
抽取 XML 文档的片段，带元素和属性标记、元素内容和属性，包括子元素。此函数与其他抽取函数不同；它们仅返回元素内容和属性。应使用 extractClob(s) 函数抽取文档片段，反之应使用 extractVarchar(s) 和 extractChar(s) 抽取简单值。

语法

标量函数



表函数



参数

表 35. extractCLOB 函数参数和 extractCLOBs 函数参数

参数	数据类型	描述
xmlobj	XMLVARCHAR、XMLFILE 或 XMLCLOB	列名。
path	VARCHAR	元素或属性的位置路径。

返回类型

CLOB(10K)

返回列名（表函数）

returnedCLOB

示例

在此例中，所有部件元素都是从采购订单抽取的。

```
SELECT returnedCLOB as part
   from table(db2xml.extractCLOBs(db2xml.XMLFile('c:\dxx\samples\xml\getstart.xml'),
   '/Order/Part')) as x;
```

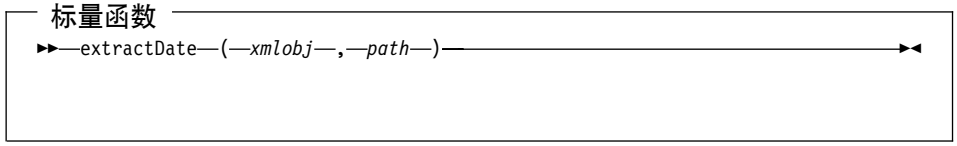
extractDate() 和 extractDates()

目的

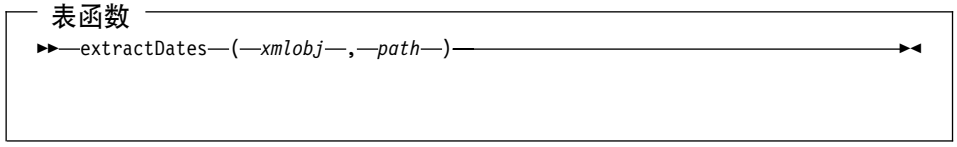
从 XML 文档抽取元素内容或属性值，并返回 DATE 类型的数据。

语法

标量函数



表函数



参数

表 36. extractDate 函数参数和 extractDates 函数参数

参数	数据类型	描述
xmlobj	XMLVARCHAR、XMLFILE 或 XMLCLOB	列名。
path	VARCHAR	元素或属性的位置路径。

返回类型

DATE

返回列名（表函数）

returnedDate

示例

下例中，ShipDate 值是以 DATE 的形式抽取的。

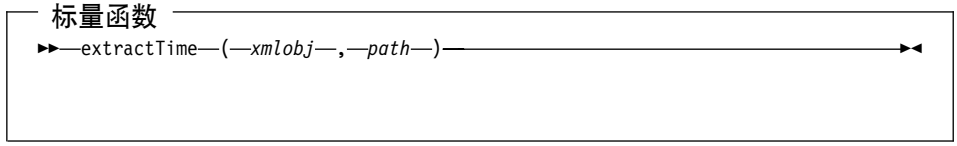
```
SELECT * from table(db2xml.extractDates(db2xml.XMLFile
   ('c:\dxx\samples\xml\getstart.xml'), '/Order/Part/Shipment/ShipDate')) as x;
```

extractTime() 和 extractTimes()

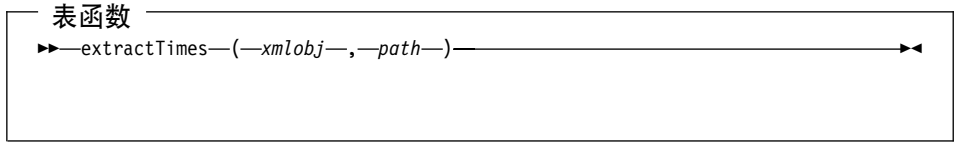
目的
从 XML 文档抽取元素内容或属性值，并返回 TIME 类型的数据。

语法

标量函数



表函数



参数

表 37. extractTime 函数参数和 extractTimes 函数参数

参数	数据类型	描述
xmlobj	XMLVARCHAR、XMLFILE 或 XMLCLOB	列名。
path	VARCHAR	元素或属性的位置路径。

返回类型
TIME

返回列名（表函数）
returnedTime

示例
此示例使用书籍样本文件。它搜索 XML 文件 `book.xml` 以找出书籍的定价时间并以 TIME 的形式返回该值。

XML 文件:

```
<?xml version="1.0">
<DOCTYPE book SYSTEM "c:\dxx\samples\book.dtd">
<book>
<chapter id="1" date="07/01/97">
<section>This is a section in Chapter One.</section>
<chapter id="2" date="01/02/1997">
<section>This is a section in Chapter Two.</section>
</chapter>
<price date="12/22/1998" time="11.12.13" timestamp="1998-12-22-11.12.13.888888">
38.281
</price>
</book>
```

抽取示例:

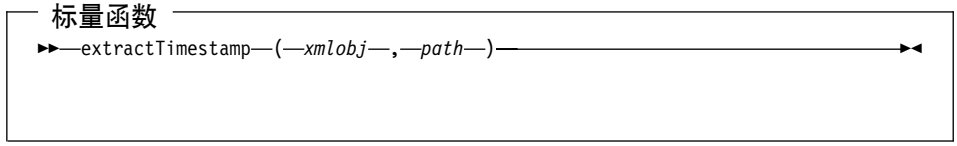
```
CREATE TABLE t1(SaleTime TIME);
INSERT INTO t1 values(db2xml.extractTime(doc, '/book/price/@time'));
SELECT * from t1;
```

extractTimestamp() 和 extractTimestamps()

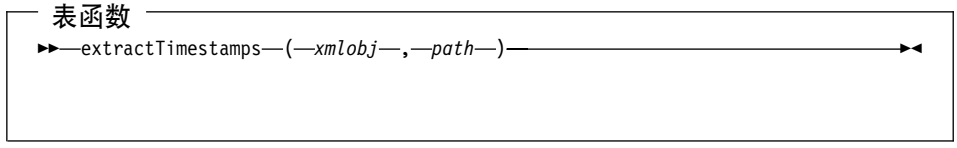
目的
从 XML 文档抽取元素内容或属性值，并返回 TIMESTAMP 类型的数据。

语法

标量函数



表函数



参数

表 38. extractTimestamp 函数参数和 extractTimestamps 函数参数

参数	数据类型	描述
xmlobj	XMLVARCHAR、XMLFILE 或 XMLCLOB	列名。
path	VARCHAR	元素或属性的位置路径。

返回类型
TIMESTAMP

返回列名（表函数）
returnedTimestamp

示例
此示例使用书籍样本文件。它搜索 XML 文件 `book.xml` 以找出指定对每本书进行定价的时间，并以 TIMESTAMP 的形式抽取该值。

XML 文件:

```
<?xml version="1.0">
<DOCTYPE book SYSTEM "c:\dxx\samples\book.dtd">
<book>
<chapter id="1" date="07/01/97">
<section>This is a section in Chapter One.</section>
<chapter id="2" date="01/02/1997">
<section>This is a section in Chapter Two.</section>
</chapter>
<price date="12/22/1998" time="11.12.13" timestamp="1998-12-22-11.12.13.888888">
38.281
</price>
</book>
```

抽取示例:

```
CREATE TABLE t1(SaleTime TIMESTAMP);
INSERT INTO t1 values(db2xml.extractTimestamp(doc, '/book/price/@timestamp'));
SELECT * from t1;
```

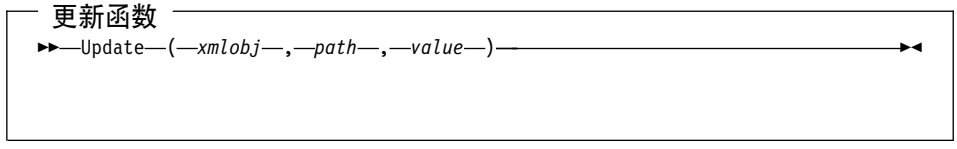
更新函数

Update() 函数更新指定的元素或属性值，该元素或值存储在 XML 列中的一个或多个 XML 文档中。您还可使用缺省强制转型函数来将 SQL 基本类型转换为 XML UDT，如第112页的『更新 XML 数据』中所述。

用途

采取列名 XML UDT、位置路径和更新值的字符串，并返回与第一个输入参数相同的 XML UDT。使用 Update() 函数，可指定要更新的元素或属性。

语法



参数

表 39. UDF Update 参数

参数	数据类型	描述
xmlobj	XMLVARCHAR，作为 LOCATOR 的 XMLCLOB	列名。
path	VARCHAR	元素或属性的位置路径。
value	VARCHAR	更新字符串。

重要事项：注意“更新”UDF 支持具有谓词的位置路径，但这些谓词只能带属性而不能带元素。例如，以下谓词受支持：

```
'/Order/Part[@color="black "]/ExtendedPrice'
```

以下谓词不受支持：

```
'/Order/Part/Shipment/[Shipdate < "11/25/00"]'
```

返回类型

数据类型	返回类型
XMLVARCHAR	XMLVARCHAR
作为 LOCATOR 的 XMLCLOB	XMLCLOB

示例

以下示例更新由销售员 Sriram Srinivasan 处理的采购订单。

```
UPDATE sales_tab
   set order = Update(order, '/Order/Customer/Name', 'IBM')
   WHERE sales_person = 'Sriram Srinivasan'
```

在此示例中， /Order/Customer/Name 的内容更新为 IBM。

用法

当您使用“更新”函数更改一个或多个 XML 文档中的值时，它实际上是在 XML 列之内替换 XML 文档。基于 XML 语法分析程序的输出，保留了原始文档的某些部分，而其他部分则丢失或被更改。下列章节描述如何处理文档，并提供文档在更新前后的样子的示例。

“更新”函数如何处理 XML 文档

当“更新”函数替换 XML 文档时，它必须基于 XML 语法分析程序的输出重新构造文档。表40用示例描述了处理文档各部分的方式。有关比较更新前后的 XML 文档的示例，参见第183页的『示例』

表 40. “更新”函数规则

项或节点类型	XML 文档代码示例	更新后的状态
XML 说明	<?xml version='1.0' encoding='utf-8' standalone='yes' >	保留 XML 说明: • 保留版本信息。 • 编码说明被保留并在原始文档中指定时出现。 • 独立说明被保留并在原始文档中指定时出现。 • 更新之后，使用单引号来描述值。

表 40. “更新”函数规则 (续)

项或节点类型	XML 文档代码示例	更新后的状态

DOCTYPE 说明	`<!DOCTYPE books SYSTEM`
	`  "http://dtds.org/books.dtd" >`
	`<!DOCTYPE books PUBLIC`
	`  "local.books.dtd"`
	`  "http://dtds.org/books.dtd" >`
	`<!DOCTYPE books>`
	`-Any of`
	`<!DOCTYPE books`
	`  ( S ExternalID ) ?`
	`  [ internal-dtd-subset ] >`
	`-Such as`
	`<!DOCTYPE books`
	`  [ <!ENTITY mydog "Spot"> ] >?`
	`  [ internal-dtd-subset ] >`
		保留文档类型说明:
		• 支持根元素名。
		• 公共和系统内部标识被保留并在原始文档中指定时出现。
		• “不”保留内部 DTD 子集。替换实体; 属性的缺省值被处理并在输出文档中出现。
		• 更新之后，使用双引号来描述公共和系统 URI 值。
		• 当前的 XML4c 语法分析程序不报告 XML 说明，该说明不包含外部标识或内部 DTD 子集。更新之后，此情况下的 DOCTYPE 说明将丢失。

处理指令	`<?xml-stylesheet`
	`  title="compact"`
	`  href="datatypes1.xsl"`
	`  type="text/xsl"?>`
		保留处理指令。

注释	`<!-- comment -->`	当注释在根元素中时被保留。 废弃根元素之外的注释。

元素	`<books>`
	content
	`</books>`
		保留元素。

属性	`id='1' date='01/02/1997"`	保留元素的属性。 • 更新之后，使用双引号来描述值。 • 属性中的数据被转义。 • 替换实体。
文本节点	`This chapter is about my dog &mydoc;.` `This chapter is about my dog Spot.`	保留文本节点（元素内容）。 • 文本节点中的数据被转义。 • 替换实体。

多次出现

当在 Update() UDF 中提供了位置路径时，使用所提供的值更新具有匹配路径的每个元素或属性的内容。这表示若文档具有多次出现的位置路径，则“更新”函数使用 *value* 参数中所提供的值替换现存的值。

您可以在 *path* 参数中指定谓词来提供相异的位置路径，以防止无意的更新。注意，“更新”UDF 支持具有谓词的位置路径，但这些谓词只能带属性而不能带元素。参见第180页的『参数』以了解详情。

示例

下列示例显示更新前后的 XML 文档的实例。

示例 1:

之前:

```
<?xml version='1.0'   encoding='utf-8'    standalone="yes"?>
<!DOCTYPE book PUBLIC "public.dtd" "system.dtd">
<?pitarget option1='value1' option2='value2'?>
<!-- comment -->
<book>
  <chapter id="1" date='07/01/1997'>
    <!-- first section -->
    <section>This is a section in Chapter One.</section>
  </chapter>
  <chapter id="2" date="01/02/1997">
    <section>This is a section in Chapter Two.</section>
    <footnote>A footnote in Chapter Two is here.</footnote>
  </chapter>
  <price date="12/22/1998" time="11.12.13"
       timestamp="1998-12-22-11.12.13.888888">38.281</price>
</book>
```

- 在 XML 说明中包含白空格
- 指定处理指令
- 在根节点外包含注释
- 指定 PUBLIC ExternalID
- 在根节点之内包含注释

之后:

```
<?xml version='1.0' encoding='utf-8' standalone='yes'?>
<!DOCTYPE book PUBLIC "public.dtd" "system.dtd">
<?pitarget option1='value1' option2='value2'?><book>
  <chapter id="1" date="07/01/1997">
        <!-- first section -->
        <section>This is a section in Chapter One.</section>
  </chapter>
  <chapter id="2" date="01/02/1997">
        <section>This is a section in Chapter Two.</section>
        <footnote>A footnote in Chapter Two is here.</footnote>
  </chapter>
  <price date="12/22/1998" time="11.12.13"
       timestamp="1998-12-22-11.12.13.888888">60.02</price>
</book>
```

- 消除标记中的白空格
- 保留处理指令
- 不保留根节点之外的注释
- 保留 PUBLIC ExternalID
- 保留根节点之内的注释
- 更改的值是 <price> 元素的值

示例 2:

之前:

```
<?xml version='1.0'     ?>
<!DOCTYPE book>
<!-- comment -->
<book>
  ...
</book>
```

包含 DOCTYPE 说明，而无 ExternalID 或内部 DTD 子集。不受支持。

之后:

```
<?xml version='1.0'?>
<book>
  ...
</book>
```

XML 语法分析程序不报告 DOCTYPE 说明，且不保留它。

示例 3:

之前:

```
<?xml version='1.0'     ?>
<!DOCTYPE book [ <!ENTITY myDog "Spot"> ]>
<!-- comment -->
<book>
  <chapter id="1" date='07/01/1997'>
       <!-- first section -->
       <section>This is a section in Chapter
         One about my dog &;myDog;.</section>
       ...
  </chapter>
  ...
</book>
```

- 在标记中包含白空格
- 指定内部 DTD 子集
- 指定文本节点中的实体

之后:

```
<?xml version='1.0'?>
<!DOCTYPE book>
<book>
  <chapter id="1" date="07/01/1997">
       <!-- first section -->
       <section>This is a section in Chapter
         One about my dog Spot.</section>
          ...
  </chapter>
  ...
</book>
```

- 消除标记中的白空格
- 不保留内部 DTD 子集
- 解析并替换文本节点中的实体

第10章 XML Extender 存储过程

XML Extender 提供了存储过程以对 XML 列和集合进行管理。可以从 DB2 客户机调用这些存储过程。可以将客户机接口嵌入 SQL、ODBC 或 JDBC。有关如何调用存储过程的详情，参考 *DB2 通用数据库管理指南*中有关存储过程的章节。

存储过程使用模式 db2xml，它是 XML Extender 的模式名

XML Extender 提供了三种类型的存储过程:

- 第187页的『管理存储过程』，它帮助用户完成管理任务
- 第194页的『组合存储过程』，它使用现存数据库表中的数据来生成 XML 文档
- 第203页的『分解存储过程』，它拆散或分割入局 XML 文档，并将数据存储在新的或现存的数据库表中

XML 集合存储过程所使用的参数限制在第257页的『附录D. XML Extender 限制』中进行介绍。

指定包含文件

确保在调用存储过程的程序中，包括了 XML Extender 外部头文件。该头文件位于 `DXX_INSTALL/include` 目录中。 `DXX_INSTALL` 是 XML Extender 的安装目录。该目录与操作系统有关。头文件是:

dxx.h XML Extender 定义的常量和数据类型

dxxrc.h XML Extender 返回码

包括这些头文件的语法是:

```
#include "dxx.h"
#include "dxxrc.h"
```

确保在 makefile 中用编译选项指定了包含文件的路径。

调用 XML Extender 存储过程

通常，使用以下语法来调用 XML Extender:

```
CALL db2xml.function_entry_point
```

其中:

© Copyright IBM Corp. 1999, 2000

db2xml

指定 XML Extender 存储过程的库。在 UNIX 操作系统，该库存储在 sqllib/function 目录。在 Windows 操作系统，该库存储在 *DXX_INSTALL*/bin 目录。

function_entry_point

指定传送给存储过程的自变量。

在 CALL 语句中，传送到存储过程的自变量必须是主变量，而不能是常量或表达式。主变量可以具有空指示符。查看 DXX_INSTALL/samples/c 和 DXX_INSTALL/samples/cli 目录中有关调用存储过程的样本，也可在本书的下列各节中查看调用存储过程的样本: 第36页的『组合 XML 文档』和第121页的『第6章 管理 XML 集合数据』。

在 DXX_INSTALL/samples/c 目录中，提供了 SQC 代码文件，以便使用嵌入式 SQL 来调用 XML 集合存储过程。在 DXX_INSTALL/samples/cli 目录中，样本文件显示了如何使用调用层接口 (CLI) 来调用存储过程。

