IBM DB2 UDB iSeries 版

XML Extender 管理与编程

版本 8

S152-0843-00

IBM DB2 UDB iSeries 版

XML Extender 管理与编程

版本 8

S152-0843-00

注意

在使用本资料及其支持的产品之前，请阅读第 251 页的『声明』中的一般信息。

第一版（2004 年 5 月）

本版本适用于 IBM DB2 Database Extenders iSeries 版 V5R3 的 V5R3（5722-DE1）及所有后续发行版和修订版，直至在新版本中另有声明为止。本版本仅适用于精简指令集计算机（RISC）系统。

目录

表

关于本书

本节包含下列信息:

- 『谁应使用本书』
- 『使用本书的方法』
- 第 x 页的『突出显示约定』

谁应使用本书

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。本文档的读者应对 XML 和 DB2 有一般的了解。要了解更多关于 XML 的信息，请访问以下 Web 站点:

  http://www.w3.org/XML

  要了解更多关于 DB2 的信息，请访问以下 Web 站点:

  http://www.ibm.com/software/data/db2/library
- 熟悉 DB2 UDB 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 UDB 应用程序的一种或多种编程语言的 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

使用本书的方法

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。

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 UDB 数据库。如果您需要管理包含 XML 数据的 DB2 UDB 数据库，则阅读本部分。

第 3 部分. 编程

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

第 4 部分. 参考

此部分描述如何使用 XML Extender 管理命令、用户定义的类型、用户定义的函数和存储过程。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的类型（UDT）、用户定义的函数（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以及其他 IBM® XML 产品。

突出显示约定

本书使用下列约定:

粗体文本指示:

- 命令
- 字段名
- 菜单名
- 按钮

斜体文本指示

- 要用值来替换的变量参数
- 强调词
- 词汇表术语的首次使用

大写字母指示:

- 数据类型
- 列名
- 表名

示例文本指示:

- 系统消息
- 您输入的值
- 编码示例
- 目录名
- 文件名

如何阅读语法图

在整本书中，命令和 SQL 语句的语法都是使用语法图来描述的。

阅读语法图的方法如下所述:

- 从左到右、从上到下且沿着线路所指的路径阅读语法图。

  ▸▸── 符号指示语句开始。

  ──▸ 符号指示语句语法在下一行上继续。

  ▸── 符号指示语句继续前一行。

  ──▸◂ 符号指示语句结束。

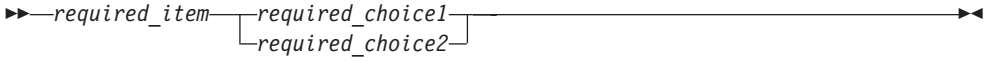
  语法单位的图例不同于其他完整的语句，它以 ▸── 符号开头，以 ──▸ 符号结束。
- 必需的项出现在水平线（主路径）上。

- 可选的项出现在主路径下面。

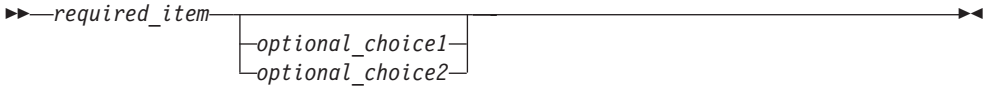
  如果某可选项出现在主路径的上面，说明那个项对本语句的执行没有影响，只用于增加可读性。

- 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。

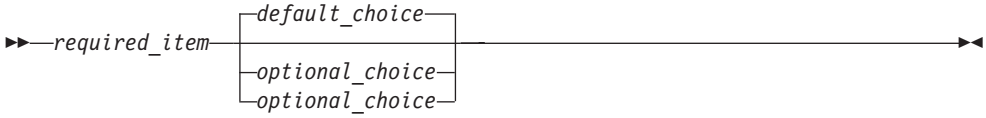
  若*必须*选择其中一个项，则堆栈的其中一个项出现在主路径上。



  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，表示这个项可以重复。

- 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，关键字可以大写也可以小写。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。
  - 若显示标点符号、圆括号、算术运算符或其他这样的符号，则必须作为语法的一部分输入它们。

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。

第 1 章 介绍

XML Extender 简介

DB2 的 XML Extender 提供了存储和访问 XML 文档、根据现有关系数据生成 XML 文档以及将来自 XML 文档的行插入关系表中的能力。XML Extender 提供了新的数据类型、函数和存储过程，来管理 DB2 UDB 关系数据库（在本书中称为 “RDB 数据库” 或简称 “数据库” ）中的 XML 数据。

XML Extender 可用于下列操作系统:

- Windows® NT
- Windows 2000
- AIX®
- Solaris Operating Environment
- Linux
- OS/390 和 z/OS
- iSeries

相关概念:

- 第 3 页的『XML 文档』
- 第 4 页的『XML Extender 的功能』
- 第 7 页的『课程: 将 XML 文档存储在 XML 列中』
- 第 16 页的『课程: 组成 XML 文档』
- 第 6 页的『XML Extender 教程』

XML 文档

由于公司趋向于在不同的应用程序间共享数据，所以他们经常会碰到这样的问题: 采用可导入其他应用程序的格式来复制、转换、导出或保存数据。其中的许多转换过程都有可能会删除一些数据，或至少要求用户完成一个冗长乏味的过程来确保数据保持一致。这种手工检查既浪费时间又浪费金钱。

解决此问题的一种方法就是由应用程序开发者编写*开放式数据库连接（ODBC）*应用程序，这是一种标准的应用程序编程接口（API），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。这些应用程序将数据保存在数据库管理系统中。从该系统可对数据进行处理，且可将数据表示为另一应用程序所需的格式。必须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。应用程序变化迅速，很快就会过时。将数据转换为 HTML 的应用程序提供了展示解决方案，但展示的数据不能实际地用于其他用途。需要一种方法将数据与其表达式分开，以提供应用程序间的实用的交换格式。

XML（即*可扩展标记语言*）解决了此问题。XML 是可扩展的，因为此语言是一种元语言，它允许您根据企业的需要创建自己的语言。使用 XML 不仅能从特定的应用程序中

捕获数据，还能捕获数据结构。虽然 XML 并不是唯一的数据交换格式，但 XML 已渐渐成为可接受标准。通过遵循此标准，应用程序可共享数据而无需事先使用专用格式来转换数据。

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利用它。

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。您的项目管理应用程序可以采用 XML 格式导出任务，然后可将这些任务“按现状”导入到日历应用程序中。在今天的互连世界中，应用程序供应商急切盼望 XML 交换格式成为应用程序的基本功能。

如何在 DB2 中处理 XML 数据

虽然 XML 通过为数据交换提供标准格式解决了许多问题，但也面临挑战。当构建企业数据应用程序时，您必须回答如下问题:

- 想多久复制一次数据？
- 必须在应用程序间共享何种类型的信息？
- 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 信息和元信息合并到数据库中，可更有效地获得其他应用程序需要的 XML 结果。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 UDB 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，可以选择将整个 XML 文档以为 XML 数据提供的用户定义的类型（XML 数据类型）存储在 DB2 中，也可以将 XML 内容映射为关系表中的基本数据类型。对于 XML 数据类型，除 Text Extender 提供的结构化文本搜索之外，XML Extender 还添加了搜索大量 XML 元素或属性值数据类型的能力。

XML Extender 提供了两个在 DB2 中存储和访问 XML 数据的方法:

XML 列方法

XML 集合方法
组成和分解带有一个或多个关系表的 XML 文档的内容。

XML Extender 的功能

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

- 帮助您管理关系表中的 XML 数据的集成的管理工具
- 用于数据库中的 XML 数据的存储和访问方法
- 数据类型定义（DTD）资源库，供您存储用于验证 XML 数据的 DTD
- 一个称为“文档访问定义”（DAD）的映射文件，用于将 XML 文档映射至关系数据
- 指定 XML 文档中元素或属性位置的位置路径。

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。

可以使用下列工具来完成 XML Extender 的管理任务:

- 可以从 OS 命令行运行 **dxxadm** 命令。
- 可以从 “iSeries™ 导航器” 运行存储过程。
- XML Extender 管理存储过程允许您从程序中调用管理命令。

存储和访问方法: 为了将 XML 文档与 DB2 数据结构集成, XML Extender 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 列方法

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 列方法非常有用。将把这些文档插入到对 XML 启用的列中, 并可对这些文档进行更新、检索和搜索。可将元素和属性数据映射至 DB2 UDB 表（副表）, 可以为该表建立索引以快速搜索。

XML 集合方法

此方法帮助您将 XML 文档结构映射至 DB2 UDB 表, 以便可从现有 DB2 UDB 数据组成 XML 文档或分解 XML 文档, 将无标记的数据存储在 DB2 UDB 表中。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更是如此。

DTD: XML Extender 还允许您存储 DTD, 即 XML 元素和属性声明的集合。当对 XML *启用*数据库时, 将创建 DTD 资源库表（DTD_REF）。此表的每一行都表示具有附加元数据信息的 DTD。用户可访问此表以插入他们自己的 DTD。DTD 用于验证 XML 文档的结构。

DAD 文件: 通过使用*文档访问定义（DAD）*文件, 可以指定 XML Extender 如何处理结构化 XML 文档。DAD 文件是一个 XML 文档, 它将 XML 文档结构映射到 DB2 UDB 表。当在列中存储 XML 文档, 或者组成或分解 XML 数据时, 都要使用 DAD 文件。DAD 文件指定您是使用 XML 列方法, 还是定义用于组成或分解的 XML 集合来存储文档。

位置路径: *位置路径*指定 XML 文档中元素或属性的位置。XML Extender 使用位置路径来导航 XML 文档的结构并定位元素和属性。

例如, 位置路径 `/Order/Part/Shipment/ShipDate` 指向 shipDate 元素, 该元素是 Shipment、Part 和 Order 元素的子元素, 如以下示例所示:

```
<Order>
  <Part>
    <Shipment>
      <ShipDate>
+...
```

第 6 页的图 1 显示了位置路径的示例以及它与 XML 文档结构之间的关系。

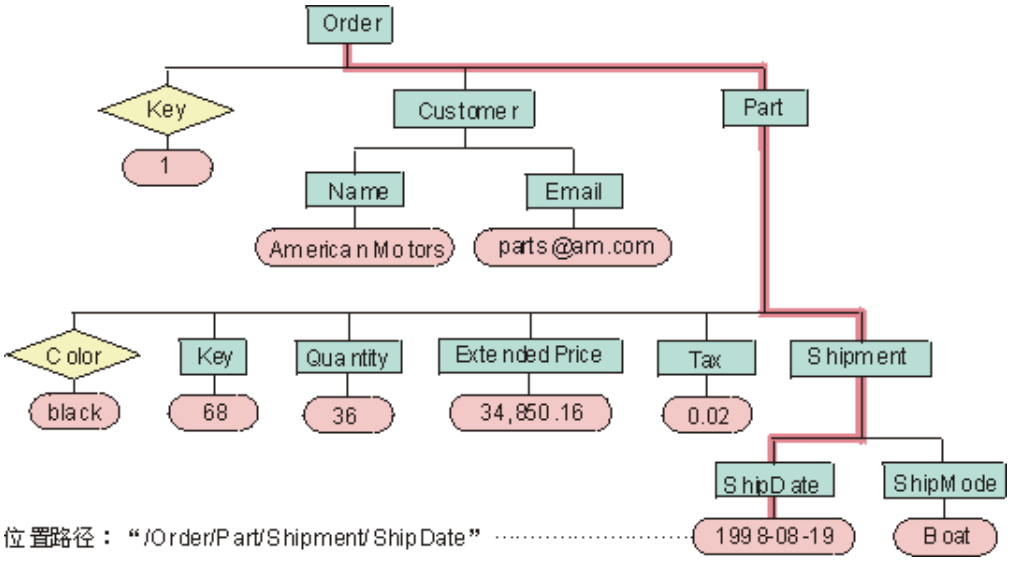


图 1. 将文档作为结构化 XML 文档存储在 DB2 UDB 表列中

位置路径在下列情况中使用:

XML 列

- 当使用 XML Extender 用户定义的函数时，用于标识要抽取或更新的元素和属性。
- 还用于将 XML 元素或属性的内容映射至副表。

XML 集合

用于从存储过程覆盖 DAD 文件中的值。

要指定位置路径，XML Extender 使用 *XML 路径语言（XPath）*（用于对 XML 文档的各部分进行寻址的语言）的一个子集。

有关 Xpath 的更多信息，请参阅以下 Web 页面:

http://www.w3.org/TR/xpath

相关概念:

- 第 4 页的『如何在 DB2 中处理 XML 数据』
- 第 7 页的『课程: 将 XML 文档存储在 XML 列中』
- 第 16 页的『课程: 组成 XML 文档』
- 第 6 页的『XML Extender 教程』

XML Extender 教程

本教程向您传授有关使用 XML Extender 来访问和修改应用程序的 XML 数据的入门知识。提供三门课程:

- 将 XML 文档存储在 XML 列中
- 组成 XML 文档
- 清除数据库

通过遵循教程的课程，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 文档并接着搜索 XML 文档以及从 XML 文档中抽取数据。

在管理课程中，您将 与 XML Extender 管理命令配合使用。在 XML 数据管理课程中，您将使用 XML Extender UDF 和存储过程。本书其余部分中的大多数示例都采用本章使用的样本数据。

先决条件

要完成本教程中的课程，必须已安装了以下必备软件:

- DB2 Universal Database™ V5R3
- 可选: “Series™ 导航器”以运行课程样本

另外，必须设置管理环境。

请参阅 “设置 iSeries 的教程环境”。

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。

运行教程的方法:

提供了几个用于运行脚本和命令的方法。可以使用 “iSeries 导航器” 或 OS 命令行。

- 使用 “导航器” 在 Windows® 环境中将本教程作为存储过程运行。
- 使用 OS 命令行来运行脚本和 SQL 语句。

相关概念:

- 第 29 页的『XML Extender 的管理工具』
- 第 36 页的『XML Extender 管理计划』
- 第 7 页的『课程: 将 XML 文档存储在 XML 列中』
- 第 16 页的『课程: 组成 XML 文档』

课程: 将 XML 文档存储在 XML 列中

XML Extender 提供了在数据库中存储和访问整个 XML 文档的方法。XML 列方法允许您使用 XML 文件类型来存储文档、对副表中的列建立索引以及查询或搜索 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。

本课程讲述如何使用 XML 列存储器和存取方法。

方案:

您的任务是将销售数据归档以供服务部门使用。您需要使用的销售数据存储在使用相同 DTD 的 XML 文档中。

服务部门提供了 XML 文档的建议结构，并指定哪个元素数据的查询最频繁。服务部门希望 XML 文档存储在 SALES_TAB 数据库的 SALES_DB 表中，且能够迅速地搜索它们。SALES_TAB 表有两列包含有关每项销售的数据，第三列将包含 XML 文档。此列名为 ORDER。

要将这个 XML 文档存储在 SALES_TAB 表中，您将:

1. 确定存储 XML 文档所使用的 XML Extender 用户定义的类型（UDT）以及将要频繁地查询哪些 XML 元素和属性。
2. 为 XML 设置 SALES_DB 数据库。
3. 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。
4. 插入 XML 文档的 DTD 以进行验证。
5. 将该文档作为 XMLVARCHAR 数据类型存储。

启用列时，您将定义为了对文档访问定义（DAD）文件中的文档进行结构化搜索而需要建立索引的副表，文档访问定义（DAD）文件是用于指定副表结构的 XML 文档。

SALES_TAB 表在表 1 中作了描述。要对 XML 启用的 XML 列 ORDER 以斜体显示。

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

脚本和样本:

在本教程中，您将使用一组脚本来设置您的环境并执行课程中的步骤。 操作系统命令行脚本位于 dxxsamples 库中。导航器 SQL 脚本文件位于 /dxxsamples 目录中。

表 2 列示了为完成入门任务而提供的样本。

表 2. XML 集合课程样本的列表

课程描述	OS 命令行脚本	导航器 SQL 脚本文件
创建并填充 SALES_DB 表	C_SALESDB	C_SalesDb.sql
将 DTD getstart.dtd 插入到 DTD_REF 表中	INSERTDTD	InsertDTD.sql
为 XML 列创建 SALES_TAB	C_SALESTAB	C_SalesTab.sql
向 SALES_TAB 添加 ORDER 列	ADDORDER	AddOrder.sql
启用 ORDER 列来作为 XML 列	在文本中描述的手工命令	EnableCol.sql
对副表创建索引	C_INDEX	C_Index.sql
将 XML 文档插入 SALES_TAB XML 列中	INSERTXML	InsertXML.sql
通过副表查询 sales_tab XML 列中存放的 XML 文档	手工命令	QueryCol.sql
除去样本表并禁用列	D_SALESDB 和 CLEANUPCLL	CleanupCol.sql

计划如何存储文档:

在使用 XML Extender 存储文档之前，您需要:

- 了解 XML 文档结构。
- 确定将用来存储 XML 文档的 XML 用户定义的类型。
- 确定服务部门将频繁搜索的 XML 元素和属性，以便可以将其内容存储在副表中，并建立索引以改进性能。

下列各节将说明如何作出这些决策。

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，该订单以订单键作为顶层，客户、部件和交付信息位于下一层。

本课程提供样本 DTD，以了解和验证 XML 文档结构。

确定 XML 列的 XML 数据类型:

XML Extender 提供 XML 用户定义的类型，您可以用这些类型来定义保存 XML 文档的列。这些数据类型为:

XMLVARCHAR
用于 DB2 中存储的小型文档

XMLCLOB
用于 DB2 中存储的大型文档

XMLFILE
用于存储在 DB2 之外的文档

在本课程中，将把小型文档存储在 DB2 中，因此将使用 XMLVARCHAR 数据类型。

确定要搜索的元素和属性:

在理解 XML 文档结构和应用程序的需要之后，您就可以确定将要最频繁地搜索或抽取哪些元素和属性，或者哪些元素或属性的查询成本最高。服务部门将频繁地查询订单的订单键、客户姓名、价格和交付日期，并且他们需要快速执行这些搜索。此信息包含在 XML 文档结构的元素和属性中。表 3 描述了每个元素和属性的位置路径。

表 3. 要搜索的元素和属性

数据	位置路径
订单键	/Order/@key
客户姓名	/Order/Customer/Name
价格	/Order/Part/ExtendedPrice
交付日期	/Order/Part/Shipment/ShipDate

将 XML 文档映射至副表:

要将 XML 文档映射至副表，必须为 XML 列创建 DAD 文件。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建立索引的 DB2 UDB 副表，从而改进搜索性能。

在标识要搜索的元素和属性之后，确定应在副表中以何方式组织它们，要使用多少个表以及哪个表中有哪些列。通过将类似的信息放入同一个表来组织副表。文档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，部门元素可重复多次，从而价格和日期元素也可出现多次。每个多次出现的元素必须在它们自己的副表中。

您还必须确定元素或属性值应使用哪些 DB2 UDB 基本类型，这由数据的格式确定。

- 如果数据是文本，则使用 VARCHAR。
- 如果数据是整数，则使用 INTEGER。
- 如果数据是日期，并且您想要进行范围搜索，则使用 DATE。

在本教程中，元素和属性映射至 ORDER_SIDE_TAB、PART_SIDE_TAB 或 SHIP_SIDE_TAB。下表显示了每个元素或属性映射至哪一个表。

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

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 将:

- 创建用户定义的类型（UDT）、用户定义的函数（UDF）和存储过程。
- 创建控制表并用 XML Extender 所需的必需元数据填充控制表
- 创建 DB2XML 模式并指定必需的特权

为 XML 启用数据库:

使用下列方法之一来启用数据库。

- **导航器**: 输入命令:

  ```
  Run EnableDB.sql
  ```

- **OS 命令行**: 输入:

  ```
  CALL PGM(QDBXM/QZXMADM) PARM(enable_db &RDBDatabase)
  ```

创建 SALES_DB 表并进行填充:

要设置课程环境，请创建并填充 SALES_DB 表。这些表包含在计划章节中所描述的表。

要创建这些表，使用下列方法之一:

- **导航器**: 运行:

  C_SalesDb.sql

- **OS 命令行**: 输入以下命令:

```
RUNSQLSTM SRCFILE(DXXSAMPLES/SQLSTMT)
          SRCMBR(C_SALESDB) NAMING(*SQL)
```

启用 XML 列并存储文档:

在本课程中，您将对 XML Extender 启用一列并在该列中存储 XML 文档。要完成这些任务，您将:

1. 将 DTD 存储在 DTD 资源库中
2. 为 XML 列创建 DAD 文件
3. 创建 SALES_TAB 表
4. 添加 XML 类型的列
5. 启用 XML 列
6. 查看列和副表
7. 对副表建立索引以进行结构化搜索。
8. 存储 XML 文档

将 DTD 存储在 DTD 资源库中:

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，该表名称为 DTD_REF。该表即为 DTD 资源库，并可用于存储 DTD。在验证 XML 文档时，必须将 DTD 存储在此资源库中。本课程的 DTD 位于 dxxsamples/dtd/getstart.dtd 中。

- **导航器**: 运行

  InsertDTD.sql

- **OS 命令行**: 输入:

```
RUNSQLSTM SRCFILE(DXXSAMPLES/SQLSTMT)
          SRCMBR(INSERTDTD) NAMING(*SQL)
```

为 XML 列创建 DAD 文件:

本节说明如何为 XML 列创建 DAD 文件。在 DAD 文件中，您指定正在使用的访问和存储方法是 XML 列。在 DAD 文件中定义用于建立索引的表和列。

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*。dxxsamples/dad/getstart_xcolumn.dad 中存在 DAD 文件的样本，它类似于您将创建的 DAD 文件样本。它与在下列步骤中生成的文件稍有差别。如果将它用于本课程，则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 NO，而不是 YES。

要创建 DAD 文件，以便与 XML 列配合使用:

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

   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 声明的 DAD 头。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM " /dxxsamples/dtd/dad.dtd">
```

DAD 文件为 XML 文档，并且需要 XML 声明。

3. 插入文档的开始和结束（<DAD> 和 </DAD>）标记。其他所有标记都位于这些标记内。
4. 如果要验证文档，则插入带 DTD 标识的开始和结束（<DTDID> 和 </DTDID>）标记以指定 DTD。

   ```
   <dtdid> dxxsamples/dtd/getstart.dtd</dtdid>
   ```

   验证此字符串是否与在将 DTD 插入 DTD 资源库表时用作第一个参数值的值相匹配。例如，如果在另一机器驱动器上工作，则用于 DTD 标识的路径可能与上面提到插入 DTD 引用表的字符串不同。
5. 插入开始和结束（<validation> 和 </validation>）标记以及关键字 YES 或 NO，以指示 XML Extender 是否将使用您插入到 DTD 引用表中的 DTD 来验证 XML 文档结构。例如:

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

   <validation> 的值必须采用大写字母。
6. 插入开始和结束（<Xcolumn> 和 </Xcolumn>）标记来指定存储方法是 XML 列。
7. 创建副表。对于要创建的每个副表:
   a. 对要生成的每个副表插入开始和结束（<table> 和 </table>）标记，并使用“name=”属性指定双引号中的副表名称，如下所示:

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

   b. 在表标记中，对于要让副表包含的每个列插入 <column> 标记。每个列有四个属性: name、type、path 和 multi_occurrence。

      示例:

      ```
      <table name="person_names">>
      <column name ="fname"
           type="varchar(50)"
           path="/person/firstName"
           multi_occurrence="NO"/>
      <column name ="lname"
           type="varchar(50)"
           path="/person/lastName"
           multi_occurrence="NO"/>
      </table>
      ```

      其中:

      name 指定在副表中创建的列的名称。

      type 指示副表中每个已建立索引的元素或属性的数据类型。

      path 指定 XML 文档中每个将要为其建立索引的元素或属性的位置路径。

      multi_occurrence
      指示 path 属性引用的元素或属性能否在 XML 文档中出现多次。可能的 *multi_occurrence* 值是 *YES* 或 *NO*。如果值为 *NO*，则可以在副表中提及多个 column 标记。如果值为 *YES*，则只能提及副表中的一个列。

```
 <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>
```

8. 确保您具有必需的结束标记:
   - 在最后一个 </table> 标记之后存在结束 </Xcolumn> 标记
   - 在 </Xcolumn> 标记之后存在结束 </DAD> 标记
9. 使用以下名称来保存文件:

   ```
   getstart_xcolumn.dad
   ```

您可以将刚刚创建的文件与样本文件 dxxsamples/dad/getstart_xcolumn.dad 作比较。此文件是启用 XML 列和创建副表所必需的 DAD 文件的工作副本。这些样本文件包含对使用绝对路径名的文件的引用。检查样本文件并将这些值更改为您的目录路径。

创建 SALES_TAB 表:

在本节中，您将要创建 SALES_TAB 表。最初，该表中有两个列包含订单的销售信息。

要创建表:

使用下列方法之一输入以下 CREATE TABLE 语句:

- **导航器**: 运行 **C_SalesTab.sql**
- **OS 命令行**: 输入:

```
RUNSQLSTM SRCFILE(DXXSAMPLES/SQLSTMT)
          SRCMBR(C_SALESTAB) NAMING(*SQL)
```

添加 XML 类型的列:

向 SALES_TAB 表添加新的列。此列将包含先前生成的完整 XML 文档，并且必须是 XML UDT。XML Extender 提供了多种数据类型。在本课程中，将此文档存储为 XMLVARCHAR。

要添加 XML 类型的列:

使用下列方法之一运行 SQL ALTER TABLE 语句:

- **导航器**: 运行 **AddOrder.sql**

- **OS 命令行**: 输入:

```
RUNSQLSTM SRCFILE(DXXSAMPLES/SQLSTMT)
          SRCMBR(ADDORDER) NAMING(*SQL)
```

启用 XML 列:

在创建 XML 类型的列之后，为 XML Extender 启用该列。在启用列时，XML Extender 将读取 DAD 文件并创建副表。在启用列之前，必须:

- 确定您是否想要创建 XML 列的缺省视图，该视图包含与副表列相连接的 XML 文档。可在查询 XML 文档时指定缺省视图。在本课程中，您将使用 `-v` 参数来指定视图。
- 确定是否要将主键指定为*根标识*，即应用程序表中主键的列名，并且它是将所有副表与应用程序表相关联的唯一标识。如果不指定主键，则 XML Extender 将 DXXROOT_ID 列添加至应用程序表和副表。

  ROOT_ID 列用作键，它将应用程序表和副表紧密联系在一起，并且允许 XML Extender 在 XML 文档更新时自动更新副表。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 并将该列添加至副表。
- 确定您是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:

使用下列方法之一运行 **dxxadm enable_column** 命令:

- **导航器**:

  运行 **EnableCol.sql**
- **OS 命令行**: 输入:

```
CALL QDBXM/QZXMADM PARM(enable_column dbname
 Sales_Tab Order
'/dxxsamples/dad/getstart_xcolumn.dad' '-v'
sales_order_view   '-r'  invoice_num)
```

  其中，*dbname* 是 RDB 数据库的名称。

XML Extender 将创建带有 INVOICE_NUM 列的副表，并创建缺省视图。

重要事项: 不要以任何方式修改副表。对副表的更新仅应通过对 XML 文档自身的更新来完成。XML Extender 将在您更新 XML 列中的 XML 文档时自动更新副表。

查看列和副表:

启用 XML 列后，就为该 XML 列和副表创建视图。在处理 XML 列时，可以使用此视图。

要查看 XML 列和副表列:

从 DB2 UDB 命令行提交以下 SQL SELECT 语句:

```
SELECT * FROM SALES_ORDER_VIEW
```

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

为了进行结构化搜索而对副表建立索引:

对副表创建索引允许您对 XML 文档执行快速结构搜索。在本节中，您将会对在启用 XML 列 ORDER 时创建的副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频繁查询的列。表 4 描述了将对其建立索引的列:

表 4. 要进行索引的副表列

列	副表
ORDER_KEY	ORDER_SIDE_TAB
CUSTOMER	ORDER_SIDE_TAB
PRICE	PART_SIDE_TAB
DATE	SHIP_SIDE_TAB

要对副表建立索引:

使用下列方法之一运行下列 CREATE INDEX SQL 命令:

- **导航器**: 运行 **C_Index.sql**
- **OS 命令行**: 输入:

```
RUNSQLSTM SRCFILE(DXXSAMPLES/SQLSTMT)
          SRCMBR(C_INDEX) NAMING(*SQL)
```

命令行: 也可以是:

存储 XML 文档:

既然启用了可包含 XML 文档的列并对副表建立了索引，您就可以使用 XML Extender 提供的函数来存储文档。在将数据存储到 XML 列中时，可使用缺省数据类型转换函数或 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 的列中，所以将使用缺省数据类型转换函数。

要存储 XML 文档:

1. 打开 XML 文档 dxxsamples/xml/getstart.xml dxxsample/xml/getstart.xmldxxsamples/xml/getstart.xml 确保 DOCTYPE 中的文件路径与 DAD 中指定的 DTD 标识以及将 DTD 插入 DTD 资源库时指定的 DTD 标识相匹配。通过查询 DB2XML.DTD_REF 表或检查 DAD 文件中的 DTDID 元素，可验证它们是否匹配。如果使用的不是缺省驱动器和目录，则可能需要更改 DOCTYPE 声明中的路径。
2. 使用下列方法之一运行 SQL INSERT 命令:
   - **导航器**: 运行 **InsertXML.sql**
   - **OS 命令行**: 输入:

     ```
     RUNSQLSTM SRCFILE(DXXSAMPLES/SQLSTMT)
               SRCMBR(INSERTXML)NAMING(*SQL)
     ```

验证已更新的表。从命令行对这些表运行下列 SELECT 语句。

```
SELECT * FROM SALES_TAB
SELECT * FROM PART_SIDE_TAB
SELECT * FROM ORDER_SIDE_TAB
SELECT * FROM SHIP_SIDE_TAB
```

查询 XML 文档:

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

要查询副表:

使用下列方法之一运行 SQL SELECT 语句:

- **导航器**: 运行 **QueryCol.sql**
- **DB2 命令行**:

  输入:

```
select distinct sales_person from schema.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。

您已经完成了将 XML 文档存储到 DB2 UDB 表中的入门教程。例如:

```
SALES_PERSON
--------------------

Sriram Srinivasan
```

相关概念:

- 第 3 页的『XML Extender 简介』
- 第 16 页的『课程: 组成 XML 文档』
- 第 6 页的『XML Extender 教程』

课程: 组成 XML 文档

本课程教您如何从现有的 DB2® 数据组成 XML 文档。

方案:

您的任务是采集现有采购单数据库 SALES_DB 中的信息，并从该数据库中抽取要存储在 XML 文档中的请求信息。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。

通过使用现有数据，您将根据这些表中的数据组成 XML 文档 `getstart.xml`。

要组成 XML 文档，您将计划和创建 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成的，所以您将创建 XML 集合，并将这些表与 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已组成的 XML 文档。

下表中描述了 XML 文档中的现有数据库数据。带星号的列名都是服务部门在 XML 文档结构中请求的列。

ORDER_TAB

列名	数据类型
ORDER_KEY *	INTEGER
CUSTOMER	VARCHAR(16)

列名	数据类型
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:

在使用 XML Extender 来组成文档之前，需要确定 XML 文档的结构，以及它与数据库数据结构的对应方式。本节概述服务部门所请求的 XML 文档结构，以及您将用来定义 XML 文档结构的 DTD。本节还将说明此文档如何映射至包含用于填充文档的数据的列。

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。服务部门希望文档结构是直观和灵活的，使用元素来描述数据，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。）

设计了文档结构之后，您会创建一个 DTD 来描述 XML 文档的结构。此课程为您提供了一个 XML 文档和一个 DTD。使用 DTD 的规则和 XML 文档的分层结构，您可以创建数据的分层映射，如第 18 页的图 2 中所示。

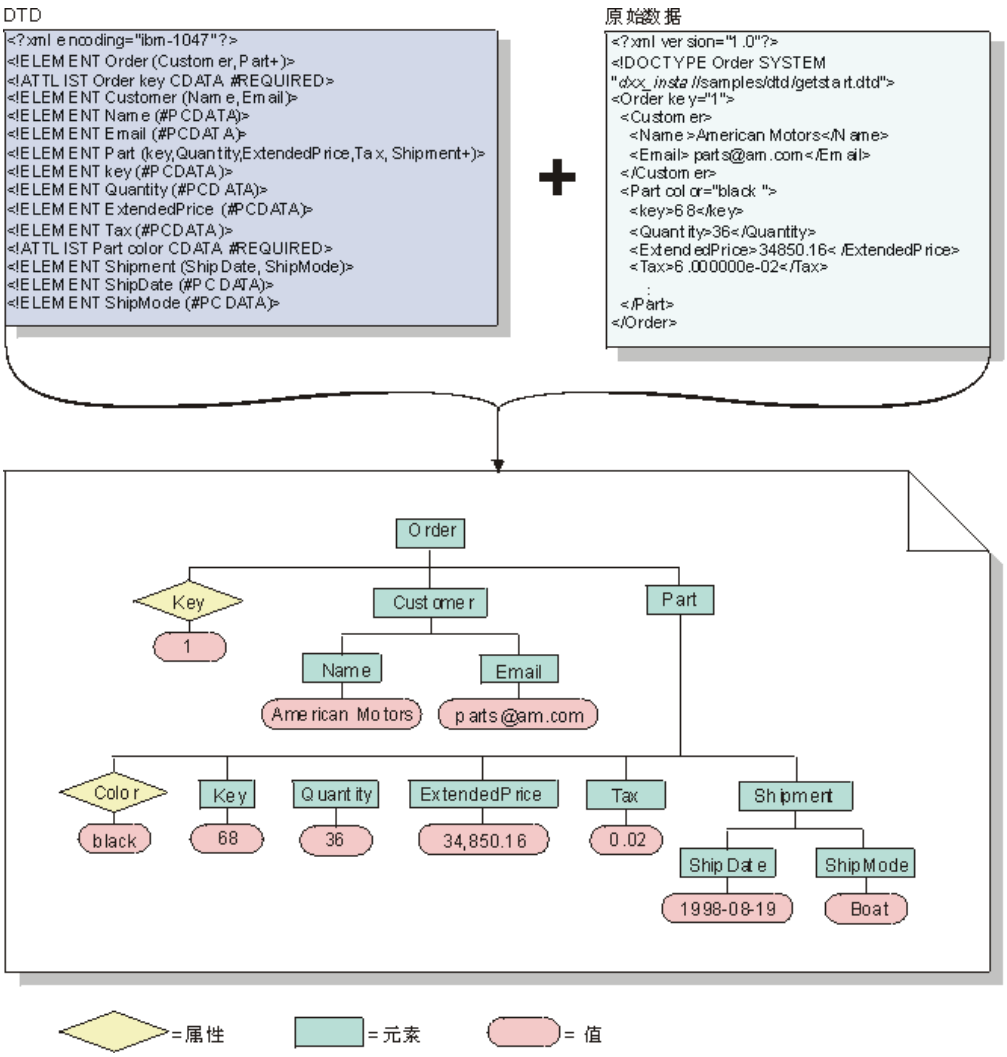


图 2. DTD 和 XML 文档的层次结构

映射 XML 文档和数据库关系:

在设计结构并创建 DTD 之后，您需要说明文档的结构与将用来填充元素和属性的 DB2 UDB 表是如何相关的。可以将分层结构映射至关系表中的特定列，如第 19 页的图 3 中所示。

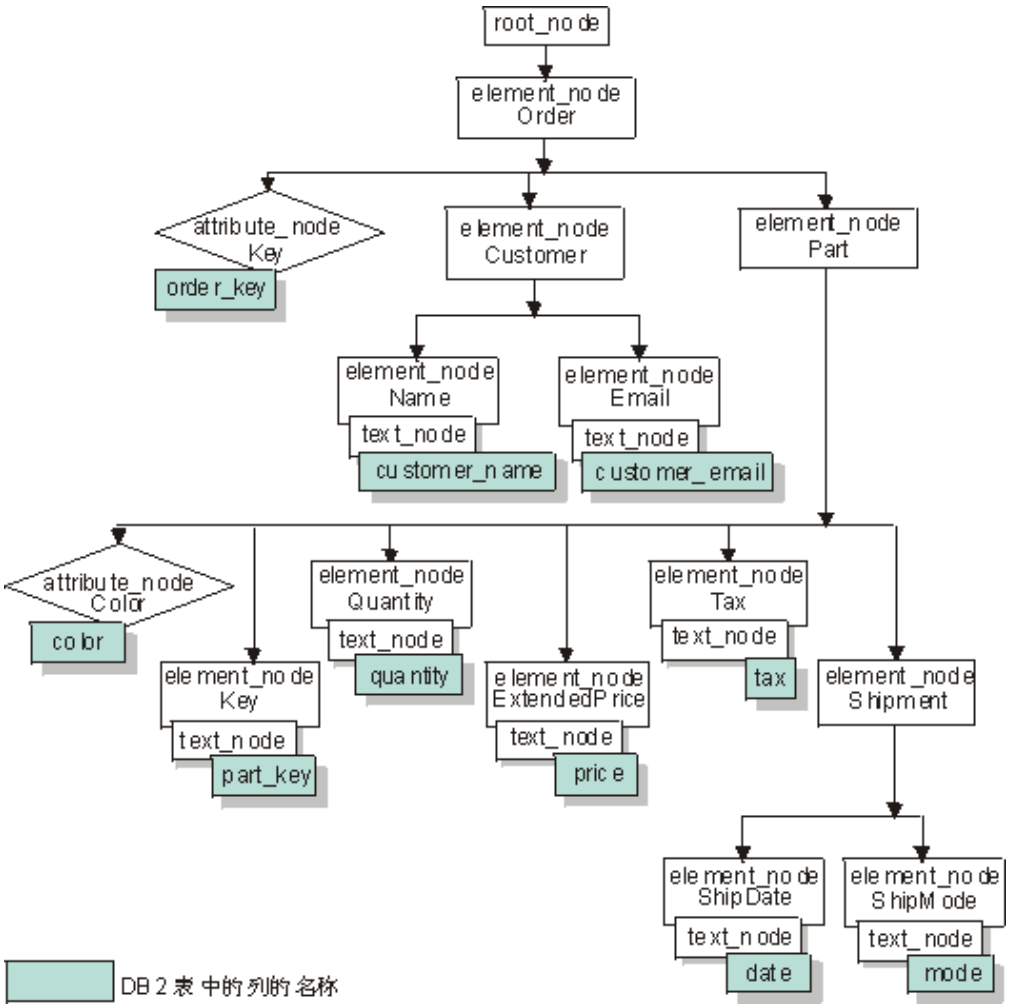


图 3. 映射至关系表列的 XML 文档

此图使用节点来显示 XML 文档结构中的元素、属性和文本。这些节点用在 DAD 文件中，将在稍后的步骤中更详细地解释这些节点。

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

在此教程中，您将为 XML 集合创建一个 DAD 文件，以用于组成文档。XML 集合 DAD 文件将具有现有数据的表映射至 XML 文档结构。

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 3 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。

脚本和样本:

此课程为您提供一组脚本，用于设置您的环境。OS 命令行脚本位于 dxxsamples 库中。导航器 SQL 脚本文件位于 /dxxsamples 目录中。

表 5 列示了为完成入门任务而提供的样本。

表 5. XML 集合课程样本的列表

课程描述	OS 命令行脚本	导航器 SQL 脚本文件
创建并填充 SALES_DB 表	C_SALESDB	C_SalesDb.sql
组成 XML 文档并将它返回至结果表	手工命令	Genxml_sql.sql
除去样本表并禁用列	D_SALESDB 和 CLEANUPCOL	CleanupCllec.sql

设置课程环境:

如果已完成第一课 “将 XML 文档存储在 XML 列中”，则跳过本节。在本节中，您将:

- 启用数据库。
- 创建并填充用于课程的表。

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:

- 创建用户定义的类型（UDT）、用户定义的函数（UDF）和存储过程。
- 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 模式并指定必要的特权。

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

要为 XML 启用数据库，使用下列方法之一:

- **导航器**: 输入命令:

```
CALL &Schema.QZXMADM('enable_db','
  &DBNAME');
```

- **OS 命令行**: 输入:

```
CALL PGM(QDBXM.QZXMADM) PARM(enable_db
  &RDBDatabase)
```

创建并填充 SALES_DB 表:

要设置课程环境，创建并填充 SALES_DB 表。这些表包含在计划章节中所描述的表。

要创建表:

- **导航器**: 运行 **C_SalesDb.sql**
- **（iSeries）OS 命令行**: 输入以下命令:

```
RUNSQLSTM SRCFILE(DXXSAMPLES/SQLSTMT)
          SRCMBR(C_SALESDB) NAMING(*SQL)
```

为 XML 集合创建 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。通过创建 DAD 文件来定义集合。

在本节中，在 DAD 文件中创建映射方案，该映射方案指定表与 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*。*dxx_install* dxxsamples/dad/getstart_xcollection.dad 中存在 DAD 文件的样本，它类似于您将创建的 DAD 文件样本。

它与在下列步骤中生成的文件稍有差别。如果将该文件用于本课程，则注意文件路径可能与您的环境的文件路径不同，您可能需要更新样本文件。

要创建 DAD 文件以组成 XML 文档:

1. 从 dxxsamples/xml 目录，打开文本编辑器并创建名为 getstart_xcollection.dad 的文件。
2. 使用以下文本来创建 DAD 头:

   ```
   <?xml version="1.0"?>
   <!DOCTYPE DAD SYSTEM "/dxxsamples/dtd/dad.dtd">
   ```

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 是否在您将 DTD 插入到 DTD 资源库表中时，验证 XML 文档结构。本课程并不需要 DTD，该值为 **NO**。

   ```
   <validation>NO</validation>
   ```

   <validation> 标记的值必须是大写的。

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 UDB 表的集合中。

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

6. 在 <Xcollection> 标记之后，提供一个 SQL 语句以指定用于 XML 集合的表和列。此方法称为 SQL 映射，它是将关系数据映射至 XML 文档结构的两种方法中的一种。输入下列语句:

   ```
   <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,
       (select db2xml.generate_unique()
     as ship_id, date, mode, part_key from ship_tab) as 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> </Xcollection>
   ```

   此 SQL 语句在使用 SQL 映射时使用下列准则。请参阅第 19 页的图 3 以了解文档结构。

   - 列是以自顶向下的顺序（即按 XML 文档结构的层次结构）来指定的。例如，order 和 customer 元素的列最先，part 元素的列其次，而 shipment 元素的列最后。
   - 需要数据库中数据的模板的重复段或非重复段的列组合在一起。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 Part 元素的各列之前。