增加 CLOB 限制

CLOB 参数在被传送至存储过程时的缺省限制为 1 MB。可以通过完成下列步骤来增加此限制:

1. 删除每个存储过程。例如:

```
db2   "drop procedure db2xml.dxxShredXML"
```

2. 以增加的 CLOB 限制创建新的过程。例如:

```
db2   "create procedure db2xml.dxxShredXML(in     dadBuf      clob(100K),
                                           in     XMLObj      clob(10M),
                                           out    returnCode  integer,
                                           out    returnMsg   varchar(1024)
                                           )
       external name 'db2xml.dxxShredXML'
       language C
       parameter style DB2DARI
       not deterministic
       fenced
       null call;
```

开始之前

利用 XML Extender 存储过程和 DB2 CLI 绑定文件来绑定数据库。可以使用样本命令文件 getstart_prep.cmd 来绑定文件。此命令文件位于 DXX_INSTALL/samples/cmd 目录中。

1. 与数据库连接。例如:

```
db2 "connect to SALES_DB"
```

2. 更改为 DXX_INSTALL/bnd 目录，并将 XML Extender 与数据库绑定。

```
db2 "bind @dxxbind.lst"
```

3. 更改为 sqllib/bnd 目录，并将 CLI 与数据库绑定。

```
db2 "bind @db2cli.lst"
```

4. 终止连接。

```
db2 "terminate"
```

管理存储过程

这些存储过程用于以下管理任务: 例如，启用或禁用 XML 列或集合。 它们是通过 XML Extender 管理向导和管理命令 **dxxadm** 来调用的。

dxxEnableDB()

目的

启用数据库。当启用了数据库时，XML Extender 将创建下列对象:

- XML Extender 用户定义类型 (UDT)。
- XML Extender 用户定义函数 (UDF)。
- XML Extender DTD 参考表 DTD_REF，它用来存储 DTD 以及有关每个 DTD 的信息。有关 DTD_REF 表的完整描述，参见第209页的『DTD 参考表』。
- XML Extender 用法表 XML_USAGE，该表存储了为 XML 和每个集合启用的每一列的公共信息。 有关 XML_USAGE 表的完整描述，参见第210页的『XML 用法表』。

```
dxxEnableDB(char(dbName) dbName,       /* input */
            long           returnCode,       /* output */
            varchar(1024)  returnMsg)        /* output */
```

参数

表 41. dxxEnableDB() 参数

参数	描述	IN/OUT 参数
dbName	数据库名。	IN
returnCode	存储过程的返回码。	OUT
returnMsg	在发生错误时返回的信息文本。	OUT

dxxDisableDB()

目的

禁用数据库。当 XML Extender 禁用数据库时，它将删除下列对象:

- XML Extender 用户定义类型 (UDT)。
- XML Extender 用户定义函数 (UDF)。
- XML Extender DTD 参考表 DTD_REF，它用来存储 DTD 以及有关每个 DTD 的信息。有关 DTD_REF 表的完整描述，参见第209页的『DTD 参考表』。
- XML Extender 用法表 XML_USAGE，该表存储了为 XML 和每个集合启用的每一列的公共信息。 有关 XML_USAGE 表的完整描述，参见第210页的『XML 用法表』。

重要事项: 在试图禁用数据库之前，必须禁用所有 XML 列。XML Extender 不能禁用其中包含了为 XML 启用的列或集合的数据库。

```
dxxDisableDB(char(dbName)     dbName,       /* input */
                 long           returnCode,   /* output */
                 varchar(1024) returnMsg)    /* output */
```

参数

表 42. dxxDisableDB() 参数

参数	描述	IN/OUT 参数
dbName	数据库名。	IN
returnCode	存储过程的返回码。	OUT
returnMsg	在发生错误时返回的信息文本。	OUT

dxxEnableColumn()

目的

启用 XML 列。当启用列时，XML Extender 将完成下列任务:

- 确定 XML 表是否具有主键；若没有，则 XML Extender 将改变 XML 表，并添加称为 DXXROOT_ID 的一列。
- 创建在 DAD 文件中指定的辅助表，并且有一列包含 XML 表中每一行的唯一标识符。 此列是由用户指定的 *root_id*，或是由 XML Extender 命名的 DXXROOT_ID。
- 为 XML 表及其辅助表创建缺省视图，可选择使用您指定的名称。

```
dxxEnableColumn(char(dbName) dbName,      /* input */
                char(tbName) tbName,      /* input */
                char(colName) colName,     /* input */
                CLOB(100K)     DAD,         /* input */
                char(tablespace) tablespace,  /* input */
                char(defaultView) defaultView, /* input */
                char(rootID) rootID,      /* input */
                long           returnCode,  /* output */
                varchar(1024)  returnMsg)   /* output */
```

参数

表 43. dxxEnableColumn() 参数

参数	描述	IN/OUT 参数
dbName	数据库名。	IN
tbName	包含 XML 列的表的名称。	IN
colName	XML 列的名称。	IN
DAD	包含 DAD 文件的 CLOB。	IN
tablespace	包含辅助表的表空间而不是缺省表空间。如果未指定，则使用缺省表空间。	IN
defaultView	用来连接应用程序表和辅助表的缺省视图的名称。	IN
rootID	应用程序表中要用作辅助表的 *root_id* 的单个主键的名称。	IN
returnCode	存储过程的返回码。	OUT
returnMsg	在发生错误时返回的信息文本。	OUT

dxxDisableColumn()

目的

禁用已经启用了 XML 的列。当禁用 XML 列时，该列就不能再包含 XML 数据类型。

```
dxxDisableColumn(char(dbName) dbName,      /* input */
                 char(tbName) tbName,      /* input */
                 char(colName) colName,     /* input */
                 long           returnCode,  /* output */
                 varchar(1024)  returnMsg)   /* output */
```

参数

表 44. dxxDisableColumn() 参数

参数	描述	IN/OUT 参数
dbName	数据库名。	IN
tbName	包含 XML 列的表的名称。	IN
colName	XML 列的名称。	IN
returnCode	存储过程的返回码。	OUT
returnMsg	在发生错误时返回的信息文本。	OUT

dxxEnableCollection()

目的

启用与应用程序表相关联的 XML 集合。

```
dxxEnableCollection(char(dbName) dbName,      /* input */
                    char(colName) colName,     /* input */
                    CLOB(100K)     DAD,         /* input */
                    char(tablespace) tablespace,  /* input */
                    long           returnCode,  /* output */
                    varchar(1024)  returnMsg)   /* output */
```

参数

表 45. dxxEnableCollection() 参数

参数	描述	IN/OUT 参数
dbName	数据库名。	IN
colName	XML 集合的名称。	IN
DAD	包含 DAD 文件的 CLOB。	IN
tablespace	包含辅助表的表空间而不是缺省表空间。如果未指定，则使用缺省表空间。	IN
returnCode	存储过程的返回码。	OUT
returnMsg	在发生错误时返回的信息文本。	OUT

dxxDisableCollection()

目的

禁用已启用了 XML 的集合，除去用来标识作为集合的一部分的表和列的标记。

```
dxxDisableCollection(char(dbName) dbName,      /* input */
                     char(colName) colName,     /* input */
                     long          returnCode,  /* output */
                     varchar(1024)  returnMsg)   /* output */
```

参数

表 46. dxxDisableCollection() 参数

参数	描述	IN/OUT 参数
dbName	数据库名。	IN
colName	XML 集合的名称。	IN
returnCode	存储过程的返回码。	OUT
returnMsg	在发生错误时返回的信息文本。	OUT

组合存储过程

组合存储过程 dxxGenXML() 和 dxxRetrieveXML() 被用于通过使用现存数据库表中的数据来生成 XML 文档。 dxxGenXML() 存储过程将 DAD 文件作为输入；它不需要已启用的 XML 集合。 dxxRetrieveXML() 存储过程将已启用的 XML 集合名作为输入。

dxxGenXML()

目的

通过使用存储在由 DAD 文件中的 <Xcollection> 指定的 XML 集合表中的数据来构造 XML 文档，并将每个 XML 文档当作一行插入结果表中。 还可以打开结果表上的游标，并取装结果集。

为了提供灵活性，dxxGenXML() 还允许用户指定要在结果表中生成的最大行数。这可以减少在任何试验进程期间，应用程序必须等待获得结果的所需时间量。该存储过程将返回表中的实际行数，以及任何错误消息，包括错误代码和错误消息。

为了支持动态查询，dxxGenXML() 采用了输入参数 *override*。根据输入 *overrideType*，应用程序可以覆盖 SQL 映射的 SQL_stmt，或者覆盖 DAD 文件中 RDB_node 映射的 RDB_node 中的条件。 输入参数 *overrideType* 被用来区分 *override* 的类型。 有关 *override* 参数的详情，参见第125页的『动态覆盖 DAD 文件中的值』。

```
dxxGenXML(CLOB(100K)     DAD,             /* input */
          char(resultTabName) resultTabName,   /* input */
          integer        overrideType     /* input */
          varchar(1024)  override,        /* input */
          integer        maxRows,         /* input */
          integer        numRows,         /* output */
          long           returnCode,      /* output */
          varchar(1024)  returnMsg)       /* output */
```

参数

表 47. dxxGenXML() 参数

参数	描述	IN/OUT 参数
DAD	包含 DAD 文件的 CLOB。	IN
resultTabName	结果表的名称，在调用之前结果表就应该存在。该表仅包含一列，其类型为 XMLVARCHAR 或 XMLCLOB。	IN

表 47. dxxGenXML() 参数 (续)

参数	描述	IN/OUT 参数
overrideType	一个标志，用来指示下列 *override* 参数的类型: • **NO_OVERRIDE**: 不覆盖。 • **SQL_OVERRIDE**: 被 SQL_stmt 覆盖。 • **XML_OVERRIDE**: 被基于 XPath 的条件覆盖。	IN
override	覆盖 DAD 文件中的条件。输入值基于 *overrideType*。 • **NO_OVERRIDE**: NULL 字符串。 • **SQL_OVERRIDE**: 有效的 SQL 语句。使用此 *overrideType* 时，必需在 DAD 文件中使用 SQL 映射。输入的 SQL 语句将覆盖 DAD 文件中的 SQL_stmt。 • **XML_OVERRIDE**: 用双引号引起来的一个字符串，该字符串包含一个或多个表达式，各表达式之间用 ″AND″ 隔开。使用此 *overrideType* 时，要求在 DAD 文件中使用 RDB_node 映射。	IN
maxRows	结果表中的最大行数。	IN
numRows	结果表中生成的实际行数。	OUT
returnCode	存储过程的返回码。	OUT
returnMsg	在发生错误时返回的信息文本。	OUT

例

以下示例假定: 结果表是用 XML_ORDER_TAB 的名称来创建的，并且该表中有一列的类型是 XMLVARCHAR。

```
#include "dxx.h"
#include "dxxrc.h"

              EXEC SQL INCLUDE SQLCA;
              EXEC SQL BEGIN DECLARE SECTION;
              SQL TYPE is CLOB(100K)  dad;              /* DAD */
              SQL TYPE is CLOB_FILE   dadFile;          /* dad file */
              char                    result_tab[32];   /* name of the result table */
              char                    override[2];      /* override, will set to NULL*/
              short                   overrideType;     /* defined in dxx.h */
              short                   max_row;          /* maximum number of rows */
              short                   num_row;          /* actual number of rows */
              long                    returnCode;       /* return error code */
              char                    returnMsg[1024];  /* error message text */
              short                   dad_ind;
              short                   rtab_ind;
              short                   ovtype_ind;
              short                   ov_inde;
              short                   maxrow_ind;
              short                   numrow_ind;
              short                   returnCode_ind;
              short                   returnMsg_ind;

              EXEC SQL END DECLARE SECTION;

              /* create table */
              EXEC CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

              /* read data from a file to a CLOB */
              strcpy(dadfile.name,"e:\dxx\dad\litem3.dad");
              dadfile.name_length = strlen("e:\dxx\dad\litem3.dad");
              dadfile.file_options = SQL_FILE_READ;
              EXEC SQL VALUES (:dadfile) INTO :dad;
              strcpy(result_tab,"xml_order_tab");
              override[0] = '\0';
              overrideType = NO_OVERRIDE;
              max_row = 500;
              num_row = 0;
              returnCode = 0;
              msg_txt[0] = '\0';
              collection_ind = 0;
              dad_ind = 0;
              rtab_ind = 0;
              ov_ind = -1;
              ovtype_ind = 0;
              maxrow_ind = 0;
              numrow_ind = -1;
              returnCode_ind = -1;
              returnMsg_ind = -1;

              /* Call the store procedure */
              EXEC SQL CALL dxxGenXML(:dad:dad_ind;
                     :result_tab:rtab_ind,
```

```
:overrideType:ovtype_ind,:override:ov_ind,
:max_row:maxrow_ind,:num_row:numrow_ind,
:returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

dxxRetrieveXML()

目的

允许将同一 DAD 文件用于组合和分解。存储过程 dxxRetrieveXML() 还用作检索分解的 XML 文档的方法。 dxxRetrieveXML() 将包含了 DAD 文件的缓冲区、创建的结果表的名称以及要返回的最大行数作为其输入。 它返回结果表的结果集、结果集中的实际行数、错误代码和信息文本。

为了支持动态查询，dxxRetrieveXML() 采用了输入参数 *override*。根据输入 *overrideType*，应用程序可以覆盖 SQL 映射的 SQL_stmt，或者覆盖 DAD 文件中 RDB_node 映射的 RDB_node 中的条件。 输入参数 *overrideType* 被用来区分 *override* 的类型。 有关 *override* 参数的详情，参见第125页的『动态覆盖 DAD 文件中的值』。

dxxRetrieveXML() 的 DAD 文件的需求与 dxxGenXML() 的需求是相同的。 唯一的区别就是 DAD 不是 dxxRetrieveXML() 的输入参数，但它是启用的 XML 集合的名称。

```
dxxRetrieveXML(char(collectionName) collectionName,   /* input */
               char(resultTabName) resultTabName,    /* input */
               integer        overrideType,     /* input */
               varchar(1024)  override,         /* input */
               integer        maxRows,          /* input */
               integer        numRows,          /* output */
               long           returnCode,       /* output */
               varchar(1024)  returnMsg)        /* output */
```

参数

表 48. dxxRetrieveXML() 参数

参数	描述	IN/OUT 参数
collectionName	启用的 XML 集合的名称。	IN
resultTabName	结果表的名称，在调用之前结果表就应该存在。该表仅包含一列，其类型为 XMLVARCHAR 或 XMLCLOB。	IN

表 48. dxxRetrieveXML() 参数 (续)

参数	描述	IN/OUT 参数
overrideType	一个标志，用来指示下列 *override* 参数的类型: • **NO_OVERRIDE**: 不覆盖。 • **SQL_OVERRIDE**: 被 SQL_stmt 覆盖。 • **XML_OVERRIDE**: 被基于 XPath 的条件覆盖。	IN
override	覆盖 DAD 文件中的条件。输入值基于 *overrideType*。 • **NO_OVERRIDE**: NULL 字符串。 • **SQL_OVERRIDE**: 有效的 SQL 语句。使用此 *overrideType* 时，必需在 DAD 文件中使用 SQL 映射。输入的 SQL 语句将覆盖 DAD 文件中的 SQL_stmt。 • **XML_OVERRIDE**: 用双引号引起来的一个字符串，该字符串包含一个或多个表达式，各表达式之间用 ″AND″ 隔开。使用此 *overrideType* 时，要求在 DAD 文件中使用 RDB_node 映射。	IN
maxRows	结果表中的最大行数。	IN
numRows	结果表中生成的实际行数。	OUT
returnCode	存储过程的返回码。	OUT
returnMsg	在发生错误时返回的信息文本。	OUT

例

以下是调用 dxxRetrieveXML() 的一个示例。在此示例中，结果表是用 XML_ORDER_TAB 的名称来创建的，并且结果表中有一列的类型是 XMLVARCHAR。

```
#include "dxx.h"
#include "dxxrc.h"

            EXEC SQL INCLUDE SQLCA;
            EXEC SQL BEGIN DECLARE SECTION;
              char    collection[32];    /* dad buffer */
              char    result_tab[32];    /* name of the result table */
              char    override[2];       /* override, will set to NULL*/
              short   overrideType;      /* defined in dxx.h */
              short   max_row;           /* maximum number of rows */
              short   num_row;           /* actual number of rows */
              long    returnCode;        /* return error code */
              char    returnMsg[1024];   /* error message text */
              short   dadbuf_ind;
              short   rtab_ind;
              short   ovtype_ind;
              short   ov_inde;
              short   maxrow_ind;
              short   numrow_ind;
              short   returnCode_ind;
              short   returnMsg_ind;

              EXEC SQL END DECLARE SECTION;

              /* create table */
              EXEC SQL CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

              /* initialize host variable and indicators */
              strcpy(collection,"sales_ord");
              strcpy(result_tab,"xml_order_tab");
              override[0] = '\0';
              overrideType = NO_OVERRIDE;
              max_row = 500;
              num_row = 0;
              returnCode = 0;
              msg_txt[0] = '\0';
              collection_ind = 0;
              rtab_ind = 0;
              ov_ind = -1;
              ovtype_ind = 0;
              maxrow_ind = 0;
              numrow_ind = -1;
              returnCode_ind = -1;
              returnMsg_ind = -1;

              /* Call the store procedure */
              EXEC SQL CALL dxxRetrieve(:collection:collection_ind;
                     :result_tab:rtab_ind,
```

```
:overrideType:ovtype_ind,:override:ov_ind,
:max_row:maxrow_ind,:num_row:numrow_ind,
:returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

分解存储过程

分解存储过程 dxxInsertXML() 和 dxxShredXML() 是用来拆散或分割入局 XML 文档，并将数据存储在新的或现存的数据库表中。 dxxInsertXML() 存储过程将启用的 XML 集合名作为输入。 dxxShredXML() 存储过程将 DAD 文件作为输入；它不需要已启用的 XML 集合。

dxxShredXML()

目的

dxxShredXML() 存储过程是 dxxGenXML() 的配对存储过程。为了使 dxxShredXML() 起作用，在 DAD 文件中指定的所有表必须存在，在 DAD 中指定的所有列及其数据类型必须与现存表一致。存储过程 dxxShredXML() 不需要连接表之间的主键与外键关系，它是在启用集合进程期间由 XML Extender 创建的。但是，这些表中必须存在根节点 element_node 的 RDB_node 中指定的连接条件列。

```
dxxShredXML(CLOB(100K)    DAD,           /* input */
            CLOB(1M)      xmlobj,        /* input */
            long          returnCode,    /* output */
            varchar(1024) returnMsg)     /* output */
```

参数

表 49. dxxShredXML() 参数

参数	描述	IN/OUT 参数
DAD	包含 DAD 文件的 CLOB。	IN
xmlobj	XMLCLOB 类型的 XML 文档对象。	IN
returnCode	存储过程的返回码。	OUT
returnMsg	在发生错误时返回的信息文本。	OUT

例

以下是调用 dxxShredXML() 的一个示例。

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
            EXEC SQL BEGIN DECLARE SECTION;
              SQL TYPE is CLOB dad;           /* DAD*/
              SQL TYPE is CLOB_FILE  dadFile; /* DAD file*/
              SQL TYPE is CLOB xmlDoc;        /* input XML document */
              SQL TYPE is CLOB_FILE xmlFile;  /* input XMLfile */
              long        returnCode;         /* error code */
              char        returnMsg[1024];    /* error message text */
              short       dad_ind;
              short       xmlDoc_ind;
              short       returnCode_ind;
              short       returnMsg_ind;
              EXEC SQL END DECLARE SECTION;

              /* initialize host variable and indicators */
              strcpy(dadFile.name,"c:\dxx\samples\dad\getstart_xcollection.dad");
```

```
dadFile.name_length=strlen("c:\dxx\samples\dad\getstart_xcollection.dad");
dadFile.file_option=SQL_FILE_READ;
strcpy(xmlFile.name,"c:\dxx\samples\cmd\getstart.xml");
xmlFile.name_length=strlen("c:\dxx\samples\cmd\getstart.xml");
xmlFile.file_option=SQL_FILE_READ;
SQL EXEC VALUES (:dadFile) INTO :dad;
SQL EXEC VALUES (:xmlFile) INTO :xmlDoc;
returnCode = 0;
returnMsg[0] = '\0';
dad_ind = 0;
xmlDoc_ind = 0;
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL db2xml.dxxShredXML(:dad:dad_ind;
                :xmlDoc:xmlDoc_ind,
                :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

dxxInsertXML()

目的

采用两个输入参数: 已启用的 XML 集合的名称以及要分解的 XML 文档, 并返回两个输出参数: 返回码和返回信息。

```
dxxInsertXML(char(collectionName) collectionName,   /* input */
             CLOB(1M)       xmlobj,           /* input */
             long           returnCode,       /* output */
             varchar(1024)  returnMsg)        /* output */
```

参数

表 50. dxxInsertXML() 参数

参数	描述	IN/OUT 参数
collectionName	启用的 XML 集合的名称。	IN
xmlobj	CLOB 类型的 XML 文档对象。	IN
returnCode	存储过程的返回码。	OUT
returnMsg	在发生错误时返回的信息文本。	OUT

例

在以下示例中, dxxInsertXML() 调用将分解输入 XML 文档 e:\xml\order1.xml, 并根据 DAD 文件中所指定的映射将数据插入 SALES_ORDER 集合表中, 就是利用该 DAD 文件来启用调用的。

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
            EXEC SQL BEGIN DECLARE SECTION;
              char                  collection[64];  /* name of an XML collection */
              SQL TYPE is CLOB_FILE xmlDoc;          /* input XML document */
              long                  returnCode;      /* error code */
              char                  returnMsg[1024]; /* error message text */
              short                 collection_ind;
              short                 xmlDoc_ind;
              short                 returnCode_ind;
              short                 returnMsg_ind;
              EXEC SQL END DECLARE SECTION;

              /* initialize host variable and indicators */
              strcpy(collection,"sales_ord")
              strcpy(xmlobj.name,"c:\dxx\samples\cmd\getstart.xml");
              xmlobj.name_length=strlen("c:\dxx\samples\cmd\getstart.xml");
              xmlobj.file_option=SQL_FILE_READ;
              returnCode = 0;
```

```
returnMsg[0] = '\0';
collection_ind = 0;
xmlobj_ind = 0;
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL db2xml.dxxInsertXML(:collection:collection_ind;
                :xmlobj:xmlobj_ind,
                :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

第11章 管理支持表

当启用数据库时，就创建了 DTD 参考表 DTD_REF 和 XML_USAGE 表。DTD_REF 表中包含关于所有 DTD 的信息。XML_USAGE 表存储了每个启用了 XML 的列的公共信息。

在支持表中列示的参数限制在第257页的『附录D. XML Extender 限制』中进行介绍。

DTD 参考表

XML Extender 还用作 XML DTD 库。当数据库启用了 XML 时，就创建了 DTD 参考表 DTD_REF。 此表的每一行都表示具有附加元数据信息的 DTD。 用户可以访问此表，并插入他们自己的 DTD。 DTD_REF 表中的 DTD 可用来验证 XML 文档以及帮助应用程序定义 DAD 文件。 它具有模式名 db2xml。 DTD_REF 表中可以具有表51中所显示的列。

表 51. DTD_REF 表

列名	数据类型	描述
DTDID	VARCHAR(128)	主键（是唯一的，且不能为空）。它用来标识 DTD。当在 DAD 文件中指定了主键时，DAD 文件必须遵循 DTD 所定义的模式。
CONTENT	XMLCLOB	DTD 的内容。
USAGE_COUNT	INTEGER	数据库中使用 DTD 来定义它们的 DAD 文件的 XML 列数和 XML 集合数。
AUTHOR	VARCHAR(128)	DTD 的作者，供用户输入的可选信息。
CREATOR	VARCHAR(128)	进行首次插入的用户标识。 CREATOR 列是可选的。
UPDATOR	VARCHAR(128)	进行最后更新的用户标识。 UPDATOR 列是可选的。

限制: 仅当 USAGE_COUNT 为零时，才能由应用程序来修改 DTD。

© Copyright IBM Corp. 1999, 2000

XML 用法表

存储每个启用了 XML 的列的公共信息。 XML_USAGE 表的模式名是 db2xml, 其主键为 (*table_name*、*col_name*)。 XML_USAGE 表是在使用表52中所列示的列来启用数据库时创建的。

表 52. XML_USAGE 表

列名	描述
table_schema	对于 XML 列，它是包含了 XML 列的用户表的模式名。对于 XML 集合，它是值 ″DXX_COLL″，作为缺省模式名。
table_name	对于 XML 列，它是包含了 XML 列的用户表的名称。对于 XML 集合，它是值 ″DXX_COLLECTION″，该值将实体标识为集合。
col_name	XML 列或 XML 集合的名称。它是组合键和 table_name 的一部分。
DTDID	DTD_REF 表中用来定义 DAD 文件的 DTD 的标识。它是外键。
DAD	与列相关联的 DAD 文件的内容。
default_view	用来存储缺省视图名（如果有一个缺省视图名的话）。
trigger_suffix	不能为空。表示唯一的触发器名。
验证	1 表示 “是”，0 表示 “否”。
access_mode	1 表示 XML 集合，0 表示 XML 列

限制: 仅当 USAGE_COUNT 为零时，才能由应用程序来修改 DTD。

第12章 诊断信息

程序中所有的嵌入式 SQL 语句和 DB2 命令行接口 (CLI) 调用（包括调用 DB2 XML Extender 用户定义函数 (UDF)）都会生成一些代码，这些代码指示是否成功地执行了嵌入式 SQL 语句或 DB2 CLI 调用。

程序可检索补充这些代码的信息。包括 SQLSTATE 信息和错误消息。可使用此诊断信息来排除和修正程序中的问题。

有时无法轻易地诊断出问题的来源。 在这些情况下，可能需要提供信息给您的软件支持供应商，以确定并修正问题。 XML Extender 包括记录 XML Extender 活动的跟踪功能。跟踪信息可能对 “IBM 软件支持” 来说很有价值。 您只应在 “IBM 软件支持” 的指导下才使用跟踪功能。

本章描述如何访问此诊断信息。它描述了:

- 如何处理 XML Extender UDF 返回码。
- 如何控制跟踪

它还列示并描述了 XML Extender 可能返回的 SQLSTATE 代码和错误消息。

处理 UDF 返回码

嵌入式 SQL 语句返回 SQLCA 结构的 SQLCODE、SQLWARN 和 SQLSTATE 字段中的代码。此结构是在 SQLCA INCLUDE 文件中定义的。（有关 SQLCA 结构和 SQLCA INCLUDE 文件的详情，参见 *DB2 Application Development Guide*。）

DB2 CLI 调用返回可使用 SQLError 函数进行检索的 SQLCODE 值和 SQLSTATE 值。（有关使用 SQLError 函数检索错误消息的详情，参见 *CLI Guide and Reference*。）

SQLCODE 值 0 意味着语句运行成功（可能会有警告条件）。正的 SQLCODE 值表示语句运行成功，但带有警告信息。（嵌入式 SQL 语句返回有关警告信息，该警告与 SQLWARN 字段中的 0 或正 SQLCODE 值相关联。） 负 SQLCODE 值表示发生了错误。

DB2 将消息与每一个 SQLCODE 值相关。如果 XML Extender UDF 遇到警告或错误条件，它会将相关信息传送至 DB2 以包含在 SQLCODE 消息中。

© Copyright IBM Corp. 1999, 2000

SQLSTATE 值包含补充 SQLCODE 消息的代码。参见『SQLSTATE 代码』以获取 XML Extender 返回的每个 SQLSTATE 代码的说明。

调用 DB2 XML Extender UDF 的嵌入式 SQL 语句和 DB2 CLI 调用可能返回 SQLCODE 消息和 SQLSTATE 值（这些值对于这些 UDF 是唯一的），但 DB2 返回这些值的方式与它对其他嵌入式 SQL 语句或其他 DB2 CLI 调用返回值的方式相同。 因此，访问这些值的方式与用于不启动 DB2 XML Extender UDF 的嵌入式 SQL 语句或 DB2 CLI 调用的方式相同。

参见『SQLSTATE 代码』以获取 SQLSTATE 值和可由 XML Extender 返回的相关信息的消息号。参见第216页的『消息』以获取有关每个信息的资料。

处理存储过程返回码

XML Extender 提供返回码以帮助解决存储过程的问题。当从存储过程接收到返回码时，应检查以下文件，该文件与带有 XML Extender 错误消息号和符号常量的返回码匹配。

```
DXX_INSTALL/include/dxxrc.h
```

可引用第216页的『消息』中的错误消息号，并使用说明中的诊断信息。

SQLSTATE 代码

表53列示并描述 XML Extender 返回的 SQLSTATE 值。每个 SQLSTATE 值的说明都包括其符号表示法。该表还列示了与每个 SQLSTATE 值相关的消息号。参见第216页的『消息』以获取有关每个信息的信息。

表 53. SQLSTATE 代码和相关联的消息号

SQLSTATE	消息号	描述
00000	DXX*nnnn*I	未发生错误。
01HX0	DXXD003W	在路径表达式中指定的元素或属性已从 XML 文档中丢失。
38X00	DXXC000E	XML Extender 无法打开指定的文件。
38X01	DXXA072E	XML Extender 尝试在启用数据库之前自动绑定该数据库，但未能找到绑定文件
	DXXC001E	XML Extender 找不到指定的文件。
38X02	DXXC002E	XML Extender 无法读取指定的文件中的数据。

表 53. SQLSTATE 代码和相关联的消息号 (续)

SQLSTATE	消息号	描述
38X03	DXXC003E	XML Extender 无法将数据写入文件中。
	DXXC011E	XML Extender 无法将数据写入跟踪控制文件。
38X04	DXXC004E	XML Extender 无法运行指定的定位器。
38X05	DXXC005E	文件大小大于 XMLVarchar 的大小，XML Extender 无法导入文件中的所有数据。
38X06	DXXC006E	文件大小大于 XMLCLOB 的大小，XML Extender 无法导入文件中的所有数据。
38X07	DXXC007E	“LOB 定位器”中的字节数不等于文件大小。
38X08	DXXD001E	标量抽取函数所使用的位置路径多次出现。标量函数仅可使用不多次出现的位置路径。
38X09	DXXD002E	路径表达式在语法上不正确。
38X10	DXXG002E	XML Extender 无法从操作系统分配内存。
38X11	DXXA009E	此存储过程仅用于“XML 列”。
38X12	DXXA010E	在试图启用一列时，XML Extender 找不到 DTDID，它是在文档访问定义 (DAD) 文件中为 DTD 指定的标识符。
38X14	DXXD000E	试图将无效文档存储到表中。验证已失败。
38X15	DXXA056E	文档访问定义 (DAD) 文件中的验证元素错误或已丢失。
	DXXA057E	文档访问定义 (DAD) 文件中的辅助表的名称属性错误或已丢失。
	DXXA058E	文档访问定义 (DAD) 文件中的列的名称属性错误或已丢失。
	DXXA059E	文档访问定义 (DAD) 文件中的列的类型属性错误或已丢失。

表 53. SQLSTATE 代码和相关联的消息号 (续)

SQLSTATE	消息号	描述
	DXXA060E	文档访问定义 (DAD) 文件中列的路径属性错误或已丢失。
	DXXA061E	文档访问定义 (DAD) 文件中的列的 multi_occurrence 属性错误或已丢失。
	DXXQ000E	必要的元素已从文档访问定义 (DAD) 文件丢失。
38X16	DXXG004E	所需参数的空值被传送至 XML 存储过程。
38X17	DXXQ001E	文档访问定义 (DAD) 中的 SQL 语句或覆盖它的某语句是无效的。SELECT 语句对于生成 XML 文档是必需的。
38X18	DXXG001E	XML Extender 遇到内部错误。
	DXXG006E	使用 CLI 时，XML Extender 遇到了内部错误。
38X19	DXXQ002E	系统内存或磁盘空间不足。没有空间可包含生成的 XML 文档。
38X20	DXXQ003W	用户定义 SQL 查询生成比指定的最大值更多的 XML 文档。仅返回指定数目的文档。
38X21	DXXQ004E	指定的列不是 SQL 查询结果中的一列。
38X22	DXXQ005E	SQL 查询至 XML 的映射是不正确的。
38X23	DXXQ006E	文档访问定义 (DAD) 文件中的 attribute_node 元素没有名称属性。
38X24	DXXQ007E	文档访问定义 (DAD) 中的 attribute_node 元素没有列元素或 RDB_node。
38X25	DXXQ008E	文档访问定义 (DAD) 文件中的 text_node 元素没有列元素。
38X26	DXXQ009E	未能在系统目录中找到指定的结果表。
38X27	DXXQ010E	attribute_node 或 text_node 的 RDB_node 必须有一个表。

表 53. SQLSTATE 代码和相关联的消息号 (续)

SQLSTATE	消息号	描述
	DXXQ011E	attribute_node 或 text_node 的 RDB_node 必须具有列。
	DXXQ017E	XML Extender 生成的 XML 文档太大，以致于不能将它写入结果表的列中。
38X28	DXXQ012E	处理 DAD 时，XML Extender 未能找到期望的元素。
	DXXQ016E	所有表都必须在文档访问定义 (DAD) 文件中的顶层元素的 RDB_node 中定义。子元素表必须与顶层元素中定义的表相匹配。此 RDB_node 中的表名不在顶层元素中。
38X29	DXXQ013E	元素表或列必须在文档访问定义 (DAD) 文件中具有名称。
	DXXQ015E	文件访问定义 (DAD) 中的条件元素中的条件具有无效的格式。
38X30	DXXQ014E	文档访问定义 (DAD) 文件中 element_node 元素没有名称属性。
	DXXQ018E	ORDER BY 子句从文档访问定义 (DAD) 文件中的 SQL 语句中丢失，该 SQL 语句将 SQL 映射至 XML。
38X31	DXXQ019E	在将 SQL 映射到 XML 的文档访问定义 (DAD) 文件中，objids 元素没有列元素。
38X36	DXXA073E	用户尝试启用数据库时，该数据库却未绑定。
38X37	DXXG007E	服务器操作系统的语言环境与 DB2 代码页不一致。
38X38	DXXG008E	在代码页表中找不到服务器操作系统语言环境。
38x33	DXXG005E	此参数在此发行版中不受支持，但将在未来的发行版中受支持。
38x34	DXXG000E	指定了无效文件名。

消息

XML Extender 提供了错误消息以帮助您确定问题。

错误消息

当 XML Extender 完成一个操作或检测到一个错误时，会生成下列消息。

DXXA000I 正在启用列 <*column_name*>。请稍候。

解释: 这是信息性消息。

用户回答: 不需要任何操作。

DXXA001S 构建 <*build_ID*>、文件 <*file_name*> 和行 <*line_number*> 的过程中发生了意外错误。

解释: 发生了意外错误。

用户回答: 如果错误仍存在，则与“软件服务供应商”联系。在报告错误时，一定要包括所有的信息文本、跟踪文件和如何再现该问题的说明。

DXXA002I 连接至数据库 <*database*>。

解释: 这是信息性消息。

用户回答: 不需要任何操作。

DXXA003E 不能连接至数据库 <*database*>。

解释: 指定的数据库可能不存在或已毁坏。

用户回答:

1. 确保正确地指定了该数据库。
2. 确保该数据库存在且是可访问的。
3. 确定该数据库是否已毁坏。如果是的话，应请您的数据库管理员从备份来恢复它。

DXXA004E 不能启用数据库 <*database*>。

解释: 该数据库可能已经启用或已被毁坏。

用户回答:

1. 确定是否已启用该数据库。
2. 确定该数据库是否已毁坏。如果是的话，应请您的数据库管理员从备份来恢复它。

DXXA005I 正在启用数据库 <*database*>。请稍候。

解释: 这是信息性消息。

用户回答: 不需要任何操作。

DXXA006I 已成功启用了数据库 <*database*>。

解释: 这是信息性消息。

用户回答: 不需要任何操作。

DXXA007E 不能禁用数据库 <*database*>。

解释: 如果该数据库包含了任何 XML 列或集合，则它不能被 XML Extender 禁用。

用户回答: 备份任何重要的数据，禁用任何 XML 列或集合，并更新或删除任何表，直到数据库中不再有任何 XML 数据类型为止。

DXXA008I 正在禁用列 <*column_name*>。请稍候。

解释: 这是信息性消息。

用户回答: 不需要任何操作。

DXXA009E Xcolumn 标记未在 DAD 文件中指定。

解释: 此存储过程仅供 XML 列使用。

用户回答: 确保在 DAD 文件中正确地指定了 Xcolumn 标记。

DXXA010E 试图查找 DTD ID <*dtdid*> 失败。

解释: 当试图启用该列时, XML Extender 找不到 DTDID, 它是对文档访问定义 (DAD) 文件中的 DTD 指定的标识符。

用户回答: 确保在 DAD 文件中指定了 DTDID 的正确值。

DXXA011E 将记录插入到 DB2XML.XML_USAGE 表中失败。

解释: 当试图启用该列时, XML Extender 未能将记录插入到 DB2XML.XML_USAGE 表中。

用户回答: 确保 DB2XML.XML_USAGE 表存在, 且在该表中不存在同名的记录。

DXXA012E 试图更新 DB2XML.DTD_REF 表失败。

解释: 当试图启用该列时, XML Extender 未能更新 DB2XML.DTD_REF 表。

用户回答: 确保 DB2XML.DTD_REF 表存在。确定该表是否已损坏, 或管理用户标识是否具有正确的权限来更新该表。

DXXA013E 试图改变表 <*table_name*> 失败。

解释: 当试图启用该列时, XML Extender 未能改变指定的表。

用户回答: 选择改变该表所需的特权。

DXXA014E 指定的 Root 用户标识列: <*root_id*> 不是表 <*table_name*> 的单个主键。

解释: 指定的 Root 用户标识不是键, 或不是表 *table_name* 的单个键。

用户回答: 确保指定的 Root 用户标识是该表的单个主键。

DXXA015E 列 DXXROOT_ID 已经存在于表 <*table_name*> 中。

解释: 列 DXXROOT_ID 存在, 但它不是由 XML Extender 创建的。

用户回答: 当启用列时, 使用一个不同的列名, 对 Root 用户标识选项指定主列。

DXXA016E 输入表 <*table_name*> 不存在。

解释: XML Extender 在系统目录中找不到指定的表。

用户回答: 确保该表在数据库中存在, 且已经正确地指定了该表。

DXXA017E 指定的表 <*table_name*> 中不存在输入列 <*column_name*>。

解释: XML Extender 在系统目录中找不到该列。

用户回答: 确保用户表中存在该列。

DXXA018E 未对 XML 数据启用指定的列。

解释: 试图禁用该列时, XML Extender 未能在 DB2XML.XML_USAGE 表中找到该列, 这指示该列未被启用。如果该列未启用 XML, 则不需要禁用它。

用户回答: 不需要任何操作。

DXXA019E 启用列所需的输入参数为空。

解释: enable_column() 存储过程所需的输入参数为空。

用户回答: 检查用于 enable_column() 存储过程的所有输入参数。

DXXA020E 表 <*table_name*> 中找不到列。

解释: 当试图创建缺省视图时, XML Extender 在指定的表中找不到列。

用户回答: 确保正确指定了列和表名。

DXXA021E 不能创建缺省视图 <*default_view*>。

解释: 当试图启用一列时, XML Extender 未能创建指定的视图。

用户回答: 确保缺省视图名是唯一的。如果具有该名称的视图已经存在, 对缺省视图指定唯一的名称。

DXXA022I 列 <*column_name*> 已启用。

解释: 这是信息性消息。

用户回答: 不需要任何响应。

DXXA023E 找不到 DAD 文件。

解释: 当试图禁用一列时, XML Extender 找不到文档访问定义 (DAD) 文件。

用户回答: 确保指定了正确的数据库名、表名或列名。

DXXA024E 当访问系统目录表时, XML Extender 遇到了内部错误。

解释: XML Extender 无法访问系统目录表。

用户回答: 确保数据库处于稳定状态。

DXXA025E 不能删除缺省视图 <*default_view*>。

解释: 当试图禁用一列时, XML Extender 未能删除缺省视图。

用户回答: 确保 XML Extender 的管理员用户标识具有删除该缺省视图所需的特权。

DXXA026E 无法删除辅助表 <*side_table*>。

解释: 当试图禁用一列时, XML Extender 无法删除指定的表。

用户回答: 确保 XML Extender 的管理员用户标识具有删除该表所需的特权。

DXXA027E 无法禁用列。

解释: 由于内部触发器失败, XML Extender 无法禁用列。可能的原因:

用户回答: 使用跟踪设施创建一个跟踪文件并尝试校正该问题。如果问题仍存在, 请与软件供应商联系, 并将跟踪文件提供给他们。

DXXA028E 无法禁用列。

解释: 由于内部触发器失败, XML Extender 无法禁用列。可能的原因:

用户回答: 使用跟踪设施创建一个跟踪文件并尝试校正该问题。如果问题仍存在, 请与软件供应商联系, 并将跟踪文件提供给他们。

DXXA029E 无法禁用列。

解释: 由于内部触发器失败, XML Extender 无法禁用列。可能的原因:

用户回答: 使用跟踪设施创建一个跟踪文件并尝试校正该问题。如果问题仍存在, 请与软件供应商联系, 并将跟踪文件提供给他们。

DXXA030E 无法禁用列。

解释: 由于内部触发器失败, XML Extender 无法禁用列。可能的原因:

用户回答: 使用跟踪设施创建一个跟踪文件并尝试校正该问题。如果问题仍存在, 请与软件供应商联系, 并将跟踪文件提供给他们。

DXXA031E 无法将应用程序表中的 DXXROOT_ID 列值重设为 NULL。

解释: 当试图禁用一列时, XML Extender 无法将应用程序表中 DXXROOT_ID 的值设置为 NULL。

用户回答: 确保 XML Extender 的管理员用户标识具有改变应用程序表所需的特权。

DXXA032E DB2XML.XML_USAGE 表中的 USAGE_COUNT 递减失败。

解释: 当试图禁用该列时, XML Extender 无法逐一地减少 USAGE_COUNT 列的值。

用户回答: 确保 DB2XML.XML_USAGE 表存在, 且 XML Extender 的管理员用户标识具有更新该表的所需特权。

DXXA033E 试图从 DB2XML.XML_USAGE 表删除行失败。

解释: 当试图禁用一列时, XML Extender 无法删除 DB2XML.XML_USAGE 表中的相关行。

用户回答: 确保 DB2XML.XML_USAGE 表存在且 XML Extender 管理员用户标识具有更新此表所需的特权。

DXXA034I XML Extender 成功地禁用了列 <*column_name*>。

解释: 这是信息性消息

用户回答: 不需要任何操作。

DXXA035I XML Extender 正在禁用数据库 <*database*>。请稍候。

解释: 这是信息性消息。

用户回答: 不需要任何操作。

DXXA036I XML Extender 成功地禁用了数据库 <*database*>。

解释: 这是信息性消息。

用户回答: 不需要任何操作。

DXXA037E 指定的表空间名长于 18 个字符。

解释: 表空间名不能长于 18 个字母数字字符。

用户回答: 指定少于 18 个字符的名称。

DXXA038E 指定的缺省视图名长于 18 个字符。

解释: 缺省的视图名不能长于 18 个字母数字字符。

用户回答: 指定少于 18 个字符的名称。

DXXA039E 指定的 ROOT_ID 名长于 18 个字符。

解释: ROOT_ID 名不能长于 18 个字母数字字符。

用户回答: 指定少于 18 个字符的名称。

DXXA046E 无法创建辅助表 <*side_table*>。

解释: 当试图启用一列时, XML Extender 无法创建指定的辅助表。

用户回答: 确保 XML Extender 的管理员用户标识具有创建辅助表所需的特权。

DXXA047E 无法启用列。

解释: 由于内部触发器失败, XML Extender 无法启用列。可能的原因:

用户回答: 使用跟踪设施创建一个跟踪文件并尝试校正该问题。如果问题仍存在, 请与软件供应商联系, 并将跟踪文件提供给他们。

DXXA048E 无法启用列。

解释: 由于内部触发器失败, XML Extender 无法启用列。可能的原因:

用户回答: 使用跟踪设施创建一个跟踪文件并尝试校正该问题。如果问题仍存在, 请与软件供应商联系, 并将跟踪文件提供给他们。

DXXA049E 无法启用列。

解释: 由于内部触发器失败, XML Extender 无法启用列。可能的原因:

用户回答: 使用跟踪设施创建一个跟踪文件并尝试校正该问题。如果问题仍存在, 请与软件供应商联

系，并将跟踪文件提供给他们。

DXXA050E 无法启用列。

解释: 由于内部触发器失败，XML Extender 无法启用列。可能的原因:

用户回答: 使用跟踪设施创建一个跟踪文件并尝试校正该问题。如果问题仍存在，请与软件供应商联系，并将跟踪文件提供给他们。

DXXA051E 无法禁用列。

解释: 由于内部触发器失败，XML Extender 无法禁用列。可能的原因:

用户回答: 使用跟踪设施创建一个跟踪文件并尝试校正该问题。如果问题仍存在，请与软件供应商联系，并将跟踪文件提供给他们。

DXXA052E 无法禁用列。

解释: 由于内部触发器失败，XML Extender 无法禁用列。可能的原因:

用户回答: 使用跟踪设施创建一个跟踪文件并尝试校正该问题。如果问题仍存在，请与软件供应商联系，并将跟踪文件提供给他们。

DXXA053E 无法启用列。

解释: 由于内部触发器失败，XML Extender 无法启用列。可能的原因:

用户回答: 使用跟踪设施创建一个跟踪文件并尝试校正该问题。如果问题仍存在，请与软件供应商联系，并将跟踪文件提供给他们。

DXXA054E 无法启用列。

解释: 由于内部触发器失败，XML Extender 无法启用列。可能的原因:

用户回答: 使用跟踪设施创建一个跟踪文件并尝试校正该问题。如果问题仍存在，请与软件供应商联系，并将跟踪文件提供给他们。

DXXA056E DAD 文件中的验证值 <*validation_value*> 是无效的。

解释: 文档访问定义 (DAD) 文件中的验证元素是错误的或已经丢失。

用户回答: 确保在 DAD 文件中正确地指定了验证元素。

DXXA057E DAD 中的辅助表名 <*side_table_name*> 是无效的。

解释: 文档访问定义 (DAD) 文件中辅助表的名称属性是错误的或已经丢失。

用户回答: 确保在 DAD 文件中正确地指定了辅助表的名称属性。

DXXA058E DAD 文件中的列名 <*column_name*> 是无效的。

解释: 文档访问定义 (DAD) 文件中某列的名称属性是错误的或已经丢失。

用户回答: 确保在 DAD 文件中正确地指定了某列的名称属性。

DXXA059E DAD 文件中的列 <*column_name*> 的类型 <*column_type*> 是无效的。

解释: 文档访问定义 (DAD) 文件中某列的类型属性是错误的或已经丢失。

用户回答: 确保在 DAD 文件中正确地指定了某列的类型属性。

DXXA060E DAD 文件中的 <*列名*> 的路径属性 <*location_path*> 无效。

解释: 文档访问定义 (DAD) 文件中列的路径属性错误或已丢失。

用户回答: 确保在 DAD 文件中正确地指定了某列的路径属性。

DXXA061E DAD 文件中的 <*column_name*> 的 multi_occurrence 属性 <*multi_occurrence*> 是无效的。

解释: 文档访问定义 (DAD) 文件中某列的 multi_occurrence 属性是错误的或已经丢失。

用户回答: 确保在 DAD 文件中正确地指定了某列的 multi_occurrence 属性。

DXXA062E 无法检索表 <*table_name*> 中的 <*column_name*> 的列号。

解释: XML Extender 未能从系统目录检索表 *table_name* 中的 *column_name* 的列号。

用户回答: 确保正确定义了应用程序表。

DXXA063I 正在启用集合 <*collection_name*>。请稍候。

解释: 这是信息性消息。

用户回答: 不需要任何操作。

DXXA064I 正在禁用集合 <*collection_name*>。请稍候。

解释: 这是信息性消息。

用户回答: 不需要任何操作。

DXXA065E 调用存储过程 <*procedure_name*> 失败。

解释: 检查共享库 db2xml 并查看许可权是否正确。

用户回答: 确保客户机具有运行存储过程的许可权。

DXXA066I XML Extender 已成功地禁用了集合 <*collection_name*>。

解释: 这是信息性消息。

用户回答: 不需要任何响应。

DXXA067I XML Extender 已成功地启用了集合 <*collection_name*>。

解释: 这是信息性消息。

用户回答: 不需要任何响应。

DXXA068I XML Extender 成功打开跟踪。

解释: 这是信息性消息。

用户回答: 不需要任何响应。

DXXA069I XML Extender 成功关闭跟踪。

解释: 这是信息性消息。

用户回答: 不需要任何响应。

DXXA070W 已经启用了数据库。

解释: 对已启用的数据库执行了启用数据库命令

用户回答: 不需要任何操作。

DXXA071W 已经禁用了数据库。

解释: 对已禁用的数据库执行了禁用数据库命令

用户回答: 不需要任何操作。

DXXA072E XML Extender 未能找到绑定文件。在启用数据库之前对其进行绑定。

解释: XML Extender 尝试在启用数据库之前自动绑定该数据库，但未能找到绑定文件

用户回答: 在启用数据库之前对其进行绑定。

DXXA073E 未绑定该数据库。请在启用数据库之前对其进行绑定。

解释: 用户尝试启用数据库时，该数据库却未绑定。

用户回答: 在启用数据库之前对其进行绑定。

DXXA074E 参数类型错误。存储过程期望 STRING 参数。

解释: 存储过程期望 STRING 参数。

用户回答: 将输入参数说明为 STRING 类型。

DXXA075E 参数类型错误。输入参数应为 LONG 类型。

解释: 存储过程期望输入参数为 LONG 类型。

用户回答: 将输入参数说明为 LONG 类型。

DXXA076E XML Extender 跟踪实例标识无效。

解释: 无法使用所提供的实例标识启动跟踪。

用户回答: 确保实例标识是一个有效的 AS/400 用户标识。

DXXC000E 无法打开指定文件。

解释: XML Extender 无法打开指定文件。

用户回答: 确保应用程序用户标识对该文件具有读写许可权。

DXXC001E 未找到指定文件。

解释: XML Extender 未能找到指定的文件。

用户回答: 确保该文件存在，且正确指定了路径。

DXXC002E 无法读取文件。

解释: XML Extender 无法从指定文件中读取数据。

用户回答: 确保应用程序用户标识对该文件具有读许可权。

DXXC003E 无法写入指定文件。

解释: XML Extender 无法将数据写入指定文件。

用户回答: 确保应用程序用户标识对该文件具有写许可权，或该文件系统具有足够的空间。

DXXC004E 无法操作“LOB 定位器”：rc=<*locator_rc*>。

解释: XML Extender 无法操作指定的定位器。

用户回答: 确保正确设置了 “LOB 定位器”。

DXXC005E 输入文件大小大于 XMLVarchar 大小。

解释: 该文件的大小比 XMLVarchar 的大小要大，且 XML Extender 无法从该文件导入所有数据。

用户回答: 使用 XMLCLOB 列类型。

DXXC006E 输入文件超过 DB2 LOB 的限制。

解释: 该文件的大小比 XMLCLOB 的大小要大，且 XML Extender 不能从该文件导入所有数据。

用户回答: 将该文件分解为较小的对象或使用 XML 集合。

DXXC007E 无法将数据从文件检索至“LOB 定位器”。

解释: “LOB 定位器”中的字节数不等于文件大小。

用户回答: 确保正确设置了 “LOB 定位器”。

DXXC008E 不能除去文件 <*文件名*>。

解释: 该文件处于共享访问违例状态或仍然打开。

用户回答: 关闭该文件，或停止任何正在使用该文件的进程。您可能必须停止并重新启动 DB2。

DXXC009E 无法将文件创建至 <*directory*> **目录。**

解释: XML Extender 无法在目录 *directory* 中创建文件。

用户回答: 确保该目录存在，应用程序用户标识对该目录具有写许可权，并且文件系统具有足够的空间来容纳该文件。

DXXC010E　写入文件 <*file_name*> 时出错。

解释: 写入文件*文件名*时出错。

用户回答: 确保该文件系统具有足够的空间来容纳该文件。

DXXC011E　无法写入跟踪控制文件。

解释: XML Extender 无法将数据写入跟踪控制文件。

用户回答: 确保应用程序用户标识对该文件具有写许可权，或该文件系统具有足够的空间。

DXXC012E　不能创建临时文件。

解释: 不能在系统临时目录中创建文件。

用户回答: 确保应用程序用户标识对文件系统临时目录具有写许可权，或该文件系统具有足够的空间来容纳该文件。

DXXD000E　拒绝无效的 XML 文档。

解释: 试图将无效的文档存储到表中。验证失败。

用户回答: 使用可查看看不到的无效字符的编辑器，检查文档和它的 DTD。要抑制这种错误，可关闭 DAD 文件中的验证。

DXXD001E　<*location_path*> 多次出现。

解释: 标量抽取函数所使用的位置路径多次出现。标量函数仅可使用没有多次出现的位置路径。

用户回答: 使用表函数（将 's' 添加至标量函数名的末尾）。

DXXD002E　在搜索路径的附近位置 <*position*> 处发生语法错误。

解释: 路径表达式在语法上不正确。

用户回答: 校正查询的搜索路径自变量。参考用于路径表达式的语法的文档。

DXXD003W　未找到路径。返回了“空”值。

解释: 在路径表达式中指定的元素或属性已从 XML 文档中丢失。

用户回答: 验证指定的路径是否正确。

DXXG000E　文件名 <*file_name*> 无效。

解释: 指定了无效文件名。

用户回答: 指定正确的文件名并重试。

DXXG001E　在构建 <*build_ID*>、文件 <*file_name*> 和行 <*line_number*> 时出现内部错误。

解释: XML Extender 遇到内部错误。

用户回答: 与“软件服务供应商”联系。在报告错误时，一定要包括所有的信息、跟踪文件和如何再现该错误的说明。

DXXG002E　系统内存不足。

解释: XML Extender 无法从操作系统分配内存。

用户回答: 关闭某些应用程序并重试。如果问题持续存在，参考操作系统文档以获取帮助。某些操作系统可能需要您重新引导该系统以校正问题。

DXXG004E　无效的空值参数。

解释: 所需参数的空值被传送至 XML 存储过程。

用户回答: 检查自变量列表中用于存储过程调用的所有必需的参数。

DXXG005E　不受支持的参数。

解释: 此参数在此发行版中不受支持，但将在未来的发行版中受支持。

用户回答: 将此参数设置为 NULL。

DXXG006E 内部错误 CLISTATE=<*clistate*>、**RC=**<*cli_rc*>、**构建** <*build_ID*>、**文件** <*file_name*>、**行** <*line_number*> **CLIMSG=**<*CLI_msg*>。