- SHIP_TAB 表没有一个单键条件列，因此，使用 generate_unique 用户定义的函数来生成 SHIP_ID 列。
- 然后，将对象标识列按自顶向下的顺序列示在 ORDER BY 语句中。ORDER BY 中的各列都不受任何模式和表名限定，而且与 SELECT 子句中的列名相匹配。

7. 添加以下要在已组成的 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> 标记以在您正在组成的 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
"/dxxsamples/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，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 UDB 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。

第 19 页的图 3 显示了 XML 文档的分层结构以及 DB2 UDB 表列，并指示使用了哪种节点。阴影框指示将从其抽取数据以便组成 XML 文档的 DB2 UDB 表列名。

要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>
```

<Shipment> 子元素具有 multi_occurrence=YES 的属性。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。这些属性嵌套在相应的 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，以指示 XML 元素中包含组成文档时要从 DB2 UDB 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 定义一个 <column/> 标记。这些标记指定组成 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。在 <column/> 标记中定义的列必须位于 <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. 确保您具有必需的结束标记:
    - 最后一个 </element_node> 标记后有结束 </root_node> 标记
    - </root_node> 标记后有结束 </Xcollection> 标记
    - </Xcollection> 标记后有结束 </DAD> 标记
12. 将文件另存为 getstart_xcollection.dad。

您可以将创建的文件与样本文件 *dxx_install* dxxsamples/dad/getstart_xcollection.dad 作比较。此文件是组成 XML 文档所必需的 DAD 文件的工作副本。样本文件包含位置路径和文件路径名，可能需要将这些名称更改为与您的环境相匹配才能成功运行。

在应用程序中，如果您经常会使用 XML 集合来组成文档，则可通过启用该集合来定义一个集合名。当运行存储过程时，如果指定了集合名（而不是 DAD 文件名），则启用该集合即在 XML_USAGE 表中注册该集合，且有助于改进性能。在这些课程中，将不启用该集合。

组成 XML 文档:

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

要组成 XML 文档:

使用下列方法之一:

- **导航器**: 运行 **Genxml_sql.sql**
- **OS 命令行**: 输入:

```
CALL DXXSAMPLES/GENX PARM(dbName'/dxxsamples
/dad/getstart_xcollection.dad' result_tab doc ' ')
```

技巧: 本步骤教您如何使用 DB2 UDB 存储过程的结果集功能部件来生成一个或多个已组成的 XML 文档。使用结果集允许您访存多行以生成多个文档。在您生成每个文档时，您可以将它导出到文件中。此方法是演示使用结果集的最简单方法。有关访存数据的更有效的方法，请参阅 DXXSAMPLES/QCSRC 源文件 *dxx_install* 中的 CLI 示例。

清除教程环境:

如果要清除课程环境，则可以运行所提供的脚本之一或从命令行输入命令来:

- 禁用 XML 列 ORDER。
- 删除在课程中创建的表。
- 从 DTD 资源库表中删除 DTD。

如果您未完成本章中的两个课程，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:

使用下列方法之一运行清除命令文件:

导航器:

- 要清除 XML 列环境，运行 **CleanupCol.sql**
- 要清除 XML 集合环境，运行 **CleanupCllec.sql**

OS 命令行:

- 要清除 XML 列环境:
  1. 输入:

     ```
     RUNSQLSTM SRCFILE(DXXSAMPLES/SQLSTMT)
               SRCMBR(D_SALESDB) NAMING(*SQL)
     ```

  2. 输入:

     ```
     CALL PGM(QDBXM/QZXMADM)
     PARM(disable_column &DBNAME Sales_Tab Order)
     ```

  3. 输入:

     ```
     RUNSQLSTM SRCFILE(DXXSAMPLES/SQLSTMT)
               SRCMBR(CLEANUPCOL) NAMING(*SQL)
     ```

- 要清除 XML 集合环境，输入:

  ```
  RUNSQLSTM SRCFILE(DXXSAMPLES/SQLSTMT)
            SRCMBR(D_SALESDB) NAMING(*SQL)
  ```

相关概念:

- 第 3 页的『XML Extender 简介』
- 第 7 页的『课程: 将 XML 文档存储在 XML 列中』
- 第 6 页的『XML Extender 教程』

第 2 部分 管理

此部分描述如何执行 XML Extender 的管理任务。

第 2 章 管理

XML Extender 的管理工具

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 为能够使用的管理任务提供以下命令行工具和编程接口。

管理 - 详细信息

iSeries 上的 XML 操作环境

下列各节描述 iSeries 的 XML 操作环境。

应用程序编程

为应用程序提供的所有 XML Extender 设施都在 iSeries 环境中作为存储过程或用户定义的函数（UDF）运行。某些引用 XML 文件数据类型的 UDF 需要访问 IFS 系统。DB2 UDB XML 跟踪文件也被写入 IFS 文件。

为开发 XML Extender 应用程序提供了两个 C 语言头文件。这些文件包含用于调用存储过程和定义错误码的有用常量。在安装了该产品后，可在下列目录中找到头文件:

- `/qibm/proddata/db2extenders/xml/include/dxx.h`
- `/qibm/proddata/db2extenders/xml/include/dxxrc.h`

要使用这些头文件来开发 C++ 应用程序，在 **CRTCPPMOD** iSeries 命令上使用 `INCDIR('/qibm/proddata/db2extenders/xml/include')` 选项。

管理环境

当在 iSeries 环境中执行管理任务时，可使用“XML Extender 管理”向导、Qshell、“导航器”或本机操作系统（OS）命令行。

管理向导

可以从 Windows 或 UNIX 客户机使用“管理”向导或 iSeries 环境来完成管理任务。

Qshell

可以在 Qshell 中运行管理命令 **dxxadm** 和它的选项。该管理命令在第 113 页的第 6 章，『dxxadm 管理命令』中描述，它为 XML Extender 提供了用于管理 XML 列、XML 集合和数据库的选项。

导航器 可以在“导航器”中调用管理存储过程。管理存储过程在第 175 页的『XML Extender 管理存储过程』中描述，它为 XML Extender 提供了用于管理 XML 列、XML 集合和数据库的选项。

OS 命令行

可以从 OS 命令行运行管理程序 QZXMADM。此程序使用第 113 页的第 6 章，『dxxadm 管理命令』中描述的管理命令参数，它为 XML Extender 提供了用于管理 XML 列、XML 集合和数据库的选项。

下表概述了 XML Extender 的管理环境。

表 6. XML Extender 存储过程和命令

环境	Qshell	导航器	OS 命令行
样本文件	DAD、DTD 和 XML 文件存储在 *dxxsamples* 目录下。 • *dxxsamples*/dtd/dad.dtd • *dxxsamples*/dtd/*.* • *dxxsamples*/dad/*.* • *dxxsamples*/xml/*.*	DAD、DTD 和 XML 文件存储在 *dxxsamples* 目录下。 • *dxxsamples*/getstart.exe • *dxxsamples*/dtd/dad.dtd • *dxxsamples*/dtd/*.* • *dxxsamples*/dad/*.* • *dxxsamples*/xml/*.*	DAD、DTD 和 XML 文件存储在 *dxxsamples* 目录下。 • *dxxsamples*/dtd/dad.dtd • *dxxsamples*/dtd/*.* • *dxxsamples*/dad/*.* • *dxxsamples*/xml/*.*
程序可执行文件	DXXSAMPLES 库，*dxxsamples*/* 中的符号链接指向它。	iSeries 上的 DXXSAMPLES 库；在 Windows 或 UNIX 上没有可执行文件。	DXXSAMPLES 库: • SQLSTMT 文件中的样本 SQL 脚本 • QCLSRC 文件中的样本 CL 源 • QCSRC 文件中的样本 C++ 源
命令脚本	无	*path*/DXXSAMPLES 中的 *.SQL 文件	DXXSAMPLES/SQLSTMT

准备管理 DB2 XML Extender

要运行 XML Extender，需要安装下列软件。

iSeries 的必需软件:

- DB2 Universal Database™ V5R3
- iSeries™ International Unicode 组件（5722SS1 的选项 39）QICU
- iSeries System Openness Includes（5722SS1 的选项 13）QSYSINC
- iSeries Portable App Solutions Environment（5722SS1 的选项 33）QPASE（编译所需）
- WebSphere® Development ToolSet（5722WDS 的选项 51）51–54 ILE-C 系列
- XML Toolkit for iSeries（5733XT1）
- 如果计划开发使用 Java™ 的应用程序或基于 Web 的应用程序，则还可能需要 5722JV1。

可选软件:

- 对于结构文本搜索，DB2 通用数据库 XML Extender V7.2，随 DB2 通用数据库 一起提供。
- 对于 XML Extender 管理向导:
  - DB2 通用数据库 Java 数据库连接（JDBC）

将 XML Extender 从版本 7 迁移到版本 8

如果已在使用 XML Extender V7.2，则必须迁移为 XML Extender 启用的数据库，然后才能将启用 XML 的现有数据库与 XML Extender V8 配合使用。另外，因为已改进了 iSeries V5R2 的 XML 列存储方法，所以必须迁移包含对 iSeries V5R1 中 XML 启用的列的每个模式。

注: 在安装 DB2 UDB XML Extender V8 之前，应用所有 V7.2 PTF 并遵循包含在 PTF 封面信函中的迁移指示信息。

过程:

要自动迁移启用了 XML 的数据库和启用了 XML 的列:

1. 安装 DB2 UDB XML Extender V8。
2. 从操作系统命令行输入:

```
CALL QDBXM/QZXMMIGV
```

IASP 注意事项

对于 iSeries™ 系统，可以使用独立辅助存储池（IASP）设备作为外部数据库。可以为 XML Extender 启用此 IASP。当对 XML Extender 启用 IASP 数据库时，考虑以下事项:

- 可以对 XML Extender 启用您的 *SYSBAS 数据库或 IASP 上的数据库。
- 可以对 XML Extender 启用多个 IASP 数据库，但是每次只能有一个 IASP 数据库处于活动状态。
- 如果已对 XML Extender 启用了 *SYSBAS 数据库，则不能启用 IASP 数据库。

将 XML Extender 数据从 SYSBAS 迁移到 IASP

要将 XML 数据从 SYSBAS 移至 IASP，则需要详细计划。

需要不同的保存／恢复步骤:

- XML 集合数据库文件
- 带有 XML 用户定义的类型（UDT）的数据库文件
- XML 列数据库文件

要迁移至 IASP:

1. 使用『XML 列和 XML 集合的保存和恢复』中描述的步骤来保存 XML 数据。
2. 禁用所有 XML 列。
3. 禁用所有 XML 集合。
4. 删除所有包含 XML 数据（您在步骤 1 中保存的数据）的模式。
5. 删除所有以前使用 XML Extender UDT 定义的列。
6. 对 XML Extender 禁用 SYSBAS 数据库。
7. 注销。
8. 注册。
9. 对 IASP 组执行 SETASPGRP。
10. 对 XML Extender 启用 IASP 数据库。
11. 使用『XML 列和 XML 集合的保存和恢复』中的指导来恢复 XML 数据。

XML 列和 XML 集合的保存和恢复

在 iSeries 上，模式的保存和恢复过程具有下列限制:

- 不保存、恢复或删除 DB2XML 模式（库）。
- 当恢复包含 XML Extender 所使用的数据库文件的用户创建模式时，下列条件适用:

- 倘若新系统上的数据库已对 XML Extender 启用，则可以使用 SAVLIB 和 RSTLIB 命令来在库级别恢复包含 XML 集合但不包含启用 XML 的列的模式。如果在旧系统上启用了 XML 集合，则必须在新系统上重新启用 XML 集合。
- 倘若尚未对 XML 启用列，并且新系统上的数据库已对 XML Extender 启用，则可以在库级别恢复包含具有 XML 用户定义的类型（XMLCLOB 和 XMLVarchar, 等等）的列的模式。
- 不能在库级别恢复包含已对 XML 启用的列的模式。可使用 RSTOBJ 命令来在对象级别恢复基本表和副表（数据库文件）。

下列过程讲述如何恢复带有与 XML 集合和 XML 列配合使用的数据库文件的模式。

要恢复 XML 集合数据库文件:

1. 在 XML Extender 的目标系统上启用数据库。
2. 使用 RSTLIB 来恢复 XML 集合数据库文件。
3. 如果在原始系统上启用了 XML 集合，则运行 enable_collection 命令来在目标系统上启用 XML 集合。

要恢复具有 XML 用户定义的类型的数据库文件:

1. 在 XML Extender 的目标系统上启用数据库。
2. 使用 RSTLIB 命令来恢复数据库文件。

要恢复 XML 列数据库文件:

1. 在 XML Extender 的目标系统上启用数据库。
2. 使用 RSTOBJ 命令来恢复基本表。
3. 使用 RMVPFTRG 命令来除去基本表中以前定义的任何旧触发器。
4. 在目标系统上启用 XML 列。如果在上一系统上使用了 “-r” 参数来启用基本表，则必须使用 “-r” 参数来标识基本表的主键。
5. 使用 ADDPFTRG 命令来将用户定义的触发器添加至基本表，并在目标系统上恢复那些程序。
6. 使用 RSTOBJ 命令来将数据恢复至副表。

限制: 当数据库文件包含启用 XML 的列时，不能使用 RSTLIB 来恢复它们，原因如下:

- 在恢复库和数据库文件时，不会将存储在 XML Extender 中的重要元数据恢复至新系统。只能通过运行 enable_column 命令来在目标系统上创建此元数据。
- 当使用 RSTLIB 来恢复库时，由于 XML Extender 中将丢失必备的元数据，所以库中的 SQL 触发器将不可使用。这些触发器的存在将使您无法运行 enable_column 命令。

为 iSeries 设置 XML Extender 样本和开发环境

下列章节描述如何设置管理环境，它取决于您计划对您的应用程序所使用的方法

- 对于所有环境 – 第 33 页的『解压缩并恢复样本文件和入门文件』
- 对于所有环境 – 第 34 页的『为样本创建 SQL 集合（模式）』
- 对于您选择的管理环境:
  - 当使用向导时 – 第 34 页的『设置向导』

- 当使用 Qshell 命令行时 - 第 34 页的『设置 Qshell』
- 当使用“导航器”时 - 第 34 页的『设置 iSeries 导航器界面』
- 当使用 OS 命令行时 - 第 35 页的『为 iSeries 命令行准备样本程序』

• 要运行入门课程 - 第 35 页的『为 iSeries 设置教程环境』

解压缩并恢复样本文件和入门文件

这些样本是在产品目录中作为两个 iSeries“保存文件”对象交付的。这些文件有:

QDBXM/QZXMSAMP1

包含 DXXSAMPLES 库的 SAVLIB 保存文件。该库包含用于应用程序开发的样本 C 和 CL 源代码、C 头文件和 SQL 语句。

QDBXM/QZXMSAMP2

包含 IFS 目录树的 SAV 保存文件，该目录树将包含样本 XML、DTD 和“数据访问定义”（DAD）文件以及要与“导航器”一起使用的自解压 `GetStart.exe` 文件。

准备使用管理环境的第一步是让 iSeries 管理员将这些保存文件解压缩并恢复到您的系统中。

管理员应:

• 解压缩 QDBXM/QZXMSAMP1“保存文件”以将“样本”源代码和 SETUP 程序恢复到您的系统中。

从 OS 命令行输入:

```
RSTLIB SAVLIB(DXXSAMPLES)
   DEV(*SAVF)
   SAVF(QDBXM/QZXMSAMP1)
```

RSTLIB 命令解压缩 DXXSAMPLES 库中的保存文件，该库包含表 7 中列示的对象:

表 7. DXXSAMPLES 库对象

对象	类型	属性	描述
SETUP	PGM	CLP	编译样本程序并添加 IFS
H	FILE	PF-SRC	C 语言头文件
QCLSRC	FILE	PF-SRC	导航器的接口
QCSRC	FILE	PF-SRC	样本程序
SQLSTMT	FILE	PF-SRC	样本的 SQL 语句

• 解压缩 QDBXM/QZXMSAMP2“保存文件”以将 XML 文件和 GetStart.exe 恢复到用户系统中。

从 OS 命令行输入:

```
RST DEV('/qsys.lib/qdbxm.lib/qzxmsamp2.file')
   OBJ(('/QIBM/UserData/DB2Extenders/XML/Samples'))
```

RST DEV 命令将 XML 文件的保存文件恢复到 IFS 目录 `/QIBM/UserData/DB2Extenders/XML/Samples`。

在样本设置期间会创建符号链接 /dxxsamples，它指向 IFS 目录 /QIBM/UserData/DB2Extenders/XML/Samples。本书中使用的术语 *dxxsamples* 指的就是这些值中的一个。

为样本创建 SQL 集合（模式）

要运行样本需有一个模式，因为将在此模式中创建带样本数据的存储过程和表的集合。

在创建 SQL 模式时，因为 SQL 中有缺省的模式规则，建议将模式命名为您在运行样本时将使用的用户标识。

如果已有与此用户标识相匹配的 SQL 模式，则无需创建新的模式。

要创建模式:

1. 从 OS 命令行，通过输入以下命令来打开 SQL 会话:

   ```
   STRSQL
   ```

2. 从 SQL 会话输入:

   ```
   CREATE SCHEMA UserId
   ```

   其中，*UserId* 是在运行样本时要使用的用户标识。

为 iSeries 设置管理工具

可以用于管理工具的工具已在第 29 页的『管理环境』中作了描述。您可以选择这些环境中的任何一个来执行管理任务。其中的某些环境需要进行设置才能与 XML Extender 管理命令、存储过程和样本配合使用。下列章节描述了设置需求。

设置向导

访问 XML Extender Web 站点:

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

设置 Qshell

除了安装 Qshell 选项外，无需设置。

设置 iSeries 导航器界面

您可以使用 “iSeries 导航器” 来运行管理命令、SQL 语句和存储过程，并可以使用它来完成 “入门” 课程。如果您计划使用 “iSeries 导航器”，请完成下列步骤以下载并构建样本程序。通过构建样本程序，可以为运行管理存储过程而准备环境，这是必需的。

1. 下载并解压缩 GetStart.exe 文件。
   a. 在 Windows 操作系统上创建一个名为 *path*/dxxsamples 的目录。*path* 是驱动器和目录，dxxsamples 目录将位于其中。
   b. 从 Windows 命令行输入:

      ```
      FTP SystemId
      ```

      其中，*SystemId* 是 iSeries 系统的主机名，您已经将保存文件恢复到该系统的 *dxxsamples* 目录中。

      输入请求的用户标识和密码。
   c. 输入以下 FTP 命令以更改为二进制方式: Binary。

d. 输入以下 FTP 命令以将入门 exe 文件移至 dxxsamples 目录:

```
get dxxsamples/getstart.exe  path/dxxsamples/getstart.exe
```

e. 使用以下命令关闭 FTP 会话:

```
exit
```

f. 从 *path*/dxxsamples 目录输入:

```
GetStart.exe
```

2. 启动 “iSeries 导航器”。
3. 展开 iSeries 系统的目录树，然后右键单击**数据库**。这将显示一个菜单。
4. 从菜单中单击**运行 SQL 脚本**。
5. 打开 *path*/dxxsamples/setup.sql 脚本文件。
6. 将出现的所有 &SCHEMA 更改为您在第 34 页的『为样本创建 SQL 集合（模式）』中创建的模式名:
   a. 从菜单中单击**编辑 -> 替换**。 “搜索并替换”窗口打开。
   b. 在“搜索并替换”窗口中，使用您的模式名替换 &SCHEMA 的所有出现。
7. 将 &DBNAME 的所有出现都更改为您将在其中运行样本程序的系统的 RDB 数据库名。要确定此名称，请从 OS 命令行运行 **WRKRDBDIRE** 命令。从已注册的数据库的列表中，选择远程地址为 *LOCAL 的名称。
   a. 从菜单中单击**编辑 -> 替换**。 “搜索并替换”窗口打开。
   b. 在“搜索并替换”窗口中，使用本地数据库名称替换 &DBNAME 的所有出现。
8. 保存 setup.sql 文件
9. 对每个 SQL 文件重复步骤 5-8。
10. 打开 *path*/dxxsamples/Setup.sql 脚本文件并单击**全部运行**。

您现在已准备好使用 “iSeries 导航器” 来开始 “入门” 课程了。

您刚刚构建的样本程序可用来输入管理命令和 DB2 UDB 命令。

为 iSeries 命令行准备样本程序

可以使用 OS 命令行来运行管理命令并完成 “入门” 课程。

运行管理命令无需进行设置。

如果您计划对样本和入门教程使用 OS 命令行，则运行 SETUP 来构建所有样本程序。从 OS 命令行输入:

```
CALL DXXSAMPLES/SETUP
```

为 iSeries 设置教程环境

下列章节描述设置环境以执行 “入门” 课程所需的步骤，这些课程将帮助您使用提供的样本来开发您自己的应用程序。

您可以使用下列环境来完成 “入门” 课程:

- iSeries 导航器
- OS 命令行

要使用这些环境，您必须:

1. 将样本源代码恢复到样本库中。使用第 33 页的『解压缩并恢复样本文件和入门文件』中的步骤。
2. 将 XML 样本文件和“入门”可执行文件恢复到 IFS 目录中。使用第 33 页的『解压缩并恢复样本文件和入门文件』中的步骤。
3. 创建一个“SQL 模式”（集合）。使用第 34 页的『为样本创建 SQL 集合（模式）』中的步骤。
4. 设置您将从中完成管理任务的环境。
   - 当使用“iSeries 导航器”时 – 第 34 页的『设置 iSeries 导航器界面』
   - 当使用 OS 命令行时 – 第 35 页的『为 iSeries 命令行准备样本程序』

XML Extender 管理计划

在为使用 XML 文档的应用程序作计划时，首先决定是否将要:

- 由数据库中的数据组成 XML 文档。
- 存储现有的 XML 文档。如果存储 XML 文档，则还必须决定是要将它们作为完整的 XML 文档存储在列中还是要将它们分解成常规 DB2® 数据。

作出此决定后，可以再决定:

- 是否验证 XML 文档
- 是否为了进行快速搜索和检索而为 XML 列数据建立索引
- 如何将 XML 文档的结构映射至 DB2 UDB 关系表

如何使用 XML Extender，取决于应用程序的需求是什么。可由现有的 DB2 UDB 数据组成 XML 文档并将 XML 文档作为完整的文档或作为 DB2 UDB 数据存储在 DB2 中。每个这样的存储和访问方法都具有不同的计划需求。

访问和存储方法

XML Extender 提供了两种将 DB2® 用作 XML 资源库的访问和存储方法: XML 列和 XML 集合。您需要决定哪种方法最符合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 UDB 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成 XML 文档。

应用程序的性质确定了哪种访问和存储方法最适合以及如何构造 XML 数据。

通过这两种访问和存储方法，使用 DAD 文件来将 XML 数据与 DB2 UDB 表相关联。第 37 页的图 4 显示了 DAD 是如何指定访问和存储方法的。

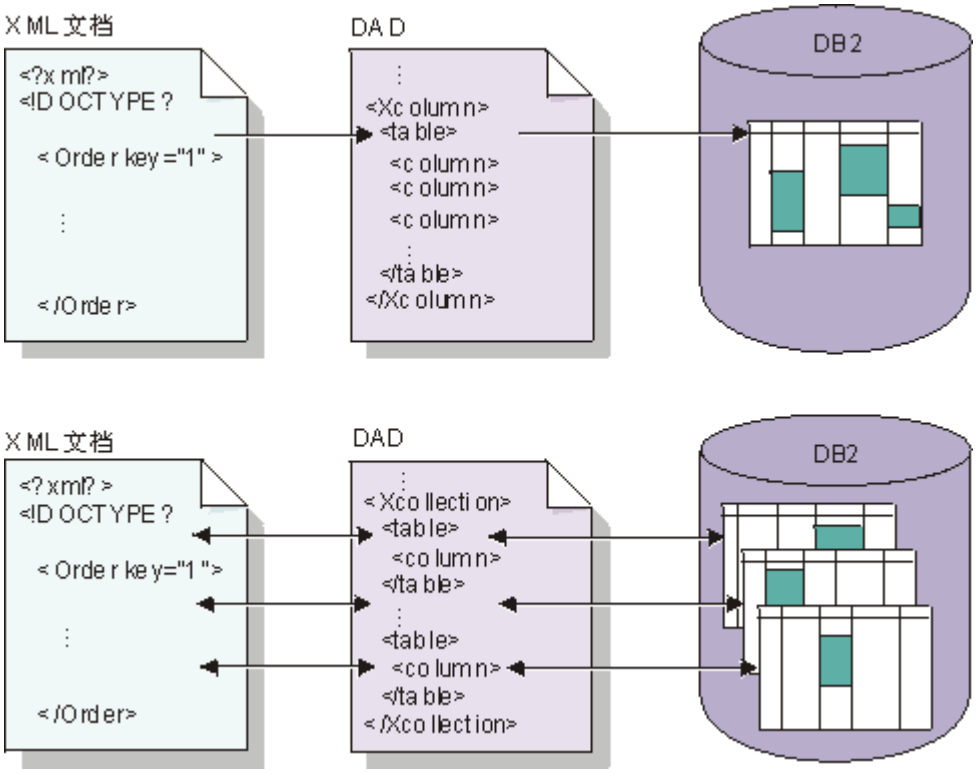


图 4. DAD 文件将 XML 文档结构映射至 DB2 UDB 关系数据结构，并指定访问和存储方法。

DAD 文件定义诸如 DTD 之类的密钥文件的位置，并指定 XML 文档结构与 DB2 UDB 数据相关的方式。最重要的是，它定义了您在应用程序中所使用的访问和存储方法。

相关概念:

- 第 37 页的『何时使用 XML 列方法』
- 第 38 页的『何时使用 XML 集合方法』

相关参考:

- 第 126 页的『XML Extender 中的存储函数概述』

何时使用 XML 列方法

在下列任何情况下 XML 列:

- XML 文档已经存在或来自外部数据源，并且您想以 Native-XML 格式存储这些文档。您想要在 DB2® 中存储它们，以便保持其完整性、以进行归档和审计。
- 频繁地读取 XML 文档，但不更新它们。
- 您想使用文件名数据类型来将 DB2 UDB 外部的 XML 文档存储在本地或远程文件系统中，并使用 DB2UDB 来进行管理和搜索操作。
- 您需要根据 XML 元素或属性的值来执行范围搜索，并且您知道什么元素或属性将频繁地用作搜索自变量。
- 文档包含带有大型文本块的元素，并且您想要在保持整个文档完整性的同时将 DB2 UDB Text Extender 用于结构化文本搜索。

何时使用 XML 集合方法

在下列任何情况下使用 XML 集合:

- 现有的关系表中有数据，并且您想要根据某个 DTD 来组成 XML 文档。
- 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方案来创建不同的关系数据视图。
- 您有来自其他数据源的 XML 文档。您关心的是数据而不是标记，并且要将纯数据存储在数据库中，并且，您想要灵活地决定是将数据存储在某些现有的表中还是存储在新表中。
- 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。

计划 XML 列

在开始使用 XML Extender 存储文档之前，需要了解 XML 文档的结构，以便您可以确定如何为文档中的元素和属性创建索引。在计划如何为文档创建索引时，需要确定:

- 存储 XML 文档将使用的 XML 用户定义的类型
- 您的应用程序将频繁搜索的 XML 元素和属性，以便可以将它们的内容存储在副表中并为这些内容创建索引以改进性能。
- 是否使用 DTD 来验证列中的 XML 文档
- 副表的结构以及将如何为它们创建索引

确定 XML 列的 XML 数据类型

XML Extender 提供了 XML 用户定义的类型，用户定义的类型用来定义存放 XML 文档的列。表 8 中描述了这些数据类型。

表 8. XML Extender UDT

用户定义的类型列	源数据类型	用法描述
XMLVARCHAR	VARCHAR(*varchar_len*)	将整个 XML 文档作为 VARCHAR 数据类型存储在 DB2 中。用于存储在 DB2 中的小文档。
XMLCLOB	CLOB(*clob_len*)	将整个 XML 文档作为 CLOB 数据类型存储在 DB2 中。用于存储在 DB2 中的大文档。
XMLFILE	VARCHAR(512)	将 XML 文档的文件名存储在 DB2 中，并将 XML 文档存储在 DB2 UDB 服务器本地的文件中。用于存储在 DB2 之外的文档。

确定要为其创建索引的元素和属性

在理解 XML 文档结构和应用程序的需要之后，您就可以确定要搜索哪些元素和属性。这些通常是将要最频繁地搜索或抽取的元素和属性，或者是查询成本最为高昂的那些元素或属性。可以将每个元素和属性的位置路径映射到关系表（副表），这些表在 XML 列的 DAD 文件中包含这些对象。然后为这些副表建立索引。

例如，表 9 显示了 XML 列的“入门”方案中的元素和属性的数据类型和位置路径的示例。已将数据指定为要频繁搜索的信息，位置路径指向包含数据的元素和属性。然后，可以将这些位置路径映射到 DAD 文件中的副表。

表 9. 要搜索的元素和属性

数据	位置路径
订单键	/Order/@key
客户	/Order/Customer/Name
价格	/Order/Part/ExtendedPrice
交付日期	/Order/Part/Shipment/ShipDate

DAD 文件

对于 XML 列，DAD 文件主要指定如何为存储在 XML 列中的文档创建索引，该文件是 XML 格式的文档，驻留在客户机上。DAD 文件指定在验证插入 XML 列中的文档时要使用的 DTD。DAD 文件的数据类型为 CLOB。此文件最大为 100 KB。

XML 列的 DAD 文件提供要存储在副表中，以用作创建索引的任何 XML 数据的映射。

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 数据将作为完整 XML 文档被检索并存储到 DB2 UDB 列中，这些 DB2 UDB 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。

计划 XML 集合

计划 XML 集合时，对于从 DB2 UDB 数据组成文档和／或将 XML 文档分解为 DB2 UDB 数据，其注意事项是不同的。下列各节描述了 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 或模式来验证 XML 数据。使用 DTD 或模式确保 XML 文档有效。

要使用 DTD 进行验证，您可能

需要在 XML Extender 资源库中有 DTD。请参阅第 49 页的『将 DTD 存储在资源库表中』以了解如何将 DTD 插入资源库。

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 不支持对已经插入 DB2 中的数据进行验证。

注意事项:

- 只可将一个 DTD 用于组合。
- 可以将多个模式用于组合。

- 如果不选择验证文档，则不处理由 XML 文档指定的 DTD。处理 DTD 以解析实体和属性缺省值很重要，即使无法验证处理文档片段。

DAD 文件

对于 XML 集合，DAD 文件将 XML 文档的结构映射至 DB2 UDB 表，您要从这些表中组成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成 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有表或是 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，可以是根元素或子元素。

text_node

表示元素的 CDATA 文本。

attribute_node

表示元素的属性。

第 41 页的图 5 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 "dxx_install/dtd/dad.dtd">
<DAD>
  ...
<Xcollection>
<SQL_stmt>
       ...
 </SQL_stmt> <prolog>?xml version="1.0"?</prolog>
<doctype>!DOCTYPE Order SYSTEM "dxx_install/sample/dtd/getstart.dtd
"</doctype> <root_node>
   <element_node name="Order">          --> Identifies the element <Order>
     <attribute_node name="key">        --> Identifies the attribute "key"
        <column name="order_key"/>      --> Defines the name of the column,
                                               "order_key", to which the element
                                                and attribute are mapped
              </attribute_node>
     <element_node name="Customer">     --> Identifies a child element of
                                               <Order> as <Customer>
        <text_node>                     --> Specifies the CDATA text for the
                                                element <Customer>
          <column name="customer">      --> Defines the name
                                                 to which the child
                                                  element is mapped
                 </text_node>
              </element_node>
        ...
              </element_node>

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

图 5. 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。

可使用 “XML Extender 管理” 向导或编辑器来创建和更新 DAD 文件。

XML 集合的映射方案

如果正在使用 XML 集合，必须选择一个*映射方案*，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。第 42 页的图 6 显示了层次结构与关系表列的映射关系。

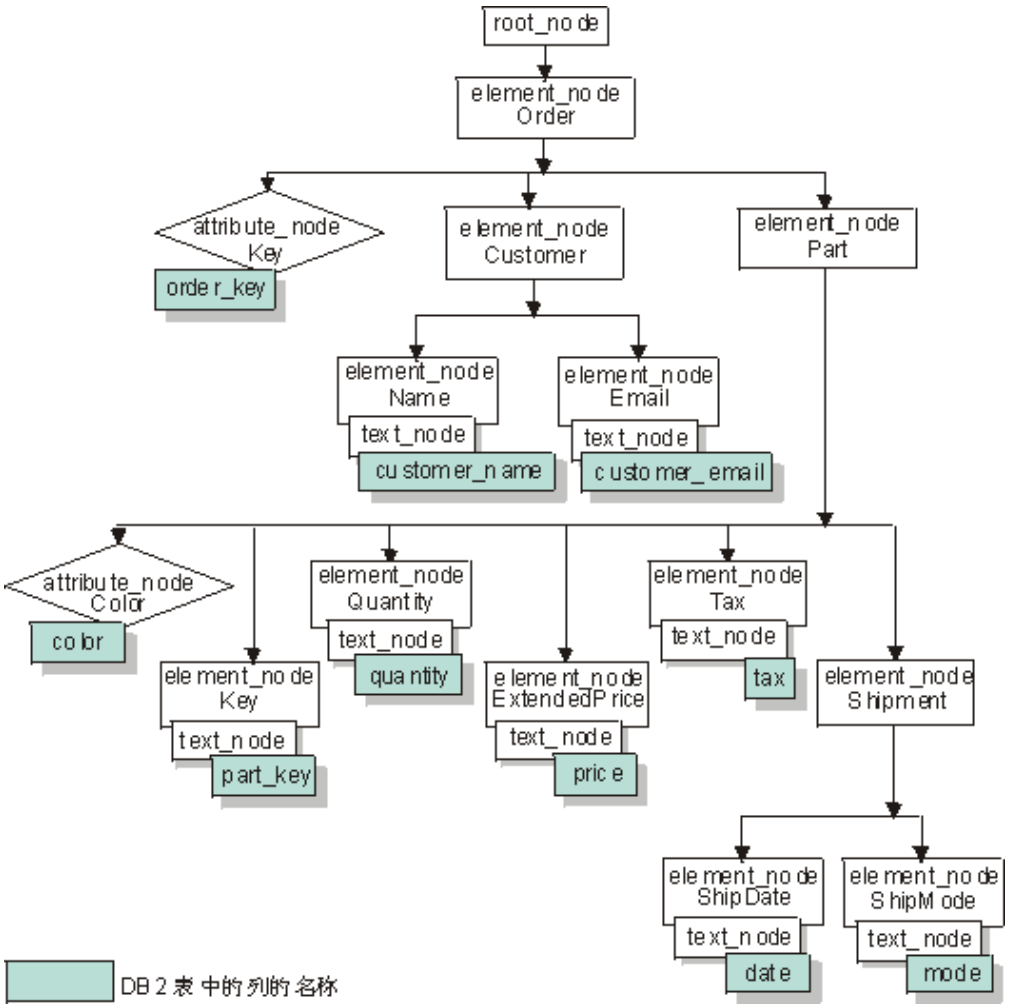


图 6. 映射至关系表列的结构化 XML 文档

XML Extender 在组成或分解位于多个关系表中的 XML 文档时，使用该映射方案。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。

映射方案的类型: 映射方案是在 DAD 文件的 <Xcollection> 元素中指定的。XML Extender 提供两种类型的映射方案：*SQL 映射*和*关系数据库（RDB_node）映射*。

SQL 映射

允许通过单个 SQL 语句和 *XPath 数据模型*，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。当为组成 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 UDB 用户能够使用 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，建议在这些表间使用*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。

此方法使用 XML Extender 提供的 *RDB_node*，它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。

要定义映射方案，用 <Xcollection> 元素创建 DAD。图 7 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成一组 XML 文档。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "dxx_install/dtd/dad.dtd">
<DAD>
  <dtdid>dxx_install/dad/getstart.dtd</dtdid>
   <validation>YES</validation>
  <Xcollection>
<SQL_stmt>
       SELECT o.order_key, customer, p.part_key, quantity, price, tax, date,
              ship_id, mode, comment
         FROM order_tab o, part_tab p,
               (select db2xml.generate_unique()                as ship_id, date, mode, from ship_t
 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 "dxx_install/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 7. SQL 映射方案

XML Extender 提供了若干个用于管理 XML 集合中的数据的存储过程。这些存储过程支持两种类型的映射，但需要 DAD 文件遵循『映射方案需求』中所描述的规则。

映射方案需求: 下列部分描述每一类型的 XML 集合映射方案的需求。

SQL 映射的需求

在此映射方案中，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用户定义的函数 generate_unique()，对该表生成一个这样的键。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**
  - 使用表表达式和用户定义的函数 generate_unique()，以对不具有主单键的表生成单键。例如:

    ```
    FROM order_tab as o, part_tab as p,
           (select db2xml.generate_unique() as        ship_id, date, mode from ship_tab) a
    ```

    在此例中，使用函数 generate_unique() 生成了单个列候选键，并取别名为 ship_id。
  - 当需要使一列有别于其它列时，使用别名。例如，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，它们唯一地标识数据库的实体关系设计中的实体。可使用表表达式和函数 generate_unique 或用户定义的函数（UDF）生成标识。
  - 维护实体自顶向下的层次结构。在 ORDER BY 子句中指定的列必须是对每个实体列示的第一列。保持该顺序可确保要生成的 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。

对根节点条件的谓词没有定序限制。

- **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:

  – 条件语句中允许行结束字符。

  – 条件元素可引用列名无限次。

  – 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，则条件元素可为空或丢失。

  – 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"/>
```

  当组成文档时，忽略对分解指定的信息。

  – 使用 orderBy 属性来重新组成 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顺序的键的列名。orderBy 属性是 DAD 文件中的表元素的一部分，且它是可选属性。

  您必须显式地拼出表名和列名。

- **每个 attribute_node 和 text_node 的 RDB_node**

  在此映射方案中，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。允许的条件是:

  - columname
  - operator
  - literal

  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"/>
```

  当组成文档时，忽略对分解指定的信息。

• 维护实体自顶向下的层次结构。这意味着确保正确地嵌套这些元素节点，以便 XML Extender 在组成或分解元素时了解各个元素之间的关系。例如，使用以下 DAD 文件时，该文件没有将 Shipment 嵌套在 Part 中:

```
               <element_node name="Part">
        ...
         <element_node name="ExtendedPrice">
              ...
           </element_node>
             ...
</element_node> <!-- end of element Part -->

  <element_node name="Shipment" multi_occurrence="YES">
           <element_node name="ShipDate">
                ...
           </element_node>
           <element_node name="ShipMode">
                ...
           </element_node>

    </element_node> <!-- end of element Shipment-->
```

它可能会生成一个 XML 文件，该文件中的 Part 和 Shipment 是兄弟元素。

```
<Part color="black ">
      <key>68</key>
<Quantity>36</Quantity>
<ExtendedPrice>34850.16</ExtendedPrice>
<Tax>6.000000e-2</Tax>
  </Part>
```

```
<Shipment>
   <ShipDate>1998-08-19</ShipDate>
  <ShipMode>BOAT  </ShipMode>
</Shipment>
```

当您宁愿具有将 Shipment 嵌套在 Part 中的 DAD 时:

```
            <element_node name="Part">
   ...
 <element_node name="ExtendedPrice">
          ...
           </element_node>
   ...
  <element_node name="Shipment" multi_occurrence="YES">
           <element_node name="ShipDate">
             ...
           </element_node>
           <element_node name="ShipMode">
             ...
           </element_node>

    </element_node> <!-- end of element Shipment-->
</element_node> <!-- end of element Part -->
```

它生成一个 XML 文件，其中 Shipment 是 Part 的子元素:

```
<Part color="black ">
      <key>68</key>
<Quantity>36</Quantity>
<ExtendedPrice>34850.16</ExtendedPrice>
<Tax>6.000000e-2</Tax>
<Shipment>
   <ShipDate>1998-08-19</ShipDate>
  <ShipMode>BOAT  </ShipMode>
</Shipment>
  </Part>
```

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。

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 UDB 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 10240 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 10240 行。如果该表包含有多个文档的行，则对于每个文档，该表最多可有 10240 行。

使用多次出现的元素（具有在 XML 结构中可以多次出现的位置路径的元素）会影响为每个文档插入的行数。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。当使用多次出现的元素时，认为从单个文档中最多只能将 10240 行分解到一个表中。

自动验证 XML 文档

选择访问和存储方法（XML 列或 XML 集合）之后，您可以确定是否*验证* XML 文档。除非您正在为归档而存储 XML 文档，否则建议您先验证这些文档，然后再将它们存入 DB2。也可以验证由 XML 集合组成的 XML 文档。

通过在 DAD 文件中将验证指定为 YES，可以自动验证 XML 数据。要在文档存储到 DB2 时验证它们，您必须在 <dtdid> 元素中或者原始文档的 <!DOCTYPE> 规范中指定一个 DTD。在从 DB2 中的 XML 集合组成文档时，要验证此文档，您必须在 DAD 文件中的 <dtdid> 元素内或 <doctype> 元素内指定一个 DTD。

在决定是否验证文档时，应考虑下列因素。

- 要验证带有模式的 DAD，插入将此 DAD 文件与模式文件相关联的模式标记。例如:

  ```
  <schemabindings>
  <nonamespacelocation location="path/schema_name.xsd"/>
  </schemabindings>
  ```

  仅当您决定验证 XML 文档时，此 DTD 标识或模式才有用。
- 存储或归档 XML 文档不需要 DTD。
- 在将 XML 数据插入 DB2 之前，必须决定是否进行验证。XML Extender 不会验证已插入 DB2 中的数据。
- 无论您是否选择进行验证，都有必要处理 DTD 来设置实体值和属性缺省值。
- 如果对“在 DAD 中进行验证”指定 NO，则不处理 XML 文档指定的 DTD。
- 验证 XML 数据会影响性能。

对 XML 启用数据库

在使用 XML Extender 来将 XML 文档存储在 DB2 UDB 中或从 DB2 UDB 检索 XML 文档之前，应对 XML 启用数据库。XML Extender 将启用您当前连接的数据库。

当对 XML 启用数据库时，XML Extender 执行下列操作:

- 创建所有用户定义的类型（UDT）、用户定义的函数（UDF）和存储过程
- 创建控制表并用 XML Extender 所需的必需元数据填充控制表
- 在用户定义的表空间中创建 DB2XML 模式，并指定必需的特权。

  XML 函数的全限定名是 db2xml.*function-name*，其中 db2xml 是一个标识，它提供 SQL 对象的逻辑分组。可以在任何引用 UDF 或 UDT 的地方使用全限定名。在引用 UDF 或 UDT 时还可省略模式名；在这种情况下，DB2 UDB 使用函数路径来确定函数或数据类型。

对于 iSeries，可以使用 *SYSBAS 数据库或独立辅助存储池（IASP）来对 XML Extender 启用数据库。可以对 XML Extender 启动多个 IASP 数据库。每次只能有一个 IASP 数据库处于活动状态。如果启用了 *SYSBAS 数据库，则不能对 XML Extender 启用 IASP 数据库。

过程:

可以使用“管理”向导来启用数据库，也可以从命令行启用数据库。

以下示例启用名为 SALES_DB 的现有数据库。