解释: 使用 CLI 时，XML Extender 遇到了内部错误。

用户回答: 与“软件服务供应商”联系。潜在地，此错误可由不正确的用户输入而导致。报告错误时，一定要包括所有的输出信息、跟踪日志和如何再现该问题的说明。在任何可能的地方，发送任何适用的 DAD、XML 文档和表定义。

DXXG007E 语言环境 <*locale*> **与 DB2 代码页** <*code_page*> **不一致。**

解释: 服务器操作系统的语言环境与 DB2 代码页不一致。

用户回答: 校正服务器操作环境的语言环境然后重新启动 DB2。

DXXG008E 语言环境 <*locale*> **不受支持。**

解释: 在代码页表中找不到服务器操作系统语言环境。

用户回答: 校正服务器操作环境的语言环境然后重新启动 DB2。

DXXQ000E <*Element*> **从 DAD 文件丢失。**

解释: 必要的元素已从文档访问定义 (DAD) 文件丢失。

用户回答: 将已丢失的元素添加至 DAD 文件。

DXXQ001E 对生成 XML 无效的 SQL 语句。

解释: 文档访问定义 (DAD) 中的 SQL 语句或覆盖它的某语句是无效的。 SELECT 语句对于生成 XML 文档是必需的。

用户回答: 校正 SQL 语句。

DXXQ002E 不能生成用于保存 XML 文档的存储空间。

解释: 系统内存或磁盘空间不足。没有空间可包含生成的 XML 文档。

用户回答: 限制要生成的文档数。通过从文档访问定义 (DAD) 文件除去某些不必要的元素节点和属性节点，来缩小每个文档的大小。

DXXQ003W 结果超过最大数。

解释: 用户定义 SQL 查询生成比指定的最大值更多的 XML 文档。仅返回指定数目的文档。

用户回答: 不需要任何操作。 如果需要所有的文档，则将零指定为最大文档数。

DXXQ004E 列 <*column_name*> **不在查询的结果之中。**

解释: 指定的列不是 SQL 查询结果中的一列。

用户回答: 在文档访问定义 (DAD) 文件中更改指定列名，以使其成为 SQL 查询结果中的一列。或者，也可以更改 SQL 查询以便它可在其结果中具有指定列。

DXXQ004W DAD 中未找到 DTD ID。

解释: 在 DAD 中， VALIDATION 为 YES，但未指定 DTDID 元素。不执行任何验证检查。

用户回答: 不需要任何操作。 如果需要验证，则在 DAD 文件中指定 DTDID 元素。

DXXQ005E 错误的关系映射。元素 <*element_name*> **处于比其子辈列** <*column_name*> **较低的级别上。**

解释: SQL 查询至 XML 的映射是不正确的。

用户回答: 确保 SQL 查询结果中的列采用的是从上向下的关系层次结构。还应确保有单列候选键来开始每一级别。如果表中没有这样的键可用，该查询应使用表表达式和 DB2 内部函数 generate_unique()，对该表生成一个键。

DXXQ006E attribute_node 元素没有任何名称。

解释: 文档访问定义 (DAD) 文件中的 attribute_node 元素没有名称属性。

用户回答: 确保每个 attribute_node 在 DAD 文件中都有一个名称。

DXXQ007E attribute_node <*attribute_name*> 不具有任何列元素或 RDB_node。

解释: 文档访问定义 (DAD) 中的 attribute_node 元素不具有列元素或 RDB_node。

用户回答: 确保每个 attribute_node 在 DAD 中都有一个列元素或 RDB_node。

DXXQ008E text_node 元素没有任何列元素。

解释: 文档访问定义 (DAD) 文件中的 text_node 元素不具有列元素。

用户回答: 确保每个 text_node 在 DAD 中都有一个列元素。

DXXQ009E 结果表 <*table_name*> 不存在。

解释: 未能在系统目录中找到指定的结果表。

用户回答: 在调用存储过程之前创建结果表。

DXXQ010E <*node_name*> 的 RDB_node 在 DAD 文件中没有表。

解释: attribute_node 或 text_node 的 RDB_node 必须有一个表。

用户回答: 对文档访问定义 (DAD) 文件中的 attribute_node 或 text_node 指定 RDB_node 的表。

DXXQ011E <*node_name*> 的 RDB_node 元素在 DAD 文件中不具有列。

解释: attribute_node 或 text_node 的 RDB_node 必须具有列。

用户回答: 在文档访问定义 (DAD) 文件中指定 attribute_node 或 text_node 的 RDB_node 的列。

DXXQ012E DAD 中出错。

解释: 处理 DAD 时, XML Extender 未能找到期望的元素。

用户回答: 检查 DAD 是否为有效的 XML 文档, 且是否包含 DAD DTD 必需的所有元素。参见 XML Extender 出版物以获取 DAD DTD。

DXXQ013E 表或列元素在 DAD 文件中不具有名称。

解释: 元素表或列必须在文档访问定义 (DAD) 文件中具有名称。

用户回答: 在 DAD 中指定表或列元素的名称。

DXXQ014E element_node 元素没有任何名称。

解释: 文件访问定义 (DAD) 文件中的 element_node 元素不具有名称属性。

用户回答: 确保每个 element_node 元素在 DAD 文件中具有名称。

DXXQ015E 条件格式是无效的。

解释: 文件访问定义 (DAD) 中的条件元素中的条件具有无效的格式。

用户回答: 确保条件的格式是有效的。

DXXQ016E 此 RDB_node 中的表名未在 DAD 文件的顶部元素中定义。

解释: 所有表必须在文档访问定义 (DAD) 文件中顶部元素的 RDB_node 中定义。子元素表必须与在顶部元素中定义的表匹配。此 RDB_node 中的表名不在顶部元素中。

用户回答: 确保 RDB 节点的表是在 DAD 文件的顶部元素中定义的。

DXXQ017E 结果表 <*table_name*> 中的列太小。

解释: 由 XML Extender 生成的 XML 文档太大而不能装入结果表的列中。

用户回答: 删除结果表。用较大的列创建另一结果表。重新运行存储过程。

DXXQ018E ORDER BY 子句从 SQL 语句丢失。

解释: ORDER BY 子句从文档访问定义 (DAD) 文件中的 SQL 语句中丢失，该 SQL 语句将 SQL 映射至 XML。

用户回答: 编辑 DAD 文件。添加包含实体标识列的 ORDER BY 子句。

DXXQ019E 元素 objids 在 DAD 文件中没有列元素。

解释: 在将 SQL 映射到 XML 的文档访问定义 (DAD) 文件中，objids 元素没有列元素。

用户回答: 编辑 DAD 文件。添加键列，作为元素 objids 的子元素。

DXXQ020I 成功地生成了 XML。

解释: 已经从数据库成功地生成了请求的 XML 文档。

用户回答: 不需要任何操作。

DXXQ021E 表 <*table_name*> 中没有列 <*column_name*>。

解释: 该表在数据库中不具有指定的列。

用户回答: 在 DAD 中指定另一列名，或将指定列添加至表数据库。

DXXQ022E <*table_name*> 的列 <*column_name*> 应具有类型 <*type_name*>。

解释: 列类型是错误的。

用户回答: 校正文档访问定义 (DAD) 中的列的类型。

DXXQ023E <*table_name*> 的列 <*column_name*> 长度不能超过 <*length*>。

解释: 在 DAD 中对该列定义的长度太长。

用户回答: 校正文档访问定义 (DAD) 中的列长度。

DXXQ024E 不能创建表 <*table_name*>。

解释: 不能创建指定的表。

用户回答: 确保创建表的用户标识具有在数据库中创建表所必需的权限。

DXXQ025I 成功地分解了 XML。

解释: XML 文档已被成功地被分解并存储在集合中。

用户回答: 不需要任何操作。

DXXQ026E XML 数据 <*xml_name*> 太长，在列 <*column_name*> 中容纳不下。

解释: XML 文档中指定的数据块太大而不能装入指定列。

用户回答: 使用 ALTER TABLE 语句增加该列的长度，或通过编辑 XML 文档减小数据的大小。

DXXQ028E 在 XML_USAGE 表中找不到集合 <*collection_name*>。

解释: 在 XML_USAGE 表中找不到集合的记录。

用户回答: 验证您是否已经启用了集合。

DXXQ029E 在 XML_USAGE 表中找不到集合 <*collection_name*> 的 DAD。

解释: 在 XML_USAGE 表中找不到该集合的 DAD 记录。

用户回答: 确保您已经正确地启用了该集合。

DXXQ030E 错误的 XML 覆盖语法。

解释: 在存储过程中不正确地指定了 XML_override 值。

用户回答: 确保 XML_override 的语法正确。

DXXQ031E 表名的长度不能超过 DB2 所允许的最大长度。

解释: 在 DAD 中，由条件元素指定的表名太长。

用户回答: 校正文档访问定义 (DAD) 中表名的长度。

DXXQ032E 列名的长度不能超过 DB2 所允许的最大长度。

解释: 在 DAD 中，由条件元素指定的列名太长。

用户回答: 校正文档访问定义 (DAD) 中的列名的长度。

DXXQ033E 以 <*identifier*> 开头的标识符无效

解释: 该字符串为无效的 DB2 SQL 标识符。

用户回答: 校正 DAD 中的字符串以符合 DB2 SQL 标识符的规则。

DXXQ034E DAD 的顶部 RDB_node 中的条件元素无效: <*condition*>

解释: 该条件元素必须为有效的 WHERE 子句，有效子句由“与”运算符 AND 连接的连接条件组成。

用户回答: 参见 XML Extender 文档，以了解 DAD 中连接条件的正确语法。

DXXQ035E DAD 的顶部 RDB_node 中的连接条件无效: <*condition*>

解释: 顶部 RDB_node 的条件元素中的列名必须以表名限定（如果 DAD 指定多个表的话）。

用户回答: 参见 XML Extender 文档，以了解 DAD 中连接条件的正确语法。

DXXQ036E DAD 条件标记下指定的模式名超过了允许的长度。

解释: 对 DAD 中条件标记下的文本进行语法分析时检测到错误。条件文本中包含了一个由过长的模式名限定的标识。

用户回答: 校正文档访问定义 (DAD) 中条件标记的文本。

DXXQ037E 不能生成多次出现的 <*element*>。

解释: 该元素节点及其后代与数据库没有映射关系，但其 multi_occurrence 等于 YES。

用户回答: 将 multi_occurrence 设置为 NO 或在它的一个子代中创建 RDB_node，以校正 DAD。

诊断跟踪

XML Extender 中包括了记录 XML Extender 服务器活动的跟踪功能。您只应在“IBM 软件支持”人员的指导下才使用跟踪功能。

跟踪功能在服务器文件中记录了有关各种事件的信息，如进入 XML Extender 部件或从该部件退出，或返回 XML Extender 部件的错误码。因为它记录了许多事件的信息，所以只应在需要时才使用跟踪功能，例如，正在调查错误情况时，可使用此功能。此外，您应在使用跟踪功能时限制活动的应用程序数。限制活动的应用程序数可使查出问题的原因变得更容易。

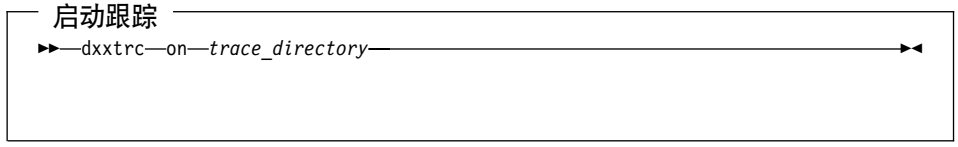
使用 **dxxtrc** 命令来启动或结束跟踪。可从 AIX、Windows NT 或 Solaris 服务器上的命令行发出此命令。 您必须具有 SYSADM、SYSCTRL 或 SYSMINT 权限才能发出该命令。

启动跟踪

目的

记录 XML Extender 服务器活动。要启动跟踪，对 **dxxtrc** 应用开选项，并输入包含跟踪文件的目录的名称。当打开跟踪时，文件 dxx*INSTANCE*.trc 被放在指定的目录中。*INSTANCE* 是 DB2INSTANCE 的值。每个 DB2 实例都有其自己的日志文件。

格式



参数

表 54. 跟踪参数

参数	描述
trace_directory	放置 dxx*INSTANCE*.trc 的目录的名称。它是必需的，无缺省值。

例

以下示例演示在 AIX 上对实例 db2inst1 的跟踪。跟踪文件 dxxdb2inst1.trc 放置在 /home/db2inst1/sqllib/log 目录中。

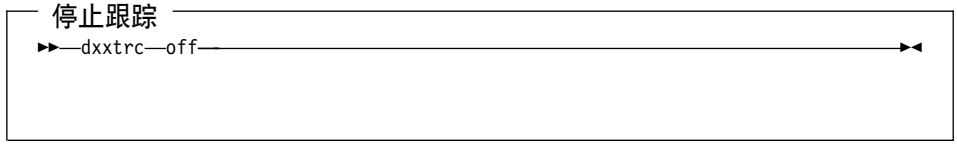
```
dxxtrc on /home/db2inst1/sqllib/log
```

停止跟踪

目的

关闭跟踪。不记录任何跟踪信息。因为运行跟踪会影响性能，所以建议在实际应用环境中关闭跟踪。

格式



例

此示例显示跟踪功能处于关闭状态。

```
dxxtrc off
```

第5部分 附录及附属资料

© Copyright IBM Corp. 1999, 2000

附录A. 用于 DAD 文件的 DTD

本节描述用于文档访问定义 (DAD) 文件的文档类型说明 (DTD)。DAD 文件本身是树形结构的 XML 文档且需要 DTD。DTD 文件名为 `dxxdad.dtd`。第234页的图13显示用于 DAD 文件的 DTD。下图描述了此文件的元素。

© Copyright IBM Corp. 1999, 2000

```
<?xml encoding="US-ASCII"?>

  <!ELEMENT DAD (dtdid?, validation, (Xcolumn | Xcollection))>
  <!ELEMENT dtdid (#PCDATA)>
  <!ELEMENT validation (#PCDATA)>
  <!ELEMENT Xcolumn (table*)>
  <!ELEMENT table (column*)>
  <!ATTLIST table name CDATA #REQUIRED
                              key CDATA #IMPLIED
                              orderBy CDATA #IMPLIED>
  <!ELEMENT column EMPTY>
  <!ATTLIST column
                      name CDATA #REQUIRED
                      type CDATA #IMPLIED
                      path CDATA #IMPLIED
                       multi_occurrence CDATA #IMPLIED>
  <!ELEMENT Xcollection (SQL_stmt?, objids?, prolog, doctype, root_node)>
  <!ELEMENT SQL_stmt (#PCDATA)>
  <!ELEMENT objids (column+)>
  <!ELEMENT prolog (#PCDATA)>
  <!ELEMENT doctype (#PCDATA | RDB_node)*>
  <!ELEMENT root_node (element_node)>
  <!ELEMENT element_node (RDB_node*,
                                       attribute_node*,
                                       text_node?,
                                       element_node*,
                                       namespace_node*,
                                       process_instruction_node*,
                                       comment_node*)>
  <!ATTLIST element_node
                     name CDATA #REQUIRED
                      ID CDATA #IMPLIED
                     multi_occurrence CDATA "NO"
                     BASE_URI CDATA #IMPLIED>
  <!ELEMENT attribute_node (column | RDB_node)>
  <!ATTLIST attribute_node
                     name CDATA #REQUIRED>
  <!ELEMENT text_node (column | RDB_node)>
  <!ELEMENT RDB_node (table+, column?, condition?)>
  <!ELEMENT condition (#PCDATA)>
  <!ELEMENT comment_node (#PCDATA)>
  <!ELEMENT namespace_node (EMPTY)>
  <!ATTLIST namespace_node
                      name CDATA #IMPLIED
                     value CDATA #IMPLIED>
  <!ELEMENT process_instruction_node (#PCDATA)>
```

图 13. 用于文档访问定义 (DAD) 的 DTD

DAD 文件有四个主要元素:

- DTDID
- 验证

- Xcolumn
- Xcollection

Xcolumn 和 Xcollection 具有子元素和属性，它们有助于将 XML 数据映射到 DB2 中的关系表。以下列表描述了主要元素及其子元素和属性。语法示例取自第234页的图13。

DTDID 元素

指定存储在 DTD_REF 表中的 DTD 的标识。DTDID 指向验证 XML 文档或指导 XML 集合表与 XML 文档之间的映射的 DTD。必须对 XML 集合指定 DTDID。对于 XML 列，它是可选的，且仅当想要为对元素或属性创建索引而创建辅助表时，或验证输入 XML 文档时才需要它。DTDID 必须与 XML 文档的 doctype 中指定的 SYSTEM 标识相同。

语法: `<!ELEMENT dtdid (#PCDATA)>`

验证元素

指示是否要使用用于 DAD 的 DTD 来验证 XML 文档。如果指定 YES，则还必须指定 DTDID。

语法: `<!ELEMENT validation(#PCDATA)>`

Xcolumn 元素

定义为 XML 列创建索引的模式。它由零个或多个表组成。

语法: `<!ELEMENT Xcolumn (table*)>`Xcolumn 有一个子元素，表。.