```
CALL QDBXM/QZXMADM PARM(enable_db SALES_DB)
```

要使用“管理”向导来启用数据库，需要完成下列任务:

1. 启动“管理”向导并在“启动板”窗口中单击**启用数据库**。

如果已经启用了数据库，则按钮将是**禁用数据库**。如果已经禁用了数据库，则按钮将是**启用数据库**

启用数据库后，您将返回到“启动板”窗口中。

启用数据库之后，将能够使用 XML Extender 来将 XML 文档存储在 DB2 UDB 中以及从 DB2 UDB 检索 XML 文档。

相关概念:

- 第 30 页的『将 XML Extender 从版本 7 迁移到版本 8』

创建 XML 表

此任务是更大型的定义和启用 XML 列任务的一部分。

XML 表用来存储完整的 XML 文档。要使用 DB2 UDB XML Extender 来将整个文档存储在数据库中，必须创建一个表，以使其包含具有 XML 用户定义的类型（UDT）的列。DB2 UDB XML Extender 提供了三个用户定义的类型，将 XML 文档作为列数据存储。这些 UDT 是: XMLVARCHAR、XMLCLOB 和 XMLFILE。当表包含 XML 类型的列时，可为 XML 启用此表。

可使用管理向导或命令行来创建新表，以添加 XML 类型的列。

过程:

要使用命令行来创建带有 XML 类型的列的表:

打开 DB2 UDB 命令提示符，并输入 Create Table 语句。

例如，在 sales 应用程序中，您可能想要将 XML 格式化的行项订单存储在名为 SALES_TAB 的表的名为 ORDER 的列中。此表还有 INVOICE_NUM 和 SALES_PERSON 列。因为这是一个小订单，所以使用 XMLVARCHAR 类型来存储销售订单。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 PULL PRIMARY KEY,
            sales_person   varchar(20),
            order          XMLVarchar);
```

在创建表之后，下一个步骤是对 XML 数据启用列。

相关概念:

- 第 53 页的『计划副表』
- 第 197 页的第 11 章，『XML Extender 管理支持表』

将 DTD 存储在资源库表中

可使用 DTD 来验证 XML 列或 XML 集合中的 XML 数据。可将 DTD 存储在 DTD 资源库表中，DTD 资源库表是名为 DTD_REF 的 DB2 UDB 表。DTD_REF 表具有模式名 DB2XML。DTD_REF 表中的每个 DTD 都具有唯一标识。在对 XML 启用数据库时，XML Extender 将创建 DTD_REF 表。可从命令行或通过使用管理向导插入 DTD。

过程:

要使用管理向导来插入 DTD:

1. 启动管理向导并从“启动板”窗口中单击**导入 DTD**，以将现有的 DTD 文件导入到当前数据库的 DTD 资源库中。这将打开“导入 DTD”窗口。
2. 在 **DTD 文件名**字段中指定 DTD 文件名。
3. 在 **DTD 标识**字段中输入 DTD 标识。

   DTD 标识是 DTD 的标识。它也可以是指定 DTD 在本地系统上的位置的路径。DTD 标识必须与 <DTDID> 元素的 DAD 文件中指定的值相匹配。
4. 可选: 在**作者**字段中输入 DTD 的作者姓名。
5. 单击**完成**以将 DTD 插入到 DTD 资源库表 DB2XML.DTD_REF 中并返回至“启动板”窗口。

要从命令行插入 DTD，请从表 10 中发出 SQL INSERT 语句。例如:

```
INSERT into DB2XML.DTD_REF values('/dxxsamples/dtd/getstart.dtd',
db2xml.XMLClobFromFile('/dxxsamples/dtd/getstart.dtd'),
0, 'user1', 'user1', 'user1');
```

表 10. DTD 资源库表的列定义

列名	数据类型	描述
DTDID	VARCHAR(128)	DTD 的标识。
CONTENT	XMLCLOB	DTD 的内容。
USAGE_COUNT	INTEGER	数据库中使用此 DTD 来定义 DAD 的 XML 列和 XML 集合的数目。
AUTHOR	VARCHAR(128)	DTD 的作者，这是用户可输入的可选信息。
CREATOR	VARCHAR(128)	执行第一次插入的用户标识。
UPDATOR	VARCHAR(128)	执行上次更新的用户标识。
ROW_ID	ROWID	行的标识。

启用 XML 列

要将 XML 文档存储在 DB2 UDB 数据库中，必须对 XML 启用将要包含文档的列。启用列时即准备为该列建立索引，以便可快速搜索该列。可使用 XML Extender “管理”向导或命令行来启用列。该列必须具有 XML 类型。

当 XML Extender 启用 XML 列时，它执行下列操作:

- 读取 DAD 文件，以便:
  - 如果指定了 DTDID 的话，检查 DTD_REF 表中的 DTD 的存在情况。
  - 对 XML 列创建副表，以便建立索引。
  - 准备列以包含 XML 数据。
- 有选择地创建 XML 表和副表的*缺省视图*。缺省视图显示应用程序表和副表。

  列名限制: 对于 iSeries，视图中的列大小限制是 10 个字符。
- 指定根标识列（如果尚未指定的话）。

在启用 XML 列之后，您可以:

- 对副表创建索引
- 将 XML 文档插入到 XML 列中
- 查询、更新或搜索 XML 列中的 XML 文档。

可以使用“管理”向导或从 DB2 命令行启用 XML 列。

过程（使用“管理向导”）：

要使用“管理”向导来启用 XML 列:

1. 设置并启动“管理”向导。
2. 在“启动板”窗口中单击**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**启用列**，然后单击**下一步**以启用现有的列。
4. 从**表名**字段中选择包含 XML 列的表。
5. 从**列名**字段中选择要启用的列。该列必须存在，且必须具有 XML 类型。
6. 在 **DAD 文件名**字段中输入 DAD 路径和文件名，或单击 **...** 以浏览现有的 DAD 文件。例如:

   dxx_install/dad/getstart.dad
7. （可选）在**表空间**字段中输入现有表空间的名称。

   表空间缺省值包含 XML Extender 创建的副表。如果指定了表空间，则将在指定的表空间中创建副表。如果未指定表空间，则在缺省表空间中创建副表。
8. （可选）在**缺省视图**字段中输入缺省视图的名称。

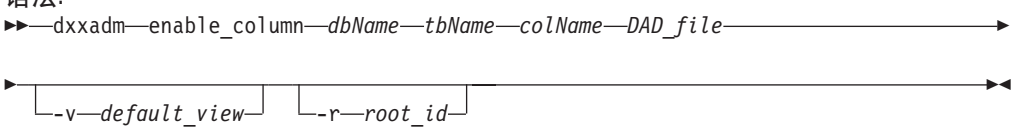
   如果指定了缺省视图，则在启用列时自动创建缺省视图并连接 XML 表和所有相关的副表。
9. （可选）在**根标识**字段中输入表的主键的列名称。建议进行此操作。

   XML Extender 将**根标识**的值用作唯一标识，以将所有副表与应用程序表相关联。XML Extender 将 DXXROOT_ID 列添加至应用程序表并生成标识。
10. 单击**完成**以启用 XML 列，创建副表，并返回至“启动板”窗口。
    - 如果启用列成功，您将接收到消息: 列已启用。
    - 如果未能成功地启用列，则会显示一则错误消息。必须更正 DAD 并再次启动启用过程。

过程（使用命令行）：

使用命令行来通过使用 DXXADM enable_column（本节对其语法和参数作了说明）来启用 XML 列

语法:



参数:

dbName

RDB 数据库的名称。

tbName

包含将要启用的列的表的名称。

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。

default_view

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

root_id 可选，但建议使用此项。应用程序表中主键的列名，即将所有副表与应用程序表相关联的唯一标识。称为 ROOT_ID。XML Extender 将 ROOT_ID 值用作唯一标识，以将所有副表与应用程序表相关联。如果未指定 ROOT_ID，则 XML Extender 将 DXXROOT_ID 列添加至应用程序表，并生成标识。

限制：如果应用程序表已有列名 DXXROOT_ID，则必须指定 *root_id* 参数；否则将发生错误。

示例:

从 Qshell:

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

从 OS 命令行:

```
CALL QDBXM/QZXMADM PARM(enable_column SALES_DB 'MYSCHEMA.SALES_TAB'
   ORDER 'getstart.dad' '-v' sales_order_view '-r' INVOICE_NUMBER)
```

从 “iSeries 导航器” :

```
CALL MYSCHEMA.QZXMADM('enable_column', 'SALES_DB', 'MYSCHEMA.SALES_TAB',
   'ORDER', 'getstart.dad', '-v sales_order_view', '-r INVOICE_NUMBER');
```

在本示例中，ORDER 列是在 SALES_TAB 表中启用的。DAD 文件为 getstart.dad，缺省视图为 sales_order_view，而 “根标识” 为 INVOICE_NUMBER。

使用本例，SALES_TAB 表具有以下各列:

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

将根据 DAD 规范创建下列副表:

ORDER_SIDE_TAB:

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

PART_SIDE_TAB:

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

SHIP_SIDE_TAB:

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

由于“根标识”是由应用程序表中的主键 INVOICE_NUM 指定的，所以所有副表都具有同一类型的 INVOICE_NUM 列。在启用列之后，当在主表中插入行时，将把 INVOICE_NUM 的值插入副表。在启用 XML 列 ORDER 时指定 *default_view* 参数将创建缺省视图 sales_order_view。该视图使用以下语句来连接上述各表:

```
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
```

计划副表

副表是 DB2® 表，它用于抽取将频繁被搜索的 XML 文档的内容。XML 列与用于存放 XML 文档内容的副表相关联。当在应用程序表中更新 XML 文档时，自动更新副表中的值。

第 54 页的图 8 显示了具有副表的 XML 列。

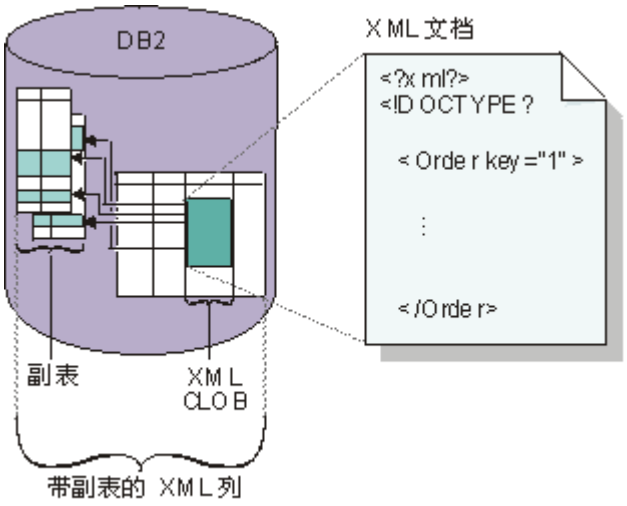


图 8. 在副表中映射了其内容的 XML 列。列中有一个 XML 文件与副表相关联，那些副表存放 XML 文档的内容。

计划副表时，必须考虑如何组织这些表、要创建多少个表及是否为副表创建缺省视图。对以上这些问题的决定取决于元素和属性是否可出现多次以及您对查询性能的需求。不要计划用任何方法来更新副表；当更新 XML 列中的文档时，将自动更新它们。

多次出现:

当元素和属性在副表中多次出现时，在您的计划中考虑下列问题:

- 对于 XML 文档中多次出现的元素或属性，由于 XML 文档具有复杂的结构，所以必须为多次出现的每个 XML 元素或属性创建单独的副表。这表示元素或属性带有多次出现的位置路径，并且必须映射至只带有一个列的表。表中不能有任何其他列。
- 当文档包含多次出现的位置路径时，XML Extender 将在每个副表中添加名为 DXX_SEQNO 且类型为 INTEGER 的列，以跟踪多次出现的元素的顺序。借助 DXX_SEQNO，可通过在 SQL 查询中指定 ORDER BY DXX_SEQNO 来检索与原始 XML 文档所具有的顺序相同的元素列表。

缺省视图和查询性能:

当启用 XML 列时，可使用唯一标识（称为根标识）来指定将应用程序表与副表相连的缺省只读视图。借助缺省视图，可通过查询副表来搜索 XML 文档。例如，如果您有应用程序表 SALES_TAB 以及副表 ORDER_TAB、PART_TAB 和 SHIP_TAB，则查询可能如下所示:

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

此 SQL 语句返回 SALES_TAB 中的那些在 ORDER 列中存储了订单并且 PRICE 列大于 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副表的虚拟单个视图。然而，创建的副表越多，查询的成本就越高。因此，建议仅当副表列的总数很少时，才创建缺省视图。应用程序可创建自己的、连接重要副表列的视图。

列名限制: 对于 iSeries，视图中的列大小限制是 10 个字符。要使用长名称，必须手工生成视图或使用别名。

为副表建立索引

此任务是更大型的定义和启用 XML 列任务的一部分。

副表在您创建 DAD 文件时指定的列中包含 XML 数据。在启用 XML 列和创建副表之后，可以为副表建立索引。建立这些表的索引有助于改进对 XML 文档进行的查询的性能。

过程:

要从 DB2 UDB 命令行为副表创建索引，使用 DB2 CREATE INDEX SQL 语句。

从 DB2 UDB 命令行。

以下示例使用 DB2 命令提示符对四个副表创建索引。

```
CREATE INDEX KEY_IDX
      ON ORDER_SIDE_TAB(ORDER_KEY)

CREATE INDEX CUSTOMER_IDX
      ON ORDER_SIDE_TAB(CUSTOMER)

CREATE INDEX PRICE_IDX
      ON PART_SIDE_TAB(PRICE)

CREATE INDEX DATE_IDX
      ON SHIP_SIDE_TAB(DATE)
```

通过使用 SQL 映射来组成 XML 文档

可从命令行使用 SQL 映射或通过使用管理向导来组成 XML 文档。

如果您正在组成 XML 文档，并且想使用 SQL 语句来定义要派生 XML 文档数据的表和列，则应使用 SQL 映射。SQL 映射仅可用于组成 XML 文档。创建 DAD 文件以使用 SQL 映射组成 XML 文档。

先决条件:

在组成文档之前，首先必须映射 DB2 UDB 表与 XML 文档之间的关系。此步骤包括映射 XML 文档的层次结构以及指定文档中的数据如何映射至 DB2 UDB 表。

过程:

要从命令行组成 XML 文档，请完成下列步骤:

1. 在文本编辑器中创建新文档并输入以下语法:

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

2. 插入 <DAD></DAD> 标记。

   此 DAD 元素将包含所有其他元素。

3. 插入用于验证带有 DTD 或模式的 DAD 的标记。
   - 要验证已组成的带有 DTD 的 XML 文档，插入将该 DAD 文件与 XML 文档 DTD 相关联的 DTDID 标记。例如:

     <dtdid>*path*/*dtd_name*.dtd>
   - 要验证已组成的带有模式的 XML 文档，插入将该 DAD 文件与模式文件相关联的模式标记。例如:

```
<schemabindings>
<nonamespacelocation location="path/schema_name.xsd"/>
</schemabindings>
```

仅当您决定验证 XML 文档时，此 dtd 或模式才有用。使用验证标记来指示 DB2 UDB XML Extender 是否验证 XML 文档:

- 如果要验证 XML 文档，则输入:

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

- 如果您不希望验证 XML 文档，则输入:

  ```
  <validation>NO</validation>
  ```

4. 输入 <XCollection> </XCollection> 标记来指定正在使用 XML 集合来作为 XML 数据的访问和存储方法。
5. 在 <Xcollection> </Xcollection> 标记内，插入 <SQL_stmt> </SQL_stmt> 标记以指定将关系数据映射至 XML 文档的 SQL 语句。此语句用于从 DB2 UDB 表中查询数据。以下示例显示了一个样本 SQL 查询:

   ```
   <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,
      (select db2xml.generate_unique()
       as ship_id, date, mode, part_key from ship_tab) as 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>
   ```

   这个将关系数据映射至 XML 文档的示例 SQL 语句具有以下语法:

   - 列是以自顶向下的顺序（即按 XML 文档结构的层次结构）来指定的。
   - 一个实体的列分组在一起。
   - 对象标识列是每组中的第一列。
   - Order_tab 表没有单一键列，因此，使用 generate_unique DB2 UDB 内置函数来生成 ship_id 列。
   - 然后，将对象标识列以自顶向下的顺序列示在 ORDER BY 语句中。ORDER BY 中的列不应被任何模式限定。而且列名必须与 SELECT 子句中的列名相匹配。
6. 添加以下要在组成 XML 文档中使用的 prolog 信息:

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

7. 输入 <doctype> </doctype> 标记。此标记包含将要针对其来验证已组成文档的 DTD 的路径。例如:

   ```
   <doctype>! DOCTYPE Order SYSTEM "dxxsamples/dtd/getstart.dtd"</doctype>
   ```

8. 指定 root 元素和组成 XML 文档的元素和属性:
   a. 添加 <root></root_node> 标记以定义根元素。构成 XML 文档的所有元素和属性都是在 root_node 中指定的。
   b. 使用 <element_node>、<attribute_node> 和 <text_node> 标记将 XML 文档中的元素和属性映射至与 DB2 UDB 数据相对应的元素和属性节点。

      <element_node> 标记

      指定 XML 文档中的元素。将 element_node 标记的名称属性设置到元素的名称。每个 element_node 可具有子 element_nodes。

<attribute_node> 标记

指定 XML 文档中元素的属性。属性嵌套在它们的元素节点中。将 attribute_node 标记的名称属性设置到属性的名称。

<text_node> 标记

指定元素的文本内容以及底层 element_nodes 的关系表中的列数据。对于每个底层元素，指定 <text_node> 标记，该标记指示在组成文档时，该元素包含从 DB2 中抽取的字符数据。对于每个底层 element_node，使用 <column> 标记来指定在组成 XML 文档时，从哪一列抽取数据。列标记通常位于 <attribute_node> 或 <text_node> 标记内。定义的所有列名都必须位于 DAD 文件开头处的 <SQL_stmt> SELECT 子句中。

9. 确保结束标记位于适当的位置:
   a. 确保结束 </root_node> 标记位于最后一个 </element_node> 标记之后。
   b. 确保结束 </Xcollection> 标记位于 </root_node> 标记之后。
   c. 确保结束 </DAD> 标记位于 </Xcollection> 标记之后。
10. 将文件保存为 *file.dad*。其中 *file* 是文件的名称。

以下示例显示了一个完整的 DAD:

```
<?xml version'"1.0">
<!DOCTYPE DAD SYSTEM "C:\dxx_xml\test\dtd/dad.dtd'>
<DAD>
<validation>NO</validation>
<Xcollection>
<SQL_stmt> select o.order_key, customer_name, customer_email,
 p.part_key, color, qty, price, tax, ship_id, date, mode from order_tab o,
 part_tab p, (select db2xml.generate_unique() 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 "dxxsamples\dtd/Order.dtd"
 </doctype>
 <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>
    <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="qty"/></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>
</element_node>
 </element_node>
</root_node>
</Xcollection>
</DAD>
```

通过使用 RDB_node 映射来组成 XML 集合

RDB_node 映射使用 <RDB_node> 标记来指定元素或属性节点的 DB2 UDB 表、列和条件。如果要使用类似于 XML 的结构来组成 XML 文档，则请使用此方法。<RDB_node> 使用下列元素:

table 定义与元素相对应的表。

column 定义包含对应元素的列。

condition （可选）指定列的条件。

在 RDB_node 元素中使用的子元素取决于该节点的上下文，并使用下列规则:

如果节点类型是:	使用下列 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] 对于多个表的情况，这是必需的

可使用管理向导或命令行来通过使用 RDB_node 映射来组成 XML 文档。

限制:

如果使用 RDB_node 映射来组成 XML 集合，则给定元素的所有语句都必须映射至同一个表中的列。

过程:

要从使用 RDB_node 映射的命令行来组成 XML 文档:

1. 打开文本编辑器，通过输入以下语法来创建 DAD 头:

   ```
   <?xml version="1.0"?>
   <!DOCTYPE DAD SYSTEM "path/dad.dtd">
   ```

   其中 *path*/dad.dtd 是 DAD 的 DTD 路径和文件名。
2. 插入 <DAD></DAD> 标记。此元素将包含所有其他元素。
3. 插入用于验证带有 DTD 或模式的 DAD 的标记。
   - 要验证带有 DTD 的 DAD，插入将该 DAD 文件与 XML 文档 DTD 相关联的 DTDID 标记。例如:

```
<dtdid>path/dtd_name.dtdid>
```

- 要验证带有模式的 DAD，插入将此 DAD 文件与模式文件相关联的模式标记。例如:

```
<schemabindings>
<nonamespacelocation location="path/schema_name.xsd"/>
</schemabindings>
```

仅当您决定验证 XML 文档时，此 dtdid 或模式才有用。使用验证标记来指示 DB2 UDB XML Extender 是否验证 XML 文档:

- 如果要验证 XML 文档，则输入:

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

- 如果您不希望验证 XML 文档，则输入:

```
<validation>NO</validation>
```

4. 插入 <XCollection> </XCollection> 标记来指定正在使用 XML 集合来作为 XML 数据的访问和存储方法。
5. 添加以下 prolog 信息:

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

6. 添加 <doctype></doctype> 标记。例如:

```
<doctype>! DOCTYPE Order SYSTEM "dxxsamples/dtd/getstart.dtd"</doctype>
```

7. 插入 <root_node></root_node> 标记。在 root_node 标记中，指定构成 XML 文档的元素和属性。
8. 在 <root_node> 标记内，将 XML 文档中的元素和属性映射至与 DB2 UDB 数据相对应的元素和属性节点。对 element_node、text_node 和 attribute_node 使用 RDB_node 元素。这些节点提供从 XML 数据至 DB2 UDB 数据的路径。要映射 XML 文档中的元素和属性:
   a. 对顶部 element_node 指定 RDB_node。此元素指定所有与 XML 文档相关联的表。要对顶部 element_node 指定 RDB_node，请在 root_node 标记之后插入 <RDB_node> 标记。
      - 对 attribute_node 指定 RDB_node。
      - 对 text_node 指定 RDB_node。
   b. 对包含要包括在 XML 文档中的数据的每个表定义表节点。例如，如果有三个表（ORDER_TAB、PART_TAB 和 SHIP_TAB）包含将要放入文档的列数据，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>
```

如果正在使用 DAD 文件来分解 XML 文档，则必须对每个表指定主键。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。如果将要启用集合，则还必须对每个表指定主键。下面的示例显示了如何对 element_node 中指定的每个表指定键列。

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

相关概念:

- 第 93 页的『XML 集合的映射方案』
- 第 100 页的『位置路径』
- 第 151 页的『XML 集合的 DAD 文件』
- 第 97 页的『RDB_Node 映射的需求』

相关任务:

- 第 60 页的『通过使用 RDB_node 映射来分解 XML 集合』
- 第 81 页的『管理 XML 集合中的数据』
- 第 90 页的『更新、删除和检索 XML 集合中的数据』

相关参考:

- 第 180 页的『XML Extender 组合存储过程』

通过使用 RDB_node 映射来分解 XML 集合

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 UDB 表、列和条件。<RDB_node> 使用下列元素:

table 定义与元素相对应的表。

column 定义包含对应元素的列。

condition （可选）指定列的条件。

在 <RDB_node> 中使用的子元素取决于该节点的上下文，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）对于多个表的情况，这是必需的

使用命令行的过程: :

要使用命令行分解 XML 文档:

1. 在任何文本编辑器中创建文件。通过输入以下语法来创建 DAD 标题:

   ```
   <?xml version="1.0"?>
   <!DOCTYPE DAD SYSTEM "path/dad.dtd">
   ```

   其中 *path*/dad.dtd 是 DAD 的 DTD 路径和文件名。

   必须将 DAD 文件存储在“集成文件系统”（IFS）目录中，或作为一个物理文件成员创建它，并在 IFS 目录中有指向该成员的链接。
2. 插入 <DAD></DAD> 标记。
3. 插入用于验证带有 DTD 或模式的 DAD 的标记。
   - 要验证带有 DTD 的 DAD，插入将该 DAD 文件与 XML 文档 DTD 相关联的 DTDID 标记。例如:

```
<dtdid>path/dtd_name.dtd>
```

- 要验证带有模式的 DAD，插入将此 DAD 文件与模式文件相关联的模式标记。例如:

```
<schemabindings>
<nonamespacelocation location="path/schema_name.xsd"/>
</schemabindings>
```

仅当您决定验证 XML 文档时，此 dtdid 或模式才有用。使用验证标记指示 DB2 UDB XML Extender 是否验证 XML 文档:

- 如果要验证 XML 文档，则输入:

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

- 如果您不希望验证 XML 文档，则输入:

```
<validation>NO</validation>
```

4. 插入 <XCollection> </XCollection> 标记来指定正在使用 XML 集合来作为 XML 数据的访问和存储方法。
5. 添加以下 prolog 信息:

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

6. 添加 <doctype></doctype> 标记。例如:

```
<doctype>! DOCTYPE Order SYSTEM "dxxsample/dtd/getstart.dtd"</doctype>
```

如果需要指定国际化的编码值，则添加 ENCODING 属性和值。

7. 使用 <root_node></root_node> 标记来定义 root_node。
8. 在 root_node 内，将 XML 文档中的元素和属性映射至与 DB2 UDB 数据相对应的元素节点和属性节点。这些节点提供从 XML 数据至 DB2 UDB 数据的路径。
   a. 定义顶级根 element_node。此 element_node 包含:
      - 表节点，且有一连接条件指定集合。
      - 子元素
      - 属性

      要指定表节点和条件:

      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:

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

      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>
```

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 文件必须既支持组成又支持分解。

b. 对 XML 文档中映射至 DB2 UDB 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 指定元素包含 DB2 UDB 表中的内容，而且该元素没有子元素。

attribute_node
指定属性。

child elements
element_node 的子元素。

text_node 包含将内容映射到 DB2 UDB 表和列名称的 RDB_node。

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

table 定义与元素相对应的表。

column 定义包含对应元素的列。

condition （可选）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> 标记以将 XML 元素映射至 DB2 UDB 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> 标记指定 Tax 元素的值为文本值，而数据存储在 TAX 列的 PART_TAB 表中。

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

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

该 attribute_node 具有 RDB_node，用于将属性值映射至 DB2 UDB 表和列。RDB_node 具有下列子元素:

table 定义与元素相对应的表。

column 定义包含对应元素的列。

condition （可选）指定列的条件。

例如，您可能具有 Order 元素的属性键。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>
```

9. 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>
```

10. 确保结束标记位于适当的位置:
    a. 确保结束 </root_node> 标记位于最后一个 </element_node> 标记之后。
    b. 确保结束 </Xcollection> 标记位于 </root_node> 标记之后。
    c. 确保结束 </DAD> 标记位于 </Xcollection> 标记之后。

相关任务:

- 第 86 页的『将 XML 文档分解为 DB2 UDB 数据』
- 第 180 页的『调用 XML Extender 组合存储过程』

相关参考:

- 第 191 页的『XML Extender 分解存储过程』

第 3 部分 编程

此部分描述了用于管理 XML 数据的编程技术。

第 3 章 XML 列

本章描述如何使用 DB2 来管理 XML 列中的数据。

管理 XML 列中的数据

当使用 XML 列来存储数据时，可将具有本机格式的整个 XML 文档作为 DB2 中的列数据进行存储。这种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建立索引和执行搜索、从文档中检索数据以及更新文档。

在对 XML 启用数据库之后，便可使用 XML Extender 提供的下列用户定义的类型（UDT）：

XMLCLOB
对作为字符大对象（CLOB）存储在 DB2 中的 XML 文档内容使用此 UDT。

XMLVARCHAR
对作为 VARCHAR 存储在 DB2 中的 XML 文档内容使用此 UDT。

XMLFile
对存储在本地文件系统上的文件中的 XML 文档使用此 UDF。

您可以创建或变更应用程序表，以使这些表带有 XML UDT 数据类型的列。这些表称为 XML 表。

在对 XML 启用表中的列之后，可创建 XML 列并执行下列管理任务:

- 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

要执行所有这些任务，请使用 XML Extender 提供的用户定义的函数（UDF）。使用缺省的数据类型转换函数来将 XML 文档存储在 DB2 中。缺省的数据类型转换函数将 SQL 基本类型强制转型为 XML Extender 用户定义的类型，并将数据类型（原点）的实例转换为另一数据类型（目标）的实例。

相关概念:

- 第 67 页的『XML 列作为存储和访问方法』
- 第 69 页的『对 XML 列数据使用索引』

XML 列作为存储和访问方法

有时候您想要存储文档结构并进行维护以使其具有当前状态。XML 包含创建一组文档必需的全部信息。

例如，如果贵公司是一家新闻发布机构，通过 Web 来发表文章，则可能想要维护已发布文章的档案。此种情况下，XML Extender 允许您在 DB2 ® 表的列（即 *XML 列*，如图 9 所示）中存储完整或部分 XML 文章。

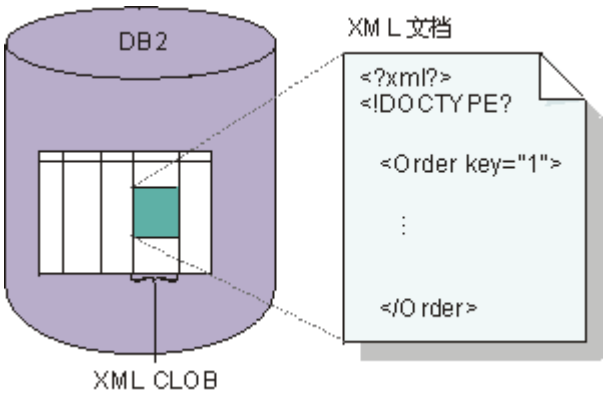


图 9. 在 DB2 UDB 表列中存储结构化 XML 文档

XML 列存储和访问方法允许您使用 DB2 来管理 XML 文档。可以将 XML 文档存储在具有 XML 类型的列中，并可以查询文档的内容以查找特定元素或属性。可以为一个或多个文档关联 DTD，并将其存储在 DB2 UDB 中。另外，可以将元素和属性内容映射至 DB2 UDB 表，称为*副表*。可以对这些副表建立索引以改进查询性能，但不会自动建立索引。用于存储文档的列称为 XML 列。它指定该列用于 XML 列存储和访问方法。

在文档访问定义（DAD）文件中，输入 <Xcolumn> 和 </Xcolumn> 标记来表示将要使用的存储和访问方法是 XML 列。然后，DAD 将映射要存储在副表中的 XML 元素和属性内容。

在开始使用 XML Extender 存储文档之前，需要了解 XML 文档的结构，以便您可以确定如何为文档中的元素和属性创建索引。在计划如何为文档创建索引时，需要确定:

- 存储 XML 文档将使用的 XML 用户定义的类型
- 您的应用程序将频繁搜索的 XML 元素和属性，以便可以将它们的内容存储在副表中并为这些内容创建索引以改进性能。
- 是否要使用 DTD 来验证列中的 XML 文档

定义和启用 XML 列

使用 XML 列来将整个 XML 文档存储在数据库中并进行访问。这种存储方法允许使用 XML 文件类型来存储文档，对副表中的列建立索引和查询或搜索 XML 文档。

如果不会频繁地更新文档，或者要存储完整无缺的 XML 文档，则当您想要将整个 XML 文档存储到 DB2 表列中时使用 XML 列。

如果要将 XML 文档结构映射至 DB2 UDB 表，以便可以从现有 DB2 UDB 数据组成 XML 文档或将 XML 文档分解成 DB2 数据，则应使用 XML 集合而不是 XML 列。

过程:

要从命令行定义和启用 XML 列:

1. 创建文档访问定义（DAD）文件。
2. 创建用于存储 XML 文档的表。
3. 对 XML 数据启用列。如果 DAD 指定了验证，则将列插入到 dtd_ref 表中。
4. 对副表建立索引。

将 XML 列创建为具有 XML 用户数据类型。在完成这些任务之后，将能够在该列中存储 XML 文档。然后，可以更新、搜索和抽取这些文档。

相关概念:

- 第 67 页的『XML 列作为存储和访问方法』
- 第 69 页的『对 XML 列数据使用索引』
- 第 47 页的『自动验证 XML 文档』
- 第 7 页的『课程: 将 XML 文档存储在 XML 列中』

相关任务:

- 第 149 页的『为 XML 列创建 DAD 文件』
- 第 49 页的『创建 XML 表』
- 第 50 页的『启用 XML 列』
- 第 55 页的『为副表建立索引』
- 第 67 页的『管理 XML 列中的数据』

对 XML 列数据使用索引

在使用 XML 列时需要作的一项重要计划决定就是是否为 XML 列文档对副表建立索引。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副表，然后对这些列创建索引。您必须确定需要为其创建索引的元素和属性。

通过对 XML 建立索引，可以使用数据库引擎中的本机 DB2® 索引支持来为频繁查询的一般数据类型（如整数、十进制数或日期）的数据建立索引。XML Extender 从 XML 文档抽取 XML 元素或属性的值，并将它们存储在副表中，以允许您对这些副表创建索引。可使用位置路径来指定副表的每一列，该位置路径标识 XML 元素或属性以及 SQL 数据类型。

当将 XML 文档存储在 XML 列中时，XML Extender 将自动填充副表。

为进行快速搜索，可使用 DB2 UDB *B 型树索引*技术创建这些列的索引。

在创建索引时，必须牢记下列注意事项:

- 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 元素或属性创建单独的副表。
- 可为一个 XML 列创建多个索引。
- 可将副表与使用根标识的应用程序表、应用程序表中的主键的列名以及将所有副表与应用程序表相关联的唯一标识相关联。您可决定是否想要应用程序表的主键为根标识，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，或由于某种原因您不想使用它，则 XML Extender 会变更该应用程序表以添加列 DXXROOT_ID，该列存储插入时创建的唯一标识。所有副表都有一个带有唯一标识的 DXXROOT_ID 列。如果将主键用作根标识，则所有副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 UDB Text Extender 启用了 XML 列，则还可使用 Text Extender 的结构化文本功能。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 为一般 SQL 数据类型建立的索引配合使用。

存储 XML 数据

通过使用 XML Extender，可以将完整的 XML 文档插入 XML 列中。如果定义副表，则 XML Extender 将自动更新这些表。当直接存储 XML 文档时，XML Extender 将把基本类型作为 XML 类型来存储。

先决条件:

- 确保创建或更新了 DAD 文件。
- 确定在存储文档时要使用的数据类型。
- 选择一种用来在 DB2® 表中存储数据的方法（数据类型转换函数或 UDF）。

指定 SQL INSERT 语句，该语句指定要包含 XML 文档的 XML 表和列。

XML Extender 提供了两种用来存储 XML 文档的方法：缺省数据类型转换函数和存储 UDF。

表 11 显示了何时使用每种方法。

表 11. XML Extender 存储函数

如果 DB2 UDB 基本类型是...	在 DB2 UDB 中存储为...			
	XMLVARCHAR	**XMLCLOB**	**XMLDBCLOB**	**XMLFILE**
VARCHAR	XMLVARCHAR()	N/A	N/A	XMLFile FromVarchar()
CLOB	N/A	XMLCLOB()	XMLDB CLOB, 数据类型转换函数	XMLFile FromCLOB()
FILE	XMLVarcha rFromFile()	XMLCLOB FromFile()	XMLDB CLOBFrom File, UDF	XMLFILE

用于存储 XML 数据的缺省数据类型转换函数

对于每个 UDT，都存在一个缺省数据类型转换函数，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 提供的数据类型转换函数来插入数据。第 71 页的表 12 显示了提供的数据类型转换函数:

表 12. XML Extender 缺省数据类型转换函数

数据类型转换函数	返回类型	描述
XMLVARCHAR(VARCHAR)	XMLVARCHAR	从 VARCHAR 的内存缓冲区输入
XMLCLOB(CLOB)	XMLCLOB	从 CLOB 或 CLOB 定位器的内存缓冲区输入
XMLFILE(VARCHAR)	XMLFILE	仅存储文件名

例如，以下语句将强制类型转换 VARCHAR 类型插入 XMLVARCHAR 类型:

```
INSERT INTO sales_tab
VALUES('123456', 'Sriram Srinivasan', DB2XML.XMLVarchar(:xml_buff))
```

用于存储 XML 数据的存储 UDF

对于每个 XML Extender UDT，都存在一个存储 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，如果要将 XML 文件文档作为 XMLCLOB 数据类型来导入到 DB2 UDB 中，则可以使用函数 XMLCLOBFromFile()。

表 13 显示了 XML Extender 提供的存储函数。

表 13. XML Extender 存储 UDF

存储用户定义的函数	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 数据类型的值。 可选: 指定文件的编码。
XMLCLOBFromFile()	XMLCLOB	从服务器上的文件中读取 XML 文档，并返回 XMLCLOB 数据类型的值。可选: 指定文件的编码。
XMLFileFromVarchar()	XMLFILE	将 XML 文档作为 VARCHAR 数据来从内存中读取，将该文档写至外部文件，并返回 XMLFILE 数据类型的值，即文件名。可选: 指定外部文件的编码。
XMLFileFromCLOB()	XMLFILE	将 XML 文档作为 CLOB 数据或作为 CLOB 定位器来从内存中读取，将该文档写至外部文件，并返回 XMLFILE 数据类型的值，即文件名。可选: 指定外部文件的编码。

例如，通过使用 XMLCLOBFromFile() 函数，以下语句将记录作为 XMLCLOB 存储在 XML 表中。

```
EXEC SQL INSERT INTO sales_tab(ID, NAME, ORDER)
    VALUES('1234', 'MyName',
      XMLCLOBFromFile('dxxsample/xml/getstart.xml'))
```

本示例从名为 dxxsamples/xml/getstart.xml 的文件中将 XML 文档导入到表 SALES_TAB 的 ORDER 列中。

检索 XML 文档的方法

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。在直接检索 XML 列时，XML Extender 将 UDT 作为列类型返回。有关检索数据的详细信息，请参阅下列各节:

- 『检索整个 XML 文档』
- 第 74 页的『从 XML 文档检索元素内容和属性值』

XML Extender 提供了两种方法来检索数据: 缺省数据类型转换函数和 Content() 重载 UDF。表 14 显示了何时使用每种方法。

表 14. XML Extender 检索函数

当 XML 类型为 ...	从 DB2 UDB 检索，作为...			
	VARCHAR	**CLOB**	**DBCLOB**	**FILE**
XMLVARCHAR	**VARCHAR**	N/A	N/A	Content() UDF
XMLCLOB	N/A	**XMLCLOB**	N/A	Content() UDF
XMLFILE	N/A	Content() UDF	N/A	FILE

检索整个 XML 文档

过程:

要检索整个 XML 文档:

1. 确保已经在 XML 表中存储了 XML 文档，并确定要检索哪些数据。
2. 选择一种用来检索 DB2 UDB 表中的数据的方法（数据类型转换函数或 UDF）。
3. 如果使用的是重载的 Content() UDF，则确定正在检索的数据的数据类型，以及将要导出的数据类型。
4. 必须将要从中抽取元素或属性的 XML 列定义为 XMLVARCHAR、作为 LOCATOR 的 XMLCLOB 或者 XMLFILE 数据类型。

指定 SQL 查询，该查询指定要从哪些 XML 表和列中检索 XML 文档。

用于检索 XML 数据的缺省数据类型转换函数

DB2 UDB 为 UDT 提供的缺省数据类型转换函数将 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 提供的数据类型转换函数来检索数据。表 15 显示了提供的数据类型转换函数。

表 15. XML Extender 缺省强制类型转换函数

在 SELECT 子句中使用的数据类型转换	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

表 15. XML Extender 缺省强制类型转换函数 (续)

在 SELECT 子句中使用的数据类型转换	返回类型	描述
varchar(XMLFile)	VARCHAR	采用 VARCHAR 的 XML 文件名

例如，以下语句将检索 XMLVARCHAR，并将它作为 VARCHAR 数据类型存储在内存中:

```
EXEC SQL SELECT DB2XML.XMLVarchar(order) from SALES_TAB
```

使用 Content() UDF 来检索 XML 数据

使用 Content() UDF 来检索从外部存储器到内存的文档内容，或从内部存储器导出文档至外部文件（外部文件是 DB2 UDB 服务器上位于 DB2 UDB 外部的文件）。

例如，XML 文档可能是作为 XMLFILE 数据类型存储的。如果要在内存中对其进行操作，则可使用 Content() UDF，该函数可采用 XMLFILE 数据类型作为输入并返回 CLOB。

根据指定的数据类型不同，Content() UDF 将执行两种不同的检索功能。它可以:

- 从外部存储器检索文档并将其放入内存。

  当 XML 文档是作为外部文件存储的时，可以使用 Content() UDF 来将文档检索到内存缓冲区或 CLOB *定位器*（一个主变量，它的值表示数据库服务器中的单个 LOB 值）中。

  使用以下函数语法，其中，*xmlobj* 是正在查询的 XML 列:

  XMLFILE 至 CLOB:

  ```
  Content(xmlobj XMLFile)
  ```

- 从内部存储器检索文档并将其导出到外部文件中。

  可使用 Content() UDF 来检索作为 XMLCLOB 数据类型存储在 DB2 UDB 中的 XML 文档并将其导出到数据库服务器文件系统上的文件中。Content() UDF 将文件名作为 VARCHAR 数据类型返回。

  使用以下函数语法:

  XML 类型至外部文件:

  ```
  Content(xmlobj XML type, filename varchar(512), targetencoding varchar(100))
  ```

  其中:

  xmlobj 是将要从中检索 XML 内容的 XML 列的名称。*xmlobj* 可以具有 XMLVARCHAR 或 XMLCLOB 类型。

  filename
  是将要用来存储 XML 数据的外部文件的名称。

  targetencoding
  可选: 指定输出文件的编码。

在下面的示例中，一个带有嵌入式 SQL 语句（在应用程序中编码的 SQL 语句）的小型 C 程序段显示了如何将 XML 文档从文件检索到内存中。本示例假定 ORDER 列的数据类型是 XMLFILE。

```
             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;
```

从 XML 文档检索元素内容和属性值

可以从一个或多个 XML 文档（单个文档或集合文档搜索）检索（抽取）元素的内容或属性值。XML Extender 提供了用户定义的抽取函数，可以在 SQL SELECT 子句中对每一 SQL 数据类型指定这些函数。

在开发应用程序时，检索元素内容和属性值是非常有用的，因为这样您就可以将 XML 数据作为关系数据来进行访问。例如，在 SALES_TAB 表的 ORDER 列中可能存储了 1000 个 XML 文档。要检索所有已经订购了超过 $2500 的商品的客户的姓名，请使用以下在 SELECT 子句中带有抽取 UDF 的 SQL 语句:

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

在本示例中，抽取 UDF 从 ORDER 列检索 <customer> 元素的内容并将其作为 VARCHAR 数据类型进行存储。位置路径是 /Order/Customer/Name。另外，通过使用 WHERE 子句来减少所返回值的数目，该子句指定只返回子元素 <ExtendedPrice> 的值大于 2500.00 的 <customer> 元素的内容。

第 75 页的表 16 通过对标量函数使用以下语法来显示可用来抽取元素内容和属性值的 UDF。

语法:

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

retrieved_datatype

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

xmlobj 要从其中抽取元素或属性的 XML 列的名称。必须将这个列定义为下列 XML 用户定义的类型之一:

- XMLVARCHAR
- 作为 LOCATOR 的 XMLCLOB
- XMLFILE

path XML 文档中的元素或属性的位置路径（如 `/Order/Customer/Name`）。

限制: 抽取 UDF 可支持具有谓词的位置路径，但这些谓词只能带有属性而不能带有元素。例如，支持下列谓词:

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

不支持下列谓词:

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

表 16 显示了抽取函数，给出了每个抽取函数的标量格式和表格式。

表 16. XML Extender 抽取函数

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

标量函数示例: 在以下示例中，插入一个属性键值为 1 的值。将该值作为整数进行抽取，并自动转换为 DECIMAL 类型。

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

表函数示例: 在以下示例中，将销售订单的每个键值（[@0000]key）作为 INTEGER 进行抽取。

```
SELECT * from table(DB2XML.extractIntegers(DB2XML.XMLFile
   ('/dxxsamples/xml/getstart.xml'), '/Order/@key')) as x;
```

更新 XML 数据

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

过程

要更新 XML 数据:

1. XML 文档必须存储在 XML 表中。
2. 必须知道想要检索什么数据。
3. 必须选择更新 DB2 UDB 表中数据的方法（数据类型转换函数或 UDF）。
4. 指定 SQL 查询，该查询指定要更新的 XML 表和列。

更新整个 XML 文档

可使用缺省数据类型转换函数或存储 UDF 来更新 XML 文档。

使用缺省数据类型转换函数来进行更新

对于每个用户定义的类型（UDT），都存在缺省的数据类型转换函数来将 SQL 基本类型强制转型为该 UDT。可以使用 XML Extender 提供的数据类型转换函数来更新 XML 文档。

例如，以下语句通过对 VARCHAR 类型进行强制转型来更新 XMLVARCHAR 类型（假定 `xml_buf` 是定义为 VARCHAR 类型的主变量。

```
UPDATE sales_tab SET=DB2XML.XMLVarchar(:xml_buff)
```

使用存储 UDF 来更新 XML 文档

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

以下示例将 XML 对象从名为 `dxxsample/xml/getstart.xml` 的文件更新至 SALES_TAB 表的 ORDER 列。

```
UPDATE sales_tab
       set order = XMLVarcharFromFile('dxxsample
       /xml/getstart.xml) WHERE sales_person = 'MyName'
```

更新 XML 文档的特定元素和属性

使用 Update UDF 来进行特定的更改，而不是更新整个文档。当使用此 UDF 时，需指定将要替换其值的元素或属性的位置路径。不需要编辑 XML 文档；XML Extender 替您进行更改。

语法:

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

此语法具有下列组件:

xmlobj 要更新其元素值或属性值的 XML 列的名称。

path 要更新的元素或属性的位置路径。

value　将要更新的新值。

例如，以下语句将 <Customer> 元素的值替换为 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 中指定了位置路径时，将用提供的值更新每个具有匹配路径的元素或属性的内容。如果某个位置路径在文档中出现多次，则 Update UDF 将使用 *value* 参数中提供的值来替换所有的现有值。

搜索 XML 文档的方法

搜索 XML 数据与检索 XML 数据相似：两种技术都是检索数据以进一步进行处理，但是它们使用 WHERE 子句的内容作为检索条件来进行搜索。

XML Extender 提供了数种方法来搜索存储在 XML 列中的 XML 文档。您可以:

- 搜索文档结构并根据元素内容或属性值来返回结果。
- 搜索 XML 列及其副表的视图。
- 直接搜索副表以获得更好的性能。
- 使用带有 WHERE 子句的抽取 UDF 进行搜索。
- 使用 DB2® Text Extender 来搜索结构化内容内的列数据，以查找文本字符串。

借助 XML Extender，可使用索引来快速搜索副表中的列。这些列包含从 XML 文档中抽取的 XML 元素内容或属性值。通过指定元素或属性的数据类型，可以搜索 SQL 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 Text Extender 来执行结构化文本搜索或全文本搜索。例如，您可能有一个名为 RESUME 的列，它包含 XML 格式的简历。如果您想要查找所有具有 Java™ 技能的申请人的姓名，则可以使用 DB2 UDB 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。

以下章节描述搜索方法:

- 『按结构搜索 XML 文档』

按结构搜索 XML 文档

使用 XML Extender 搜索功能，可以根据文档结构（文档中的元素和属性）搜索某列中的 XML 数据。

过程:

要搜索数据，您可以:

- 直接查询副表。
- 使用*连接的视图*。
- 使用抽取 UDF。

在基于以下方案的示例中描述了这些搜索方法。SALES_TAB 表有一个名为 ORDER 的 XML 列。此列有三个副表：ORDER_SIDE_TAB、PART_SIDE_TAB 和 SHIP_SIDE_TAB。启用 ORDER 列时指定了缺省视图 sales_order_view。此视图使用以下 CREATE VIEW 语句将这些表连接在一起:

```
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
```

示例: 使用直接查询副表来进行搜索

当对副表进行索引时，对于结构化搜索，使用子查询搜索来直接查询将获得最佳性能。

过程:

可以使用查询或子查询来正确搜索副表。

例如，以下语句使用查询和子查询来直接搜索副表:

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

在本示例中，invoice_num 是 SALES_TAB 表中的主键。

示例: 从连接的视图中搜索

XML Extender 可以创建一个缺省视图，该缺省视图通过使用唯一的标识来连接应用程序表和副表。可以使用此缺省视图或连接应用程序表和副表的任何视图，来搜索列数据以及查询副表。此方法为应用程序表及其副表提供了单个虚拟视图。然而，创建的副表越多，运行查询的时间也越长。

技巧: 当创建您自己的视图时，可以使用根标识或 DXXROOT_ID（由 XML Extender 创建）来连接表。

例如，以下语句将搜索名为 SALES_ORDER_VIEW 的视图并从 SALES_PERSON 列中返回价格大于 2500.00 的行项订单的值。

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

示例: 使用抽取 UDF 来搜索

当您还没有为应用程序表创建索引或副表时，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 数据的成本昂贵，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。

下列语句使用抽取 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
```

在本示例中，抽取 UDF 将抽取包含子串 IBM 的 </Order/Customer/Name> 元素。

示例: 搜索多次出现的元素或属性

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

下列语句使用 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，并为它创建副表 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。

删除 XML 文档

使用 SQL DELETE 语句来从 XML 列中删除包含 XML 文档的行。可指定 WHERE 子句以删除特定的文档。

例如，以下语句将删除 <ExtendedPrice> 的值大于 2500.00 的所有文档:

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

将自动删除副表中的相应行。

相关概念:

- 第 67 页的『XML 列作为存储和访问方法』

相关任务:

- 第 67 页的『管理 XML 列中的数据』

从 Java 数据库调用函数时的限制

当在函数中使用参数标记时，JDBC 限制要求必须将函数的参数标记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。

例如，JDBC 不能解析下列代码:

```
DB2XML.XMLdefault_casting_function(length)
```

可以使用 CAST 规范来为参数标记提供类型，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，参数标记被强制转型为 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，参数标记被强制转型为 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 4 章 管理 XML 集合中的数据

XML 集合作为存储和访问方法

关系数据是从入局 XML 文档*分解*出来的，或用来*组成*出局 XML 文档。分解的数据是 XML 文档的无标记内容，此数据存储在一个或多个数据库表中。或者，根据一个或多个数据库表中的现有数据组成 XML 文档。如果将要与其他应用程序一起共享数据，则您可能希望能够组成和分解入局和出局 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 文档存储称为 *XML 集合*。

图 10 显示了 XML 集合的示例。

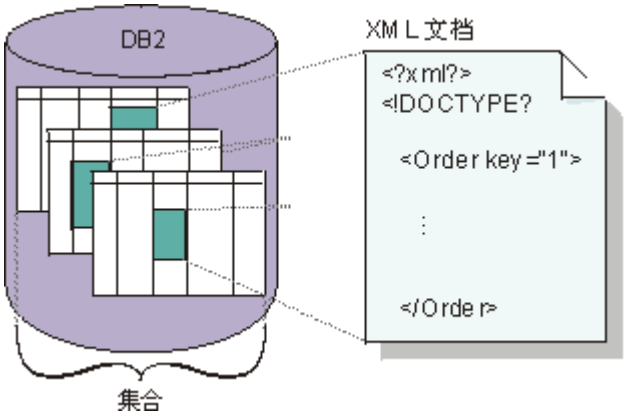


图 10. 将文档作为无标记数据存储在 DB2 UDB 表中

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。集合是与 DAD 文件相关联的列的集合，这些列包含特定 XML 文档中或一组 XML 文档中的数据。可以通过启用集合来定义集合名，然后在发出存储过程来组成或分解 XML 文档时通过该名称引用它。它被称为启用了 XML 的集合。为集合命名以便可以轻松地将其与组成和分解 XML 文档的存储过程一起运行。

在 DAD 文件中定义集合时，使用两类映射方案（*SQL 映射*或 *RDB_node 映射*）的其中之一，映射方案定义用来将 XML 数据与 DB2 UDB 表相关联的表、列和条件。SQL 映射使用 SQL SELECT 语句来定义用于集合的 DB2 UDB 表和条件。RDB_node 映射使用基于 XPath 的关系数据库节点或 RDB_node，它有子元素。

提供了存储过程来组成或分解 XML 文档。存储过程名称由 DB2XML（即 XML Extender 的*模式名*）限定。

管理 XML 集合中的数据

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有数据中组成 XML 文档，分解 XML 文档，并将 XML 用作交换方法。

构成集合的关系表可以是新表，也可以是带有要与 XML Extender 配合使用以便为应用程序组成 XML 文档的现有表。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。可以使用存储过程来存储、检索、更新、搜索和删除 XML 集合数据。

可以增大存储过程的结果的 CLOB 大小。

准备从 DB2 UDB 数据组成 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成 XML 文档。要使用这些存储过程，请创建文档访问定义（DAD）文件。DAD 文件指定 XML 文档与 DB2 表结构之间的映射。存储过程使用 DAD 文件来组成 XML 文档。

过程: :

开始组成 XML 文档之前:

1. 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。
2. 选择映射方法: SQL 映射或 RDB_node 映射。
3. 准备 DAD 文件。

XML Extender 提供了四个存储过程 dxxGenXML()、dxxGenXMLCLOB()、dxxRetrieveXML() 和 dxxRetrieveXMLCLOB 来组成 XML 文档。在选择将要使用的存储过程时，您所计划的更新 XML 文档的频率是一项关键因素。

组成将偶尔更新的 XML 文档

如果只是将偶尔地更新文档，则使用 dxxGenXML 存储过程来组成文档。不必启用集合即可使用此存储过程。存储过程使用 DAD 文件。

dxxGenXML 存储过程通过使用存储在 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到结果表中。还可以对结果表打开游标并访存结果集。结果表必须由应用程序来创建，并且始终具有 VARCHAR、CLOB、XMLVARCHAR 或 XMLCLOB 类型的一列。

另外，如果 DAD 文件中验证元素的值是 YES，则 XML Extender 将 INTEGER 类型的 DXX_VALID 列添加到结果表中（如果表中尚不存在 DXX_VALID 列的话）。对于有效的 XML 文档，XML Extender 插入值 1，对于无效的文档，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，以及任何返回码和消息。

用于分解的相应存储过程是 dxxShredXML；它也将 DAD 作为输入参数并且不要求启用 XML 集合。

过程:

要使用 dxxGenXML 存储过程来组成 XML 集合，请使用以下存储过程声明来在应用程序中嵌入存储过程调用:

```
dxxGenXML(CLOB(100K)     DAD,       /* input */
          char(32      ) resultTabName,     /* input */
               char(30)           result_column, /* input */
               char(30)       valid_column,     /* input */
               integer        overrideType,     /* input */
          varchar(1024)   override,     /* input */
               integer        maxRows,          /* input */
               integer        numRows,          /* output */
            long          returnCode,     /* output */
            varchar(1024) returnMsg)      /* output */
```

示例: 以下示例将组成一个 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 */

              char    result_tab[32];    /* name of the result table */
char    result_colname[32];    /* name of the result column */
char    valid_colname[32];     /* name of the valid column, will set to NULL */
              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   rcol_ind;
short   vcol_ind;
              short            ovtype_ind;
short   ov_ind;
              short            maxrow_ind;
              short            numrow_ind;
              short       returnCode_ind;
              short       returnMsg_ind;
EXEC SQL END DECLARE SECTION;

FILE   *file_handle;
long    file_length=0;

         /* create table */
         EXEC SQL CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

/* initialize the DAD CLOB object. */
file_handle = fopen("/dxx/dad
    /getstart_xcollection.dad", "r");
if ( file_handle != NULL ) {
  file_length = fread ((void *) &dad.data,
1, FILE_SIZE, file_handle);
  if (file_length == 0) {
    printf ("Error reading dad file
             /dxx/dad
              /getstart_xcollection.dad\n");
      rc = -1;
      goto exit;
 } else
    dad.length = file_length;
}
else {
  printf("Error opening dad file  \n", );
      rc = -1;
      goto exit;
}
```

```
        /* initialize host variable and indicators */
         strcpy(result_tab,"xml_order_tab");
strcpy(result_colname, "xmlorder")
valid_colname = '\0';
              override[0] = '\0';
              overrideType = NO_OVERRIDE;
         max_row = 500;
         num_row = 0;
        returnCode = 0;
         msg_txt[0] = '\0';
              dad_ind = 0;
         rtab_ind = 0;
rcol_ind = 0;
vcol_ind = -1;
              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 "DB2XML.dxxGenXML"
       (:dad:dad_ind;
                 :result_tab:rtab_ind,
                 :result_colname:rcol_ind,
                 :valid_colname:vcol_ind,
                 :overrideType:ovtype_ind,:override:ov_ind,
                 :max_row:maxrow_ind,:num_row:numrow_ind,
                 :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);

if (SQLCODE < 0) {
    EXEC SQL ROLLBACK;
  else
    EXEC SQL COMMIT;
}

exit:
  return rc;
```

因为 DAD 文件中指定的 SQL 查询生成了 250 个 XML 文档，所以，在调用存储过程后，结果表包含 250 行。

组成将频繁更新的 XML 文档

如果将要频繁地更新文档，则使用 dxxRetrieveXML 存储过程来组成文档。因为需要重复相同的任务，所以改进性能十分重要。

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，它的工作方式与 dxxGenXML 相同。启用 XML 集合后，DAD 文件存储在 XML_USAGE 表中。因此，XML Extender 检索 DAD 文件并使用它来以与 dxxGenXML 存储过程相同的方式组成文档。

dxxRetrieveXML 存储过程允许将同一 DAD 文件同时用于组合和分解。

用于分解的相应存储过程是 dxxInsertXML；它也采用已启用的 XML 集合的名称。

过程:

要使用 dxxRetrieveXML 存储过程来组成 XML 集合，请使用以下存储过程声明来在应用程序中嵌入存储过程调用:

```
dxxRetrieveXML(char() collectionName,  /* input */
               char() resultTabName,   /* input */
               char(30)           result_column, /* input */
               char(30)       valid_column,     /* input */
               integer        overrideType,     /* input */
          varchar(1024)   override,     /* input */
               integer        maxRows,          /* input */
               integer        numRows,          /* output */
            long          returnCode,     /* output */
            varchar(1024) returnMsg)      /* output */
```

示例: 以下示例是对 dxxRetrieveXML() 的调用。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    collectionName[32];    /* name of an XML collection */
              char    result_tab[32];    /* name of the result table */
char    result_colname[32];    /* name of the result column */
char    valid_colname[32];     /* name of the valid column, will set to NULL*/
              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   collectionName_ind;
              short            rtab_ind;
short   rcol_ind;
short   vcol_ind;
              short            ovtype_ind;
short   ov_ind;
              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);

/* initial host variable and indicators */
strcpy(collection, "sales_ord");
         strcpy(result_tab,"xml_order_tab");
strcpy(result_col,"xmlorder");
valid_colname[0] = '\0';
              override[0] = '\0';
              overrideType = NO_OVERRIDE;
         max_row = 500;
         num_row = 0;
        returnCode = 0;
         msg_txt[0] = '\0';
collectionName_ind = 0;
         rtab_ind = 0;
rcol_ind = 0;
vcol_ind = -1;
              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 "DB2XML.DXXRETRIEVE"
                 (:collectionName:collectionName_ind,
                 :result_tab:rtab_ind,
                 :result_colname:rcol_ind,
                 :valid_colname:vcol_ind,
                 :overrideType:ovtype_ind,:override:ov_ind,
                 :max_row:maxrow_ind,:num_row:numrow_ind,
                              :returnCode:returnCode_ind,
                              :returnMsg:returnMsg_ind);

if (SQLCODE < 0) {
    EXEC SQL ROLLBACK;
  else
    EXEC SQL COMMIT;
}
```

相关概念:

- 第 81 页的『XML 集合作为存储和访问方法』
- 第 93 页的『XML 集合的映射方案』
- 第 100 页的『位置路径』
- 第 151 页的『XML 集合的 DAD 文件』

相关任务:

- 第 58 页的『通过使用 RDB_node 映射来组成 XML 集合』
- 第 100 页的『XML 集合的样式表』
- 第 60 页的『通过使用 RDB_node 映射来分解 XML 集合』
- 第 90 页的『更新、删除和检索 XML 集合中的数据』
- 第 92 页的『搜索 XML 集合』

将 XML 文档分解为 DB2 UDB 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 提供了存储过程来将源 XML 文档中的 XML 数据分解到关系表中。要使用这些存储过程，必须创建一个 DAD 文件，这个 DAD 文件指定 XML 文档与 DB2 UDB 表结构之间的映射。存储过程使用 DAD 文件来分解 XML 文档。

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。必须启用集合的情况有:

- 在将 XML 文档分解到新表中时，必须启用一个 XML 集合，这是因为 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 只为启用集合时创建的表保持多次出现的元素或属性的顺序。在将 XML 文档分解到现有的关系表中时，不能保证会保留该顺序。

有关 enable_collection 选项的信息，请查看关于 dxxadm 管理命令的章节。

当数据库中已经存在表时，如果要传送 DAD 文件，则不需要启用 XML 集合。

在将 XML 文档分解成 DB2 UDB 数据之前:

1. 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。

2. 使用 RDB_node 映射来准备 DAD 文件。
3. 可选: 启用 XML 集合。

过程: :

使用 DB2 UDB XML Extender 提供的两个存储过程之一（dxxShredXML() 或 dxxInsertXML）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: 返回码和返回消息。根据输入 DAD 文件中的 <Xcollection> 规范，它将 XML 文档中的数据插入 XML 集合。然后，存储过程 dxxShredXML() 将分解 XML 文档，并将未标记的 XML 数据插入到 DAD 文件中所指定的表中。 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。

用于组合的相应存储过程是 dxxGenXML(); 它也将 DAD 作为输入参数并且不需要启用 XML 集合。

要使用 dxxShredXML() 来分解 XML 集合

使用以下存储过程声明，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 */
```

示例: 以下示例是对 dxxShredXML() 的调用:

```
#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(100K) xmlDoc; /* input xml document */

              long    returnCode;        /* return 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;

FILE   *file_handle;
long    file_length=0;

/* initialize the DAD CLOB object. */
  file_handle = fopen( "/dxx
/dad/getstart_xcollection.dad", "r" );
if ( file_handle != NULL ) {
  file_length = fread ((void *) &dad.data, 1, FILE_SIZE,
                                  file_handle);
  if (file_length == 0) {
    printf ("Error reading dad file getstart_xcollection.dad\n");
```

```
      rc = -1;
      goto exit;
 } else
    dad.length = file_length;
}
else {
  printf("Error opening dad file  \n");
      rc = -1;
      goto exit;
}

/* Initialize the XML CLOB object. */
  file_handle = fopen( "/dxx
/xml/getstart_xcollection.xml", "r" );
  if ( file_handle != NULL ) {
    file_length = fread ((void *) &xmlDoc.data, 1,
        FILE_SIZE, file_handle);
    if (file_length == 0) {
      printf ("Error reading xml file getstart_xcollection.xml \n");
      rc = -1;
      goto exit;
    } else
        xmlDoc.length = file_length;
  }
  else {
    printf("Error opening xml file \n");
      rc = -1;
      goto exit;
  }

        /* initialize host variable and indicators */
        returnCode = 0;
         msg_txt[0] = '\0';
              dad_ind = 0;
              xmlDoc_ind = 0;
        returnCode_ind = -1;
        returnMsg_ind = -1;

  /* Call the store procedure */
EXEC SQL CALL "DB2XML.DXXSHRED" (:dad:dad_ind;
                                 :xmlDoc:xmlDoc_ind,
                                 :returnCode:returnCode_ind,
                                 :returnMsg:returnMsg_ind);

if (SQLCODE < 0) {
    EXEC SQL ROLLBACK;
  else
    EXEC SQL COMMIT;
}

exit:
  return rc;
```

dxxInsertXML()