表元素 定义一个或多个为文档的元素或属性创建索引而创建的关系表，这些文档存储在 XML 列中。

语法:

```
<!ELEMENT table (column+)>
 <!ATTLIST table name CDATA #REQUIRED
key CDATA #IMPLIED
orderBy CDATA #IMPLIED>
```

表元素有一个属性:

名称属性

指定辅助表的名称

表元素有一个子元素:

键属性 表的主要单个键

orderBy 属性

列的名称，这些列确定当生成 XML 文档时，多次出现的元素文本和属性值的顺序次序。

列元素 指定表的列，该列包含指定类型的位置路径的值。

语法:

```
<!ATTLIST column
                      name CDATA #REQUIRED
                      type  CDATA #IMPLIED
                      path  CDATA #IMPLIED
                      multi_occurrence CDATA #IMPLIED>
```

列元素有下列属性:

名称属性

指定列的名称。它是标识元素或属性的位置路径的别名

类型属性

定义列的数据类型。它可以是任何 SQL 数据类型。

路径属性

显示 XML 元素或属性的位置路径，且该路径必须为在表 3.1.a（修正链接）中指定的简单位置路径。

multi_occurrence 属性

指示此元素或属性是否可在一个 XML 文档中出现多次。值可为 YES 或 NO。

Xcollection

定义 XML 文档与关系表的 XML 集合之间的映射。

语法: `<!ELEMENT Xcollection(SQL_stmt*, prolog, doctype, root_node)>Xcollection` 有以下子元素:

SQL_stmt

指定 XML Extender 用于定义集合的 SQL 语句。准确地讲，该语句从 XML 集合表选择 XML 数据，并使用该数据以在集合中生成 XML 文档。 此元素的值必须为有效的 SQL 语句。 它仅用于组合，且仅允许一个 `SQL_stmt`。 对于分解，可指定 `SQL_stmt` 的多个值，以执行必要的表创建和插入。

语法: `<!ELEMENT SQL_stmt  #PCDATA >`

prolog

XML prolog 的文本。在整个集合中对所有文档提供同一 prolog。`prolog` 的值是固定的。

语法: `<!ELEMENT prolog  #PCDATA>`

doctype

定义 XML 文档类型定义的文本。

语法: <!ELEMENT doctype #PCDATA | RDB_node>doctype 可以下列其中一种方式指定:

- 定义显式值。此值是对整个集合中的所有文档提供的。
- 当使用分解时，指定可被映射至、且作为表的列数据存储的子元素 RDB_node。

doctype 有一个子元素:

RDB_node

尚未实现。

root_node

定义虚拟根节点。root_node 必须具有一个必需的子元素 element_node，它仅可被使用一次。 root_node 下的 element_node 实际上是 XML 文档的 root_node。

语法: <!ELEMENT root_node(element_node)>

element_node

表示 XML 元素。它必须在对集合指定的 DTD 中定义。对于 RDB_node 映射，根 element_node 必须具有 RDB_node，以指定包含用于它自身及所有其子节点的 XML 数据的所有表。它可具有零个或多个 attribute_nodes 和子 element_nodes，以及零个或一个 text_node。对于不同于根元素的元素，不需要任何 RDB_node。

语法:

一个 element_node 是由下列子元素定义的:

RDB_node

（可选）指定 XML 数据的表、列和条件。仅需要对 RDB_node 映射定义元素的 RDB_node。在这种情况中，必须指定一个或多个表。由于元素内容是由 text_node 指定的，所以不需要该列。条件是可选的，视 DTD 和查询条件而定。

子节点 （可选）element_node 还可有下列子节点:

element_node

表示当前 XML 元素的子节点

attribute_node

表示当前 XML 元素的属性

text_node

表示当前 XML 元素的 CDATA 文本

attribute_node

表示 XML 属性。一个节点，它定义 XML 属性与关系表中列数据间的映射。

语法:

attribute_node 必须具有对名称属性、列或 RDB_node 子元素的定义。 attribute_node 具有以下属性:

名称 属性的名称。

attribute_node 具有下列子元素:

列 用于 SQL 映射。该列必须在 SQL_stmt 的 SELECT 子句中指定。

RDB_node

用于 RDB_node 映射。该节点定义此属性与关系表中的列数据间的映射。必须指定该表和列。该条件是可选的。

text_node

表示 XML 元素的文本内容。一个节点，它定义 XML 元素内容与关系表中列数据间的映射。

语法: 它必须由 列或 RDB_node 子元素来定义:

列 SQL 映射需要用到。在这种情况中，该列必须在 SQL_stmt 的 SELECT 子句中。

RDB_node

RDB_node 映射需要使用。该节点定义此文本内容与关系表中列数据间的映射。必须指定表 和列。该条件是可选的。

附录B. 样本

此附录显示在本书中使用的示例的样本对象。

- 『XML DTD』
- 『XML 文档: getstart.xml』
- 第240页的『文档访问定义文件』
  - 第241页的『DAD 文件: XML 列』
  - 第241页的『DAD 文件: XML 集合 - SQL 映射』
  - 第243页的『DAD 文件: XML - RDB_node 映射』

XML DTD

以下 DTD 用于整本书中引用且显示在第240页的图15中的 getstart.xml 文档。

```
<!xml encoding="US-ASCII"?>

<!ELEMENT Order (Customer, Part+)>
<!ATTLIST Order key CDATA #REQUIRED>
<!ELEMENT Customer (Name, Email)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Email (#PCDATA)>
<!ELEMENT Part (key,Quantity,ExtendedPrice,Tax, Shipment+)>
<!ELEMENT key (#PCDATA)>
<!ELEMENT Quantity (#PCDATA)>
<!ELEMENT ExtendedPrice (#PCDATA)>
<!ELEMENT Tax (#PCDATA)>
<!ATTLIST Part color CDATA #REQUIRED>
<!ELEMENT Shipment (ShipDate, ShipMode)>
<!ELEMENT ShipDate (#PCDATA)>
<!ELEMENT ShipMode (#PCDATA)>
```

图 14. 样本 XML DTD: getstart.dtd

XML 文档: getstart.xml

以下 XML 文档 getstart.xml 为在本书各示例中使用的样本 XML 文档: 它包含要构成采购定单的 XML 标记。

© Copyright IBM Corp. 1999, 2000

```
<?xml version="1.0"?>
<!DOCTYPE Order SYSTEM "c:\dxx\samples\dtd\getstart.dtd">
<Order key="1">
  <Customer>
    <Name>American Motors</Name>
    <Email>parts@am.com</Email>
  </Customer>
  <Part color="black ">
    <key>68</key>
    <Quantity>36</Quantity>
    <ExtendedPrice>34850.16</ExtendedPrice>
    <Tax>6.000000e-02</Tax>
    <Shipment>
      <ShipDate>1998-08-19</ShipDate>
      <ShipMode>BOAT  </ShipMode>
    </Shipment>
    <Shipment>
       <ShipDate>1998-08-19</ShipDate>
       <ShipMode>AIR   </ShipMode>
    </Shipment>
  </Part>
  <Part color="red   ">
    <key>128</key>
    <Quantity>28</Quantity>
    <ExtendedPrice>38000.00</ExtendedPrice>
    <Tax>7.000000e-02</Tax>
    <Shipment>
      <ShipDate>1998-12-30</ShipDate>
      <ShipMode>TRUCK </ShipMode>
    </Shipment>
  </Part>
</Order>
```

图 15. 样本 XML 文档: getstart.xml

文档访问定义文件

下列部分包含文档访问定义 (DAD) 文件，该文件使用 XML 列或 XML 集合访问方式，将 XML 数据映射至 DB2 关系表。

- 第241页的『DAD 文件: XML 列』
- 第241页的『DAD 文件: XML 集合 - SQL 映射』显示使用 SQL 映射的 XML 集合的 DAD 文件。
- 第243页的『DAD 文件: XML - RDB_node 映射』显示使用 RDB_node 映射的 XML 集合的 DAD。

DAD 文件: XML 列

此 DAD 文件包含用于 XML 列的映射，定义要包含 XML 数据的表、辅助表和列。

```
<?xml version="1.0"?>
<!DOCTYPE Order SYSTEM "c:\dxx\dad.dtd">
<DAD>
   <dtdid>c:\dxx\samples\dtd\getstart.dtd</dtdid>
   <validation>YES</validation>

   <Xcolumn>
     <table name="order_side_tab">
         <column name="order_key"
            type="integer"
            path="/Order/@key"
            multi_occurrence="NO"/>
         <column name="customer"
            type="varchar(50)"
            path="/Order/Customer/Name"
            multi_occurrence="NO"/>
     </table>
     <table name="part_side_tab">
         <column name="price"
            type="decimal(10,2)"
            path="/Order/Part/ExtendedPrice"
            multi_occurrence="YES"/>
     </table>
     <table name="ship_side_tab">
         <column name="date"
            type="DATE"
            path="/Order/Part/Shipment/ShipDate"
            multi_occurrence="YES"/>
     </table>

   </Xcolumn>

</DAD>
```

图 16. 用于 XML 列的样本 DAD 文件

DAD 文件: XML 集合 - SQL 映射

此 DAD 文件包含 SQL 语句，该语句指定要包含的 XML 数据的 DB2 表、列和条件。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:\dxx\dtd\dad.dtd">
<DAD>
<validation>NO</validation>
<Xcollection>
<SQL_stmt>SELECT o.order_key, customer_name, customer_email, p.part_key, color, quantity,
   price, tax, ship_id, date, mode from order_tab o, part_tab p,
   table (select substr(char(timestamp(generate_unique())),16)
     as ship_id, date, mode, part_key from ship_tab) s
         WHERE o.order_key = 1 and
               p.price > 20000 and
               p.order_key = o.order_key and
               s.part_key = p.part_key
         ORDER BY order_key, part_key, ship_id</SQL_stmt>
      <prolog>?xml version="1.0"?</prolog>
<doctype>!DOCTYPE Order SYSTEM "c:\dxx\samples\dtd\getstart.dtd"</doctype>
```

图 17. 使用 SQL 映射的 XML 集合的样本 DAD 文件 (1/2)

```
      <root_node>
         <element_node name="Order">
  <attribute_node name="key">
                 <column name="order_key"/>
              </attribute_node>
           <element_node name="Customer">
    <element_node name="Name">
      <text_node><column name="customer_name"/></text_node>
  </element_node>
    <element_node name="Email">
      <text_node><column name="customer_email"/></text_node>
  </element_node>
              </element_node>
           <element_node name="Part">
    <attribute_node name="color">
      <column name="color"/>
              </attribute_node>
    <element_node name="key">
      <text_node><column name="part_key"/></text_node>
  </element_node>
<element_node name="Quantity">
      <text_node><column name="quantity"/></text_node>
  </element_node>
           <element_node name="ExtendedPrice">
      <text_node><column name="price"/></text_node>
  </element_node>
           <element_node name="Tax">
      <text_node><column name="tax"/></text_node>
  </element_node>
    <element_node name="Shipment" multi_occurence="YES">
              <element_node name="ShipDate">
        <text_node><column name="date"/></text_node>
    </element_node>
              <element_node name="ShipMode">
        <text_node><column name="mode"/></text_node>
  </element_node>
    </element_node>
    </element_node>
    </element_node>
      </root_node>
  </Xcollection>
</DAD>
```

图 17. 使用 SQL 映射的 XML 集合的样本 DAD 文件 (2/2)

DAD 文件: XML - RDB_node 映射

此 DAD 文件使用 <RDB_node> 元素来定义要包含 XML 数据的 DB2 表、列和条件。

```
<?xml version="1.0"?>
<!DOCTYPE Order SYSTEM "c:\dxx\dtd\dad.dtd">
<DAD>
   <dtdid>c:\dxx\samples\dtd\getstart.dtd</dtdid>
   <validation>YES</validation>
<Xcollection>
      <prolog>?xml version="1.0"?</prolog>
      <doctype>!DOCTYPE Order SYSTEM "c:\dxx\samples\dtd\getstart.dtd"</doctype>
      <root_node>
         <element_node name="Order">
                     <RDB_node>
                     <table name="order_tab"/>
                     <table name="part_tab"/>
                        <table name="ship_tab"/>
                        <condition>
                 order_tab.order_key = part_tab.order_key AND
                     part_tab.part_key = ship_tab.part_key
                        </condition>
                     </RDB_node>
  <attribute_node name="key">
                     <RDB_node>
                     <table name="order_tab"/>
                  <column name="order_key"/>
                     </RDB_node>
               </attribute_node>
           <element_node name="Customer">
                  <text_node>
                     <RDB_node>
                     <table name="order_tab"/>
                     <column name="customer"/>
                     </RDB_node>
                  </text_node>
               </element_node>
           <element_node name="Part">
                     <RDB_node>
                     <table name="part_tab"/>
                        <table name="ship_tab"/>
                        <condition>
                     part_tab.part_key = ship_tab.part_key
                        </condition>
                     </RDB_node>
  <attribute_node name="key">
                     <RDB_node>
                     <table name="part_tab"/>
                     <column name="part_key"/>
                     </RDB_node>
               </attribute_node>
```

图 18. 使用 RDB_node 映射的 XML 集合的样本 DAD 文件 (1/3)

```
<element_node name="Quantity">
      <text_node>
         <RDB_node>
         <table name="part_tab"/>
         <column name="quantity"/>
         </RDB_node>
      </text_node>
   </element_node>
<element_node name="ExtendedPrice">
      <text_node>
         <RDB_node>
         <table name="part_tab"/>
            <column name="price"/>
            <condition>
               price > 2500.00
            </condition>
         </RDB_node>
      </text_node>
   </element_node>
<element_node name="Tax">
      <text_node>
         <RDB_node>
         <table name="part_tab"/>
         <column name="tax"/>
         </RDB_node>
      </text_node>
   </element_node>
```

图 18. 使用 RDB_node 映射的 XML 集合的样本 DAD 文件 (2/3)

```
            <element_node name="shipment">
                    <RDB_node>
                       <table name="ship_tab"/>
                       <condition>
                    part_key = part_tab.part_key
                       </condition>
                    </RDB_node>
               <element_node name="ShipDate">
                  <text_node>
                     <RDB_node>
                        <table name="ship_tab"/>
                        <column name="date"/>
                        <condition>
                           date > "1966-01-01"
                        </condition>
                     </RDB_node>
                  </text_node>
               </element_node>
               <element_node name="ShipMode">
                  <text_node>
                     <RDB_node>
                        <table name="ship_tab"/>
                        <column name="mode"/>
                     </RDB_node>
                  </text_node>
               </element_node>
               <element_node name="Comment">
                  <text_node>
                     <RDB_node>
                        <table name="ship_tab"/>
                        <column name="comment"/>
                     </RDB_node>
                  </text_node>
               </element_node>
             </element_node> <! -- end of element Shipment>
           </element_node> <! -- end of element Part --->
         </element_node> <! -- end of element Order --->
       </root_node>
  </Xcollection>
</DAD>
```

图 18. 使用 RDB_node 映射的 XML 集合的样本 DAD 文件 (3/3)

附录C. 代码页注意事项

对于访问 XML 文档和其他相关文件的每个客户机或服务器，确保对这些文件正确编码是很重要的。因此，在处理文件时知道 XML Extender 假设，以及它然后如何处理代码页转换也很重要。主要的注意事项有:

- 确保从 DB2 检索 XML 文档的客户机的实际代码页与该文档的编码匹配。
- 确保当 XML 语法分析程序处理文档时，该 XML 文档的编码说明也与编码一致。

下列章节描述有关这些注意事项的问题: 您如何为可能的问题进行准备，以及当将文档从客户机传送至服务器和数据库时， XML Extender 和 DB2 如何支持代码页。

术语

本节使用下列术语:

文档编码
XML 文档的实际代码页。

文档编码说明
XML 说明中所指定的代码页的名称。例如:

```
<?xml version="1.0" encoding="ibm037"?>
```

一致的文档
其文档编码与文档编码说明代码页相匹配的文档。

不一致的文档
其文档编码与文档编码说明代码页不匹配的文档。

DB2CODEPAGE 注册表（环境）变量
指定从数据库客户机应用程序呈示给 DB2 的数据的代码页。除非设置此变量，否则 DB2 从客户机操作系统语言环境获得客户机的代码页。对于 DB2，若设置此值，则它将覆盖客户机操作系统的语言环境。

客户机代码页
应用程序代码页。若设置了 DB2CODEPAGE 变量，则客户机代码页的值为 DB2CODEPAGE。否则，客户机代码页是客户机的操作系统语言环境。

服务器代码页，或服务器操作系统语言环境代码页
在其上安装了 DB2 数据库的操作系统语言环境。

© Copyright IBM Corp. 1999, 2000

数据库代码页:

存储的数据的编码，它在创建数据库时确定。若未使用 USING CODESET 子句显式指定此值，则此值缺省为服务器操作系统的语言环境。

DB2 和 XML Extender 代码页假设

DB2 在接收或发送 XML 文档时，它不会检查编码说明。而是检查客户机的代码页以查看它是否与数据库代码页相匹配。如果两者不同，DB2 会将 XML 文档中的数据转换为与下列项的代码页匹配:

- 数据库（在将文档或文档片段导入数据库表时）
- 数据库（在将文档分解为一个或多个数据库表时）
- 客户机（在从数据库导出文档并将该文档呈示给客户机时）
- 服务器（在处理带有 UDF 的文件时，该 UDF 返回服务器文件系统上某个文件中的数据）

当将 XML 文档导入数据库中时，通常将它作为 XML 文档导入以存储在 XML 列中，或作为 XML 集合导入进行分解，其中的元素和属性内容将作为 DB2 数据保存。当导入文档时， DB2 将该文档的编码转换为数据库的编码。DB2 假设该文档使用下表中的“源代码页”列中所指定的代码页。表55总结了 DB2 在导入 XML 文档时所执行的转换。

表 55. 当将 XML 文件导入数据库中时，使用 UDF 和存储过程

处理方法	转换的源代码页	转换的目的代码页	注释
将 DTD 文件插入到 DTD_REF 表中	客户机代码页	数据库代码页	
启用列或启用用于导入 DAD 文件的集合存储过程或管理命令	客户机代码页	数据库代码页	
UDF: • XMLVarcharFromFile() • XMLCLOBFromFile() • Content(): 从 XMLFILE 至 CLOB 的检索	服务器代码页	数据库代码页	
分解存储过程	客户机代码页	数据库代码页	假设要分解的文档使用客户机代码页。来自分解的数据存储在使用数据库代码页的表中。

当从数据库导出 XML 文档时，通常是基于呈示文档的客户机请求； XML 文档内容的查询来导出该文档；或在从 DB2 数据组合文档时导出该文档。当导出文档时，根据请求的来源和数据要呈示的位置，DB2 将该文档的编码转换为客户机或服务器的编码。表56 总结了 DB2 在导出 XML 文档时所执行的转换。

表 56. 当从数据库导出 XML 文件时，使用 UDF 和存储过程

处理方法	转换	注释
UDF • XMLFileFromVarchar() • XMLFileFromCLOB()	在向客户机呈示数据时，数据库代码页至客户机代码页	
UDF • Content(): 从 XMLVARCHAR 至外部服务器文件的检索导出它。	数据库代码页至服务器代码页	
组合存储过程: 产生的表存储在结果表中，可以查询和导出它。	在向客户机呈示结果集时，数据库代码页至客户机代码页	当组合文档时， XML Extender 将由 DAD 中的标记所指定的编码说明复制到新创建的文档中。当呈示它时，它应与客户机代码页匹配。

编码说明注意事项

编码说明说明了 XML 文档的编码，且出现在 XML 说明语句中。当使用 XML Extender 时，重要的是根据文件所在的位置，确保文档的编码与客户机或服务器的代码页相匹配。

合法编码说明

只要遵循某些准则，您可以在 XML 文档中使用任何编码说明。在本节中定义了这些准则和受支持的编码说明。

根据 XML 规范，建议的可移植 XML 数据编码有 UTF-8 和 UTF-16。若您使用这些编码，则您的应用程序在不同的公司之间是可相互操作的。若您使用列示在第250页的表57中的编码，则您的应用程序可能在 IBM 操作系统之间是可移植的。若您使用其他编码，则您的数据可移植的可能性较小。

对于所有操作系统，下列编码说明都受支持。以下列表描述每列的含义:

- **编码**指定 XML 说明中要使用的编码字符串。

- **OS** 显示在其上 DB2 支持给定代码页的操作系统。
- **代码页**显示 IBM 定义的与给定编码相关的代码页

表 57. XML Extender 所支持的编码说明

类别	编码	OS	代码页
Unicode	UTF-8	AIX, SUN, Linux	1208
	UTF-16	AIX, SUN, Linux	1200
ASCII	iso-8859-1	AIX, Linux, Sun	819
	ibm-1252	Windows NT	1252
	iso-8859-2	AIX, Linux, Sun	912
	iso-8859-5	AIX, Linux	915
	iso-8859-6	AIX	1089
	iso-8859-7	AIX, Linux	813
	iso-8859-8	AIX, Linux	916
	iso-8859-9	AIX, Linux	920
MBCS	gb2312	Windows NT	1386
	ibm-932, shift_jis78	AIX	932
	Shift_JIS	AIX 4.3 only, Windows NT	943
	IBM-eucCN	AIX, Sun	1383
	IBM-eucJP, EUC-JP	AIX, Linux, Sun	954, 33722
	euc-tw, IBM-eucTW	AIX, Sun	964
	euc-kr, IBM-eucKR	AIX	970
	big5	AIX, Sun, Windows NT	950

编码字符串必须与文档目的地的代码页兼容。若正在将文档从服务器返回客户机，则其编码字符串必须与客户机的代码页兼容。有关不兼容编码的后果，参见第251页的『一致的编码和编码说明』。参见以下 URL，以获得由 XML Extender 使用的 XML 语法分析程序所支持的代码页列表:

```
http://www.ibm.com/software/data/db2/extenders/xmlext/moreinfo/encoding.html
```

一致的编码和编码说明

当处理 XML 文档或与另一个系统交换 XML 文档时，编码说明与文档的实际编码相对应很重要。确保文档编码与客户机代码页一致非常重要，因为 XML 工具（如语法分析程序）会对包括编码说明的实体而不是在说明中命名的实体产生错误。

图19显示客户机具有与文档编码和说明的编码相一致的代码页。

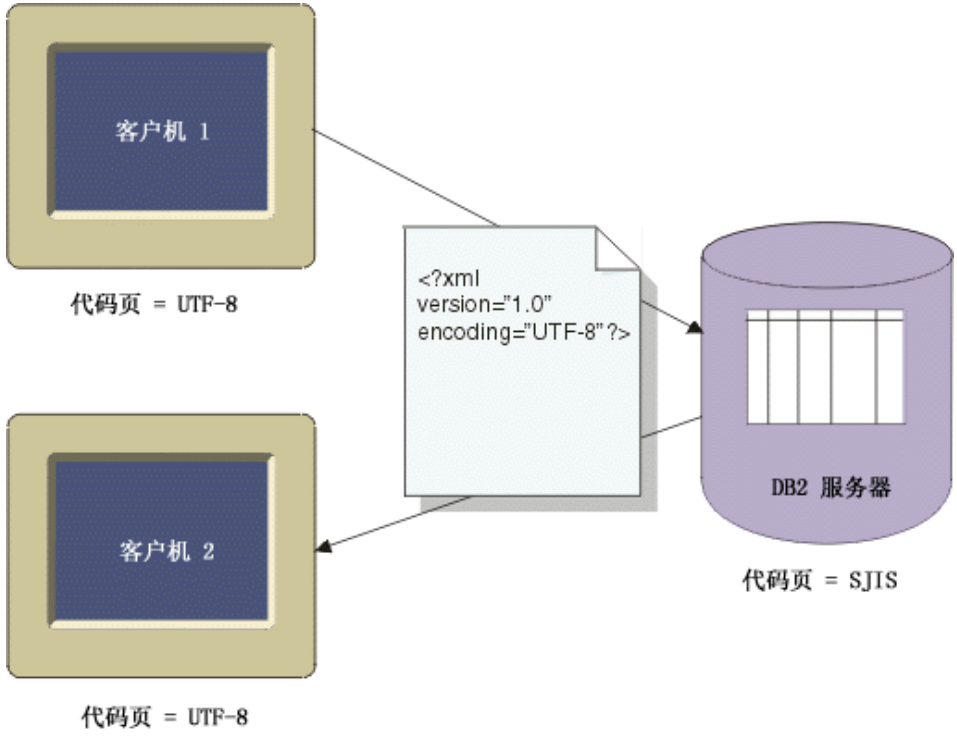


图 19. 客户机具有匹配的代码页

使用不同的代码页可能会导致下列情况:

- 可能会发生进行转换时数据丢失的情况，尤其是当源代码页为 Unicode，而目标代码页不是 Unicode 时。Unicode 包含全套的字符转换集。若将文件从 UTF-8 转换为某个代码页，而该代码页并非文档中所使用的所有字符都支持，则在转换期间可能会丢失数据。
- XML 文档的说明编码可能不再与实际文档编码一致（如果该文档是由客户机检索，而该客户机使用的代码页与该文档的说明编码不同）。

第252页的图20显示了一种环境，该环境中的客户机代码页不一致。

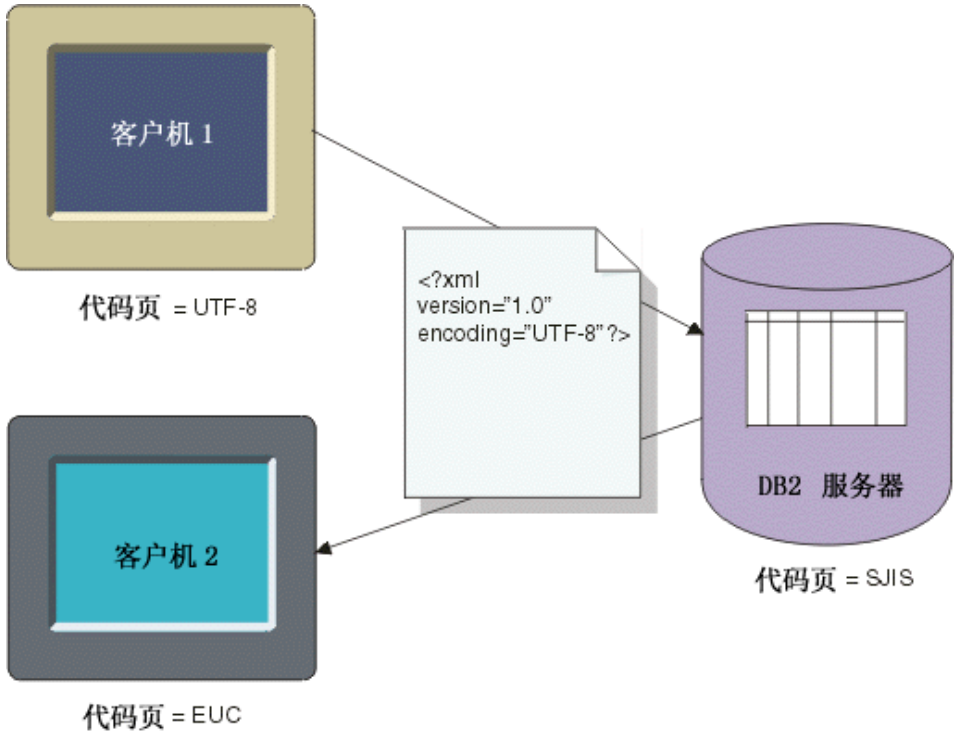


图 20. 客户机具有不匹配的代码页

客户机 2 接收到 EUC 中的文档，但该文档将具有编码说明 UTF-8。

说明编码

编码说明的缺省值是 UTF-8，无编码说明意味着文档属性为 UTF-8。

要说明编码值:

在 XML 文档说明中，使用客户机代码页的名称来指定编码说明。例如:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

转换方案

在下列情况时，XML Extender 处理 XML 文档:

- 使用 XML 列存储和访问方法，存储和检索 XML 列数据
- 组合和分解 XML 文档

当从客户机或服务器向数据库传送文档时，文档将进行代码页转换。在从客户机、服务器和数据库的代码页进行转换时，最可能发生 XML 文档的不一致或损

坏。当选择文档的编码说明以及计划可以从数据库导入或导出文档的客户机和服务器时，考虑上表中所描述的转换以及下述方案。

下列方案描述可能发生的公共转换方案:

方案 1: 此方案是这样配置的，它具有一致的编码、无 DB2 转换并从服务器导入文档。文档编码说明是 UTF-8，服务器是 UTF-8，数据库是 UTF-8。

1. 使用 XMLClobfromFile UDF 将文档导入 DB2。
2. 将文档抽取到服务器。
3. 因为服务器代码页和数据库代码页是相同的，所以 DB2 无需转换文档。编码和说明是一致的。

方案 2: 此方案是这样配置的，它具有一致的编码、进行 DB2 转换、从服务器导入文档并导出到客户机。文档编码和说明是 SJIS、客户机代码页是 SJIS，服务器和数据库代码页是 UTF-8。

1. 使用 XMLClobfromfile UDF 将文档从服务器导入 DB2。
2. DB2 从 SJIS 转换文档并用 UTF-8 格式存储它。数据库中的编码说明和编码不一致。
3. 使用 SJIS 的客户机请求该文档以在 Web 浏览器中呈示。
4. DB2 将文档转换为 SJIS，即客户机的代码页。现在客户机上的文档编码和说明已一致。

方案 3: 此方案是这样配置的，它具有不一致编码、进行 DB2 转换、从服务器导入文档并导出到客户机。对于入局文档，文档编码说明是 SJIS。服务器代码页是 SJIS，客户机和数据库是 UTF-8。

1. 使用存储器 UDF 将文档导入数据库。
2. DB2 将文档从 SJIS 转换为 UTF-8。编码和说明不一致。
3. 具有 UTF-8 代码页的客户机请求该文档以在 Web 浏览器上呈示。
4. 因为客户机和数据库的代码页是相同的，所以 DB2 不进行转换。
5. 因为说明是 SJIS，而编码为 UTF-8，所以文档编码和说明不一致。不能使用 XML 语法分析程序或其他 XML 处理工具来处理该文档。

方案 4: 此方案是这样配置的，它会丢失数据、进行 DB2 转换并从 UTF-8 服务器导入文档: 文档编码说明是 UTF-8，服务器是 UTF-8，数据库是 SJIS。

1. 使用 XMLClobfromFile UDF 将文档导入 DB2。
2. DB2 将编码转换为 SJIS。当导入文档时，存储在数据库中的文档可能已损坏，因为用 UTF-8 表示的字符可能没有 SJIS 表示。

方案 5:

此方案是带 Windows NT 限制的配置。在 Windows NT 上，不能将操作系统语言环境设置为 UTF-8，但是，DB2 允许客户机使用 `db2set DB2CODEPAGE=1208` 将代码页设置为 UTF-8。在此方案中，客户机和服务器位于同一机器上。客户机是 UTF-8，但不能将服务器设置为 UTF-8；其代码页是 1252。该文档编码为 1252，编码说明为 ibm-1252。数据库代码页是 UTF-8。

1. 存储器 UDF 将文档从服务器导入并将它从 1252 转换为 1208。
2. 客户机使用 Content() UDF 将文档从 DB2 导出。
3. DB2 将文档从 UTF-8 转换为1252，即使客户机可能期望 1208 也是如此，因为该客户机与服务器在同一系统上，并被设置为 1208。

防止不一致 XML 文档

上面的章节讨论了 XML 文档在什么情况下才可能具有不一致的编码，也就是编码说明与文档编码冲突。不一致的编码可导致数据丢失和／或不可用的 XML 文档。

在将文档交给 XML 处理程序（如语法分析程序）进行处理之前，使用下列建议之一以确保 XML 文档编码与客户机代码页一致。

- 当使用 XML Extender UDF 将文档从数据库导出时，尝试下列技巧之一：（假定：XML Extender 已将使用服务器代码页的文件导出到服务器上的文件系统）。
  - 将文档转换为说明编码代码页
  - 覆盖说明编码（如果该工具具有覆盖功能的话）
  - 人工将导出的文档的编码说明更改为文档的实际编码（即服务器代码页）
- 当使用 XML Extender 存储过程将文档从数据库导出时，尝试下列技巧之一：（假设客户机正在查询存储组合文档的结果表）
  - 将文档转换为说明编码代码页
  - 覆盖说明编码（如果该工具具有覆盖功能的话）
  - 在查询结果表之前，使客户机设置环境变量 DB2CODEPAGE，以强制客户机代码页为与 XML 文档的编码说明兼容的代码页。
  - 人工将导出的文档的编码说明更改为文档的实际编码（即客户机代码页）
- **使用 Unicode 和 Windows NT 时的限制**：在 Windows NT 上，不能将操作系统语言环境设置为 UTF-8。在导入或导出文档时，使用下列准则:
  - 当导入用 UTF-8 编码的文件和 DTD 时，使用以下命令将客户机代码页设置为 UTF-8:

    ```
    db2set DB2CODEPAGE=1208
    ```

在下列情况下使用此技巧:

- 将 DTD 插入 db2xml.DTD_REF 表中
- 启用列或集合
- 分解存储过程

– 当使用 Content() 或 XMLxxxfromFile UDF 来导入 XML 文档时，必须用服务器操作系统语言环境的代码页来编码文档，该代码页不能为 UTF-8。
– 当从数据库导出 XML 文件时。使用以下命令设置客户机代码页，以使 DB2 用 UTF-8 编码结果数据:

```
db2set DB2CODEPAGE=1208
```

在下列情况下使用此技巧:

- 在组合之后查询结果表
- 使用抽取 UDF 从 XML 列抽取数据

– 当使用 Content() 或 XMLxxxfromFile UDF 来将 XML 文档导出到服务器文件系统上的文件时，必须用服务器操作系统语言环境的代码页来编码结果文档，该代码页不能为 UTF-8。

附录D. XML Extender 限制

XML Extender DAD 文件、存储过程和表具有下列限制。

表 58. XML Extender 限制

值	限制
分解 XML 集合中的表	来自每个分解的 XML 文档的 1024 行
存储过程参数:	
XML 文档 CLOB	1 MB[2]
文档访问定义 (DAD) CLOB	100KB[2]
collectionName	30
colName	30
dbName	18
defaultView	128
rootID	128
resultTabName	128
tablespace	128
tbName	128[1]
db2xml.DTD_REF 表列	
AUTHOR	128
CREATOR	128
UPDATOR	128
DTDID	128
CLOB	100 KB
注: 1. 若通过模式名限定 *tbName*，则整个名称（包括分隔符）不得长于 128 个字符。 2. 可以更改此大小。要了解如何更改，参见第186页的『增加 CLOB 限制』。	

当 DB2 将名称从客户机代码页转换为数据库代码页时，可以将这些名称进行扩展。名称可能适合客户机的大小限制，但它在存储过程获得转换的名称时会超过限制。有关详情，参见 *DB2 Application Development Guide* 的『Programming in Complex Environments』这一章中的『National Language Support Application Development』节。

© Copyright IBM Corp. 1999, 2000

注意事项

本资料是为在美国提供的产品和服务开发的。 IBM 可能不在其他国家提供本文档中讨论的产品、服务或功能部件。有关您所在地区当前可用的产品和服务的信息，向您的本地 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，都可以用来代替 IBM 产品、程序或服务。但是，评估和验证任何非 IBM 的产品、程序或服务的操作由用户负责。

IBM 可能已经申请或正在申请与本文档有关的各项专利权。提供本文档并不表示允许您使用这些专利。您可以用书面方式将特许查询寄往:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

有关双字节 (DBCS) 信息的特许查询，与您国家的 “IBM 知识产权部” 联系，或以书面方式将查询寄往:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

下列短文不适用于英国或这种规定与当地法律不一致的任何国家: 国际商用机器公司『按原样』提供此出版物，但不作任何明确或隐含的保证，包括（但不限于）商业性的隐含保证或符合特殊目的。在某些交易中，一些国家不允许否认明确的或隐含的担保，因此，此说明可能不适用于您。

本资料可能包含技术上的不准确性或印刷错误。本书中的信息会定期更改；这些更改将会合并在此出版物的新版本中。 IBM 可能对本出版物中描述的产品和/或程序随时作出改进和/或更改，而不会发出通知。

为了以下目的: (1) 允许在独立建立的程序和其它程序（包括本程序）之间进行信息交换和 (2) 允许对已经交换的信息进行相互使用，希望获取有关本程序信息的合法用户应联系:

IBM Corporation
J74/G4

© Copyright IBM Corp. 1999, 2000

555 Bailey Avenue
P.O. Box 49023
San Jose, CA 95161-9023
U.S.A.

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，可获取这方面的信息。

本资料中描述的特许程序和所有该特许程序可用的特许资料，由 IBM 依据 “IBM 客户协议”、 “IBM 国际程序使用许可协议”或者我们之间的任何等效的协议来提供。

关于非 IBM 产品的信息是从那些产品的供应商、他们发布的说明或其他公用的来源获得的。 IBM 没有测试那些产品，因此不能确认性能、兼容性或任何其他与非 IBM 产品相关的说明的准确性。有关非 IBM 产品性能的疑问应向那些产品的供应商提出。

所有关于 IBM 未来方向或意向的说明都可能更改或撤消，而不作任何通知，并且仅代表目标和结果。

版权许可证:

本资料包含用源语言编写的样本应用程序，以示范在各种操作平台上的程序设计技术。为了开发、使用、经销或分发应用程序，而这些应用程序须符合对其编写了样本程序的操作平台的应用程序设计接口，您可以以任何形式复制、修改和分发这些样本程序，而无需向 IBM 付款。这些示例未在所有条件下经过彻底测试。因此，IBM 不能保证或暗示这些程序的可靠性、可服务性或功能。

商标

以下各项是国际商用机器公司在美国或其他国家的商标:

DB2	Net.Data
DB2 Extender	OS/2
DB2 Universal Database	OS/390
IBM	OS/400
IMS	VTAM

Java 和所有基于 Java 的商标和标志是 Sun Microsystems 公司在美国和／或其他国家的商标或注册商标。

Microsoft、Windows、Windows NT 和 Windows 标志是 Microsoft 公司在美国和／或其他国家的商标。

UNIX 是经 X/Open 有限公司专门许可的在美国和／或其它国家的注册商标。

其他公司、产品和服务名可能是其他公司的商标或服务标记。

词汇表

API. 参见*应用程序设计接口 (application programming interface)*。

attribute_node. 元素的属性的一种表示法。

B 型树索引 (B-tree indexing). DB2 引擎提供的本机索引模式。它构建采用 B 型树结构的索引项。支持 DB2 基本数据类型。

CLOB. 字符大对象。

DAD. 参见*文档访问定义 (Document access definition)*。

DBCLOB. 双字节字符大对象。

DTD. (1) . (2) 参见*文档类型定义 (Document type definition)*。

DTD_REF 表 (DTD_REF table). DTD 参考表。

DTD 参考表 （DTD_REF 表）(DTD reference table)（DTD_REF table）. 包含了 DTD 的表，可用来验证 XML 文档，以及帮助应用程序定义 DAD。用户可以将他们自己的 DTD 插入 DTD_REF 表中。此表是在为 XML 启用数据库时创建的。

DTD 库(DTD repository). 一种称为 DTD_REF 的 DB2 表，其中，该表的每一行都表示一个具有附加元数据信息的 DTD。

EDI. Electronic Data Interchange 的缩写，电子数据交换。

element_node. 元素的一种表示法。element_node 可以是根元素或子元素。

Java 数据库链接 (Java Database Connectivity) (JDBC). 与开放式数据库链接 (ODBC) 具有相同特性的应用程序设计接口 (API)，但是，它是专门设计用来供 Java 数据库应用程序使用的。另外，对于没有 JDBC 驱动程序的数据库，JDBC 包括从 JDBC 至 ODBC 的网桥，它是用来将 JDBC 转换为 ODBC 的机制；JDBC 为 Java 数据库驱动程序提供 JDBC API，并将它转换为 ODBC。JDBC 是由 Sun Microsystems公司以及各个合作伙伴和供应商开发的。

JDBC. Java Database Connectivity 的缩写，Java 数据库链接。

LOB. 大对象。

ODBC. Open Database Connectivity（开放式数据库链接）的缩写。

RDB_node. 关系数据库节点。

RDB_node 映射 (RDB_node mapping). XML 元素的内容或 XML 属性的值的定位，它们是由 RDB_node 定义的。XML Extender 使用此映射来确定在何处存储或检索 XML 数据。

Root 用户标识 (root ID). 用来将所有辅助表与应用程序表关联起来的唯一的标识符。

SQL 映射 (SQL mapping). 通过使用一个或多个 SQL 语句和 XSLT 数据模型，来定义 XML 元素的内容的关系或者具有关系数据的 XML 属性的值。XML Extender 使用该定义来确定在何处存储或检索 XML 数据。SQL 映射是使用 DAD 中的 SQL_stmt 元素来定义的。

text_node. 元素的 CDATA 文本的表示法。

UDF. 参见*用户定义函数 (user-defined function)*。

UDT. 参见*用户定义类型 (user-defined type)*。

UNION. 一种 SQL 操作，它将两个选择语句的结果组合起来。通常使用 UNION 将从几个表中得到的值的列表合并在一起。

© Copyright IBM Corp. 1999, 2000

URL. 统一资源定位器。

Web 浏览器 (Web browser). 一个客户机程序，它启动对 Web 服务器的请求，并显示服务器所返回的信息。

XML. 可扩展标记语言。

XML UDF. 由 XML Extender 提供的 DB2 用户定义函数。

XML UDT. 由 XML Extender 提供的 DB2 用户定义类型。

XML 标记 (XML tag). 任何有效的 XML 标记语言标记，主要是指 XML 元素。可交换使用术语标记和元素。

XML 表 (XML table). 包括一个或多个 XML Extender 列的应用程序表。

XML 对象 (XML object). 等效于 XML 文档。

XML 集合 (XML collection). 关系表的集合，它提供数据以组合 XML 文档，或者从 XML 文档中分解数据。

XML 列 (XML column). 已经为 XML Extender UDT 启用的应用程序表中的列。

XML 路径语言 (XML Path Language). 用来对 XML 文档的部分进行寻址的语言。“XML 路径语言”是专门设计以供 XSLT 使用的。可以使用为 XPath 定义的语法来表示每个位置路径。

XML 属性 (XML attribute). 在 DTD 中的 XML 元素下面，由 ATTLIST 指定的任何属性。XML Extender 使用位置路径来标识属性。

XML 元素 (XML element). 在 XML DTD 中指定的任何 XML 标记或 ELEMENT。XML Extender 使用位置路径来标识元素。

XPath. 用来对 XML 文档的部分进行寻址的语言。

XPath 数据模型 (XPath data model). 用来使用节点来创建 XML 文档的模型和浏览 XML 文档的树结构。

XSL. XML 样式表语言。

XSLT. XML 样式表语言变换。

本地文件系统 (local file system). 存在于 DB2 中的文件系统。

标量函数 (scalar function). 从另一个值中产生单个值的 SQL 操作，表示为一个后面跟着用括号括起来的自变量列表的函数名。

表空间 (table space). 容器集合的一个子集，数据库对象存储在其中。表空间在数据库与该数据库中存储的表之间提供了一定程度的间接处理。表空间:

- 在分配给它的媒体存储设备上拥有空间。
- 其中创建有表。这些表将消耗属于表空间的容器中的空间。数据、索引、长字段和表的 LOB部分可以存储在同一表空间中，也可以个别地分散到单独的表空间中。

部分搜索 (section search). 对一个部分提供文本搜索，可以由应用程序来定义这一部分。要支持结构化文本搜索，可以由 Xpath 的缩写位置路径来定义这一部分。

查询 (query). 根据特定的条件来请求数据库中的信息；例如，查询可能请求获取客户表中余额大于 1000 美元的所有客户的列表。

存储过程 (stored procedure). 过程结构与嵌入式 SQL 语句的块，存储在数据库中，可以通过名称调用。存储过程使应用程序可能分成两部分运行。一部分在客户机上运行，另一部分在服务器上运行。这使得只需调用一次过程，便可多次访问数据库。

大对象 (large object) (LOB). 字节序列，其长度最大可达 2GB。 LOB 可以是三种类型: *二进制大对象* (BLOB)、*字符大对象 (CLOB)* 或*双字节字符大对象 (DBCLOB)*。

单值类型 (distinct type). 参见*用户定义类型 (user-defined type)*。

电子数据交换 (Electronic Data Interchange) (EDI). 商务应用程序 (B2B) 之间进行电子数据交换的标准。

顶部 element_node (top element_node). DAD 中的 XML 文档的根元素的表示法。

定位器 (locator). 可用来定位对象的指针。在 DB2 中，大对象块 (LOB) 定位器是用来定位 LOB 的数据类型。

对象 (object). 在面向对象的程序设计中，是一个由数据和与该数据相关联的操作组成的抽象概念。

多次出现 (multiple occurrence). 指示在一个文档中是否可以多次使用列元素或属性。多次出现是在 DAD 中指定的。

访问和存储方法 (access and storage method). 通过两种主要的访问和存储方法（XML 列和 XML 集合），将 XML 文档与 DB2 数据库关联。参见 *XML 列 (XML column)* 和 *XML 集合 (XML collection)*。

分解 (decompose). 将 XML 文档分隔成 XML 集合中的关系表的集合。

分区 (partition). 对存储器进行固定大小的划分。

辅助表 (side table). 由 XML Extender 创建的附加表，以便在搜索 XML 列中的元素或属性时提高性能。

根元素 (root element). XML 文档的顶部元素。

关系数据库节点 (relational database node) (RDB_node). 一种节点，它包含对表、可选列和可选条件的一个或多个元素的定义。使用表和列来定义如何在数据库中存储 XML 数据。该条件指定选择 XML 数据的标准或连接 XML 集合表的方法。

管理支持表 (administrative support tables). 一种表，DB2 Extender 用来处理用户对 XML 对象的请求。某些管理支持表标识对 Extender 启用的用户表和列。其他管理支持表包含关于已启用列中的对象的属性信息。与元数据表 (metadata table) 是同义词。

过程 (procedure). 参见*存储过程 (stored procedure)*。

过载函数 (overloaded function). 存在多个函数实例的函数名。

合式文档 (well-formed document). 不包含 DTD 的 XML 文档。即使采用 XML 规范，具有有效 DTD 的文档也必须是形式良好的。

简单位置路径 (simple location path). 通过单斜杠 (/) 连接的一序列元素类型名。

节点 (node). 在数据库分区中，与数据库分区 (database partition) 是同义词。

结构化文本索引 (structural text index). 通过使用 DB2 Text Extender，根据 XML 文档的树形结构来对文本字符串进行索引。

结果表 (result table). 一个表，它包含了作为 SQL 查询或执行存储过程的结果的行。

结果集 (result set). 由存储过程返回的一系列行。

静态 SQL (static SQL). 嵌入在程序中，在程序执行前的程序准备过程期间准备的 SQL 语句。准备之后，尽管静态 SQL 语句指定的主变量的值会更改，但是该静态 SQL 语句不更改。

绝对位置路径 (absolute location path). 对象的全路径名。绝对路径名从最高层或“根”元素开始，“根”元素是由正斜杠 (/) 或反斜杠 (\) 字符标识的。

开放式数据库链接 (Open Database Connectivity). 一种标准的应用程序设计接口 (API)，它用于访问关系数据库管理系统和非关系数据库管理系统中的数据。即使每个数据库管理系统都使用不同的数据存储格式和程序设计界面，通过使用此 API，数据库应用程序也可以访问存储在不同

计算机上的数据库管理系统中的数据。ODBC 基于 X/Open SQL Access Group 的调用层界面 (CLI) 规范，它是由 Digital Equipment Corporation (DEC)、Lotus、Microsoft 和 Sybase 公司开发的。与 *Java 数据库链接 (Java Database Connectivity)* 对照。

可扩展的样式表语言 (Extensible Stylesheet language) (XSL). 用来表示样式表的语言。XSL 由两部分组成: 用来转换 XML 文档的语言，用来指定格式化语义的 XML 词汇表。

可扩展的样式表语言变换 (Extensive Stylesheet Language Transformation) (XSLT). 用来将 XML 文档变换为其他 XML 文档的语言。XSLT 被用作 XSL 的一部分，它是 XML 的样式表语言。

连接 (join). 一种关系操作，它允许根据匹配列的值，从两个和多个表检索数据。

连接视图 (joined view). 由 ″CREATE VIEW″ 语句创建的 DB2 视图，它将多个表连接在一起。

列数据 (column data). 存储在 DB2 列中的数据。数据类型可以是 DB2 支持的任何数据类型。

浏览器 (browser). 参见 *Web 浏览器 (Web browser)*。

路径表达式 (path expression). 参见*位置路径 (location path)*。

模式 (schema). 数据库对象（如表、视图、索引或触发器）的集合。它提供对数据库对象的逻辑分类。

嵌入式 SQL (embedded SQL). 在应用程序中编码的 SQL 语句。参见*静态 SQL (static SQL)*。

强制转型函数 (cast function). 一种函数，用来将（源）数据类型的实例转换为另一种（目标）数据类型的实例。 通常，强制转型函数的名称就是目标数据类型的名称。它有一个类型为源数据类型的单个自变量；其返回类型是目标数据类型。

全文本搜索 (full text search). 使用 DB2 Text Extender，搜索任何地方的文本字符串，而不管文本结构如何。

缺省强制转型函数 (default casting function). 将 SQL 库类型强制转型为 UDT。

缺省视图 (default view). 一种数据表示法，在这种表示法中，XML 表与其所有相关辅助表都互相连接。

属性 (attribute). 参见 *XML 属性 (XML attribute)*。

数据交换 (data interchange). 共享应用程序之间的数据。XML 支持数据交换，而不需要首先将数据从所有者格式进行转换。

数据类型 (data type). 列和文字的属性。

数据链接 (datalink). 一种 DB2 数据类型，它允许从数据库中，对存储在该数据库外部的文件进行逻辑引用。

数据源 (data source). 本地或远程关系或非关系数据管理程序，它能够支持通过支持 ODBC API 的 ODBC 驱动程序来进行数据访问。

双字节字符大对象 (double-byte character large object) (DBCLOB). 双字节字符的字符串，或者是单字节字符与双字节字符的组合，其中字符串最多可达 2 GB。DBCLOB 具有相关联的代码页。包括双字节字符的文本对象作为 DBCLOB 存储在 DB2 数据库中。

索引 (index). 按键值逻辑排序的一个指针集。索引提供对数据的快速访问，并可以增强表中各行的唯一性。

条件 (condition). 选择 XML 数据的标准的规范，或连接 XML 集合表的方法的规范。

统一资源定位器 (uniform resource locator) (URL). 一个地址，用来指定 HTTP 服务器的名称并有选择地指定目录和文件的名称，例如，http://www.ibm.com/data/db2/extenders。

外部文件 (external file). 存在于 DB2 外部的文件系统中的一种文件。

外键 (foreign key). 一种键，它是参考约束的定义的一部分、并且由从属表的一列或多列组成。

位置路径 (location path). 位置路径是用来标识 XML 元素或属性的 XML 标记序列。位置路径标识 XML 文档的结构，指示元素或属性的上下文。单斜杠 (/) 路径指示上下文是整个文档。在抽取 UDF 中使用位置路径来标识要抽取的元素和属性。在 DAD 文件中还使用位置路径来指定 XML 元素或属性，以及 DB2 列（当定义 XML 列的创建索引模式时）之间的映射。另外，Text Extender 使用位置路径来进行结构化文本搜索。

谓词 (predicate). 搜索条件的一个元素，表示或暗示一个比较运算。

文档访问定义 (Document Access Definition)(DAD). 用来定义 XML 列的创建索引模式或 XML 集合的映射模式。它可以用来启用 XML 集合的 XML Extender 列（它是 XML 格式的）。

文档类型定义 (Document type definition)(DTD). 对 XML 元素和属性的一组说明。DTD 定义在 XML 文档中使用哪些元素、可以按什么次序使用它们、以及哪些元素可以包含其他元素。可以将 DTD 与文档访问定义 (DAD) 文件关联，以验证 XML 文档。

验证 (validation). 使用 DTD 来确保 XML 文档有效、并允许对 XML 数据进行结构化搜索的进程。DTD 是存储在 DTD 库中。

应用程序设计接口（API）(application programming interface, API).

(1) 操作系统或可单独订购的特许程序所提供的功能接口。API 允许用高级语言编写的应用程序使用操作系统或特许程序的特定数据或函数。

(2) 在 DB2 中是接口中的一种函数，例如，获取错误消息 API。

(3) 在 DB2 中，为接口内的一个功能。例如，获取错误消息 API。

映射模式 (mapping scheme). 定义在关系数据库中如何表示 XML 数据。映射模式是在 DAD 中指定的。XML Extender 提供两种类型的映射模式：*SQL 映射*和*关系数据库节点 (RDB_node) 映射*。

用户表 (user table). 为应用程序创建的表，并供应用程序使用。

用户定义函数 (UDF) (user-defined function (UDF)). 这是对数据库管理系统定义的函数，此后可在 SQL 查询中引用。它可以是下列函数之一：

- 外部函数，其中的函数主体是用程序设计语言编写的，该函数的自变量为标量值，并且每次调用时都会产生一个标量结果。
- 有源函数，由 DBMS 已知的另一内部函数或用户定义函数来实现。此函数可以是标量函数或列（聚合）函数，并从一组值返回单一值（例如，MAX 或 AVG）。

用户定义类型 (user-defined type) (UDT). 一种数据类型，它对于数据库管理程序来说，不是本机数据类型，该类型是用户创建的。参见*单值类型 (distinct type)*。

有效文档 (valid document). 具有相关联的 DTD 的 XML 文档。要使其有效，XML 文档不能违反在其 DTD 中指定的语法规则。

元数据表 (metadata table). 参见*管理支持表 (administrative support table)*。

元素 (element). 参见 *XML 元素 (XML element)*。

主键 (primary key). 作为表定义一部分的唯一键。主键是参考约束定义的缺省父键。

子查询 (subquery). 在 SQL 语句的搜索条件内使用的完整 SELECT 语句。

字符大对象 (character large object) (CLOB). 单字节字符的字符串，该字符串最多可达 2

GB。CLOB 具有相关联的代码页。包含单字节字符的文本对象作为 CLOB 存储在 DB2 数据库中。

组合 (compose). 从 XML 集合中的关系数据中生成 XML 文档。

索引

© Copyright IBM Corp. 1999, 2000

[D]

[F]

[G]

[H]

[J]

[K]

[L]

[M]

[Q]

[R]

[S]

[T]

[W]

[X]

[Y]

[Z]

[特别字符]

A

B

C

D

E

F

J

M

O

R

S

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，或与任何授权的 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、产品描述、培训计划等等的当前 DB2 信息。

http://www.ibm.com/software/data/db2/library/

DB2 Product and Service Technical Library 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/

Professional Certification Program from the IBM Web 站点提供各种 IBM 产品（包括 DB2）的认证测试信息。

ftp.software.ibm.com

以匿名形式登录。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。

© Copyright IBM Corp. 1999, 2000

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 授权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。

中国印刷