此存储过程可用于进行定期更新的应用程序。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 文档中的数据插入已启用的 XML 集合，此集合与 DAD 文件相关联。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。

用于组合的相应存储过程是 dxxRetrieveXML(); 它也采用已启用的 XML 集合的名称。

过程:

要分解 XML 集合: dxxInsertXML():

使用以下存储过程声明，在应用程序中嵌入存储过程调用:

```
dxxInsertXML(char(
             ) collectionName,   /* input */
            CLOB(1M)      xmlobj,         /* input */
            long          returnCode,     /* output */
            varchar(1024) returnMsg)      /* output */
```

示例: 以下是 dxxInsertXML() 调用的示例:

```
#include "dxx.h"
#include "dxxrc.h"

              EXEC SQL INCLUDE SQLCA;
               EXEC SQL BEGIN DECLARE SECTION;
char    collectionName[32];    /* name of an XML collection */
SQL TYPE is CLOB(100K) xmlDoc; /* input xml document */
              long    returnCode;        /* return error code */
              char        returnMsg[1024];    /* error message text */
short   collectionName_ind;
              short       xmlDoc_ind;
              short       returnCode_ind;
              short       returnMsg_ind;
EXEC SQL END DECLARE SECTION;

FILE   *file_handle;
long    file_length=0;

/* initialize the DAD CLOB object. */

file_handle = fopen( "dxxsamples/dad
/getstart_xcollection.dad", "r" );
  if ( file_handle != NULL ) {
  file_length = fread ((void *) &dad.data, 1, FILE_SIZE,
                                    file_handle);
    if (file_length == 0) {
    printf ("Error reading dad file getstart_xcollection.dad\n");
      rc = -1;
      goto exit;
 } else
    dad.length = file_length;
}
else {
  printf("Error opening dad file  \n");
      rc = -1;
      goto exit;
}

        /* initialize host variable and indicators */
strcpy(collectionName, "sales_ord");
        returnCode = 0;
         msg_txt[0] = '\0';
collectionName_ind = 0;
              xmlDoc_ind = 0;
        returnCode_ind = -1;
        returnMsg_ind = -1;

  /* Call the store procedure */
EXEC SQL CALL "db2xml.DXXINSERTXML"
                  (:collection_name:collection_name_ind,
```

```
                                        :xmlDoc:xmlDoc_ind,
                                        :returnCode:returnCode_ind,
                                        :returnMsg:returnMsg_ind);

  if (SQLCODE < 0) {
      EXEC SQL ROLLBACK;
    else
      EXEC SQL COMMIT;
  }

  exit:
    return rc;
```

分解表大小限制

通过抽取元素和属性值并将它们存储在表行中，分解使用 RDB_node 映射来指定将 XML 文档分解为 DB2 UDB 表的方式。每个 XML 文档的值都被存储在一个或多个 DB2 UDB 表中。每个表最多可自每个文档分解 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。

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。当使用多次出现的元素时，认为从单个文档中最多只能将 1024 行分解到一个表中。

相关任务:

- 第 60 页的『通过使用 RDB_node 映射来分解 XML 集合』
- 第 180 页的『调用 XML Extender 组合存储过程』

相关参考:

- 第 191 页的『XML Extender 分解存储过程』
- 第 193 页的『dxxInsertXML() 存储过程』
- 第 191 页的『dxxShredXML() 存储过程』

更新、删除和检索 XML 集合中的数据

可以更新、删除、搜索和检索 XML 集合。但是，使用 XML 集合的目的在于存储或检索数据库表中未标记的纯数据。现有数据库表中的数据与任何入局 XML 文档都没有关系；更新、删除和搜索操作是对这些表进行正常的 SQL 访问。

XML Extender 提供了从 XML 集合视图对数据执行操作的能力。可以使用 UPDATE 和 DELETE SQL 语句来修改用来组成 XML 文档的数据，因此，更新了 XML 集合。对集合表执行 SQL 操作将影响所生成的文档。

限制:

- 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，您可以删除和插入下级表中的行，而不删除该文档。
- 要删除文档，请删除组成 DAD 中指定的顶部 element_node 的行。

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。更新也可以删除多次出现的元素或属性中的数据实例。

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，如果根 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。对集合表执行的 SQL 操作将影响根据那些表生成的文档。

要求：在更新 XML 集合中的数据时，必须遵守下列规则。

- 当现有集合表的主外键之间具有关系时，要指定此关系。如果不具有此关系，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，将连接条件包括在 <SQL_stmt> 元素中。
  - 对于 RDB_node 映射，将连接条件包括在根元素节点的顶部 <condition> 元素中。

更新元素和属性值

在 XML 集合中，元素文本和属性值都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，并包含在 DAD 文件中所指定的连接条件。

示例:

```
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，来更新已组成的 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。如果不删除包含了顶部 element_node 值的行，则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。已组成的文档最初包含两个 <shipment> 元素，但是现在只包含一个元素。

从 XML 集合删除 XML 文档

可以删除从集合中组成的 XML 文档。这意味着如果您具有组成多个 XML 文档的 XML 集合，则可以删除这些已组成的文档中的其中一个文档。对集合表执行 SQL 操作将影响所生成的文档。

过程:

要删除文档，则删除组成顶部 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下级表的外键。仅当在 SQL 中完全指定了主键和外键约束，并且 DAD 中显示的表的关系与那些约束刚好匹配时，使用此方法删除文档才会起作用。

示例:

以下 DELETE 语句指定了主键列的值。

```
DELETE from order_tab
   WHERE order_key=1
```

ORDER_KEY 是 ORDER_TAB 表中的主键，ORDER_TAB 表是 DAD 中指定的顶层表。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成文档。

DAD 文件注意事项: 当在 XML 集合中分解 XML 文档时，可能会丢失多次出现的元素和属性值的顺序，除非在 DAD 文件中指定了该顺序。 要保持此顺序，您应该使用 RDB_node 映射方案。此映射方案使您可对表指定 orderBy 属性，该表包含它的 RDB_node 中的根元素。

搜索 XML 集合

本节描述通过使用搜索条件生成 XML 文档以及搜索分解的 XML 数据来搜索 XML 集合。

使用搜索条件组成 XML 文档

本任务与使用条件进行组合相同。

过程:

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,

(select db2xml.generate_unique()
                as ship_id, date, mode from ship_tab) as s
              WHERE p.price > %d and s.date >'1996-06-01' AND
                o.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);
```

price > ? 将覆盖 order.dad 中的条件 price > 2500.00，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。应用结构化搜索的结果以检索或生成指定的 XML 文档。

XML 集合的映射方案

如果正在使用 XML 集合，则必须选择*映射方案*，它指定 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数据库的关系结构的 XML 文档的分层结构相匹配，所以您应了解这两种结构是如何进行比较的。第 94 页的图 11 显示了层次结构与关系表列的映射关系。

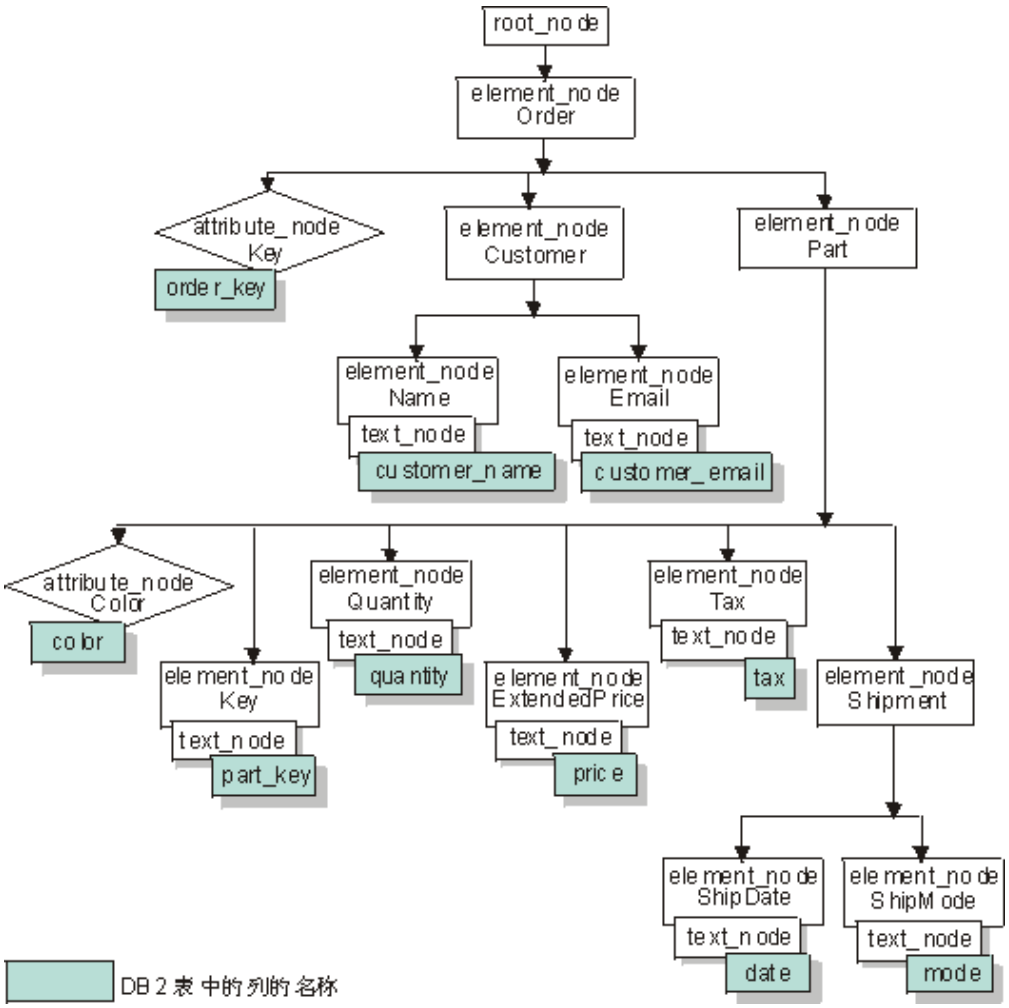


图 11. 映射至关系表列的结构化 XML 文档

XML Extender 在组成或分解位于多个关系表中的 XML 文档时，使用映射方案。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。

映射方案的类型:

使用 <Xcollection> 来在 DAD 文件中指定映射方案。XML Extender 提供两种类型的映射方案: *SQL 映射*和*关系数据库（RDB_node）映射*。

SQL 映射

此方法允许通过单个 SQL 语句直接地从关系数据映射至 XML 文档。SQL 映射只用于组合。<SQL_stmt> 元素的内容必须是有效的 SQL 语句。<SQL_stmt> 元素指定 SELECT 子句中的那些以后要映射至 DAD 中的 XMl 元素或属性的列。当为组成 XML 文档而进行定义时，SQL 语句的 SELECT 子句中的列名用于将 *attribute_node* 的值或 *text_node* 的内容与具有相同 *name_attribute* 的列相关联。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 来映射数据。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，建议在这些表间使用*主键*和*外键*关系。

RDB_node 映射

定义 XML 元素内容或 XML 属性值的位置，以便 XML Extender 可以确定存储或检索 XML 数据的位置。

此方法使用 XML Extender 提供的 *RDB_node*，它包含了一个或多个用于表、可选列和可选条件的节点定义。DAD 中的 <table> 和 <column> 元素用于定义如何将 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。

要定义映射方案，必须使用 <Xcollection> 元素创建 DAD 文件。图 12 显示了带有 XML 集合的 SQL 映射的样本 DAD 文件的一个片段，该文件根据三个关系表中的数据组成一组 XML 文档。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "dxx_install/samples/db2xml/dtd/dad.dtd">
<DAD>
  <dtdid>dxxsamples/dad/getstart.dtd</dtdid>
   <validation>YES</validation>
  <Xcollection>
<SQL_stmt>
       SELECT o.order_key, customer, p.part_key, quantity, price, tax, date,
              ship_id, mode, comment
                FROM order_tab o, part_tab p,
               (select db2xml.generate_unique()
                as ship_id, date, mode, from ship_tab) as
        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
    "
dxxsamples/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 映射方案

XML Extender 提供了若干个用于管理 XML 集合中的数据的存储过程。这些存储过程支持这两种类型的映射。

相关概念:

- 第 151 页的『XML 集合的 DAD 文件』
- 第 96 页的『使用 SQL 映射时的需求』
- 第 97 页的『RDB_Node 映射的需求』

相关任务:

- 第 55 页的『通过使用 SQL 映射来组成 XML 文档』
- 第 58 页的『通过使用 RDB_node 映射来组成 XML 集合』
- 第 60 页的『通过使用 RDB_node 映射来分解 XML 集合』

使用 SQL 映射时的需求

使用 SQL 映射时的需求

在此映射方案中，必须在 DAD <Xcollection> 元素中指定 <SQL_stmt> 元素。<SQL_stmt> 必须包含单一 SQL 语句，该语句可以将多个关系表与查询*谓词*相连接。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。
  - 将同一个表的列分组到一起，并根据表映射至 XML 文档的分层结构时的树层对它们进行排序。每个列分组中的第一个列是对象标识。在 SELECT 子句中，较高级别表的列必须位于较低级别表的列之前。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
    ```

    在本示例中，ORDER_TAB 表中的 order_key 和 customer 列位于最高的关系层，这是因为它们在 XML 文档的分层树中位置较高。SHIP_TAB 表中的 ship_id、date 和 mode 列位于最低的关系层。
  - 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，则查询应使用表表达式和 generate_unique() 用户定义的函数来为该表生成一个这样的键。在以上示例中，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。由于 SHIP_TAB 表没有主键，所以生成 ship_id 来作为主键。将 ship_id 列示为 SHIP_TAB 表组的第一列。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 generate_unique() 用户定义的函数来为没有主单一键的表生成单一键。例如:

    ```
    FROM order_tab as o, part_tab as p,
            (select
             db2xml.generate_unique() as
              ship_id, date, mode, part key from ship_tab) as s
    ```

    在本示例中，使用 generate_unique() 函数来生成单一列候选键，并对其指定名为 ship_id 的别名。

- 当有必要使列相异时，请使用别名名称。例如，可对 ORDER_TAB 表中的列使用 o，对 PART_TAB 表中的列使用 p，并对 SHIP_TAB 表中的列使用 s。

- **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。确保列名后面没有诸如 ASC 或 DESC 之类的内容。
  - 确保各列名与 SELECT 子句中的各列名相匹配。
  - 按对象标识在 SELECT 子句中出现时所具有的相对顺序来列示所有对象标识。
  - 可使用表表达式和 generate_unique() 函数或用户定义的函数来生成标识。
  - 维护实体自顶向下的层次结构。ORDER BY 子句中指定的第一列必须是对每个实体列示的第一列。保持该顺序可确保要生成的 XML 文档不包含不正确的复制。
  - 在 ORDER BY 子句中，不要使用模式或表名来对列进行限定。

<SQL_stmt> 元素的功能十分强大，其原因在于只要谓词中的表达式使用表中的列，就可以在 WHERE 子句中指定任何谓词。

相关参考:

- 第 221 页的附录 A, 『样本』

RDB_Node 映射的需求

当使用 RDB_Node 来作为映射方法时，不要在 DAD 文件的 <Xcollection> 元素中使用 <SQL_stmt> 元素。而是，应该在元素节点的每个顶部节点中使用 <RDB_node> 元素，并将该元素用于每个属性节点和文本节点。

- **顶部 element_node 的 RDB_node**

  DAD 文件中的顶部 element_node 表示 XML 文档的根元素。如下所述指定顶部 element_node 的 RDB_node:
  - 指定所有与 XML 集合相关联的表。例如，以下映射指定元素节点 <Order>（它是顶层元素节点）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>
    ```

如果集合中只有一个表，则条件元素可为空或丢失。

– 条件元素可引用列名无限次。

– 如果分解或启用 DAD 文件指定的 XML 集合，则必须对每个表指定主键。主键可由单个列组成，也可以由多个列组成（此时称为组合键）。通过对 RDB_node 的 table 元素添加*属性键*来指定主键。当提供了组合键时，*key* 属性由通过空格分隔的键列的名称指定。例如:

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

如果将同一个 DAD 用于组合，则忽略对分解指定的信息。

– 使用 orderBy 属性来重新组成 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顺序的键的列名。orderBy 属性是 DAD 文件中的表元素的一部分，且它是可选属性。

在 <table> 标记中拼写出表名和列名。

- **每个 attribute_node 和 text_node 的 RDB_node**

  XML Extender 需要知道从数据库中的何处检索数据。XML Extender 还需要知道将 XML 文档的内容放到数据库中的何处。必须对每个属性节点和文本节点指定 RDB_node。还必须指定表名和列名，条件值是可选的。

  1. 指定包含列数据的表的名称。表名必须包括在顶部 element_node 的 RDB_node 中。在本示例中，对于元素 <Price> 的 text_node，将表指定为 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>
```

  2. 指定包含元素文本的数据的列的名称。在上一个示例中，将列指定为 PRICE。
  3. 如果要使用某个查询条件来生成 XML 文档，请指定该条件。只有满足条件的数据才会存在于生成的 XML 文档中。该条件必须是有效的 WHERE 子句。在以上示例中，将条件指定为 `price > 2500.00`，因此只有价格高于 2500 的行才会包括在 XML 文档中。
  4. 如果正在分解文档，或正在启用 DAD 文件指定的 XML 集合，则必须对每个属性节点和文本节点指定列类型。通过对每个属性节点和文本节点指定列类型，确保在启用 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"/>
```

  在进行组合时，将忽略分解文档时指定的列类型。

- 维护实体自顶向下的层次结构。确保正确嵌套元素节点，以便 XML Extender 在组成或分解文档时了解元素之间的关系。例如，以下 DAD 文件没有将 Shipment 嵌套在 Part 中:

```
<element_node name="Part">
    ...
    <element_node name="ExtendedPrice">
          ...
    </element_node>
```

```
            ...
    </element_node> <!-- end of element Part -->

   <element_node name="Shipment" multi_occurrence="YES">
            <element_node name="ShipDate">
                ...
            </element_node>
            <element_node name="ShipMode">
                ...
            </element_node>

    </element_node> <!-- end of element Shipment-->
```

此 DAD 文件将生成一个 XML 文档，在该文档中，Part 和 Shipment 元素是兄弟节点。

```
<Part color="black ">
  <key>68</key>
  <Quantity>36</Quantity>
  <ExtendedPrice>34850.16</ExtendedPrice>
  <Tax>6.000000e-2</Tax>
</Part>

<Shipment>
    <ShipDate>1998-08-19</ShipDate>
    <ShipMode>BOAT  </ShipMode>
</Shipment>
```

以下代码显示了 DAD 文件中嵌套在 Part 元素内的 Shipment 元素。

```
 <element_node name="Part">
     ...
     <element_node name="ExtendedPrice">
           ...
     </element_node>
     ...
    <element_node name="Shipment" multi_occurrence="YES">
          <element_node name="ShipDate">
              ...
          </element_node>
          <element_node name="ShipMode">
              ...
          </element_node>

     </element_node> <!-- end of element Shipment-->
 </element_node> <!-- end of element Part -->
```

通过将 Shipment 元素嵌套在 Part 元素内，将生成把 Shipment 作为 Part 元素的子元素的 XML 文件:

```
<Part color="black ">
  <key>68</key>
  <Quantity>36</Quantity>
  <ExtendedPrice>34850.16</ExtendedPrice>
  <Tax>6.000000e-2</Tax>
  <Shipment>
    <ShipDate>1998-08-19</ShipDate>
    <ShipMode>BOAT  </ShipMode>
  </Shipment>
</Part>
```

对根节点条件的谓词没有定序限制。

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 元素可能会更困难。

对于具有映射至相同表的 element_nodes 和 attribute_nodes 的 DAD 子树，以下条件成立:

- 属性节点不必是映射至相同表的元素节点的最低层共同祖先的第一个子代。
- 只要属性节点不涉及连接条件，它们就可以出现在子树的任何位置。

限制: RDB_node 映射 DAD 中允许的表的数量限制为 30 个。每个表中允许的列数为 500。另外，在条件语句的连接谓词中可指定的每个表或列的次数不受限制

XML 集合的样式表

当组成文档时，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>
```

位置路径

*位置路径*定义 XML 文档结构中 XML 元素或属性的位置。XML Extender 使用位置路径来:

- 在使用抽取 UDF 时（如 dxxRetrieveXML），定位要抽取的元素和属性
- 在 XML 列的 DAD 中定义索引方案时，指定 XML 元素或属性与 DB2® 列之间的映射
- 用于使用 Text Extender 进行的结构化文本搜索
- 覆盖存储过程中的 XML 集合 DAD 文件值。

第 101 页的图 13 显示了位置路径的示例以及它与 XML 文档结构之间的关系。

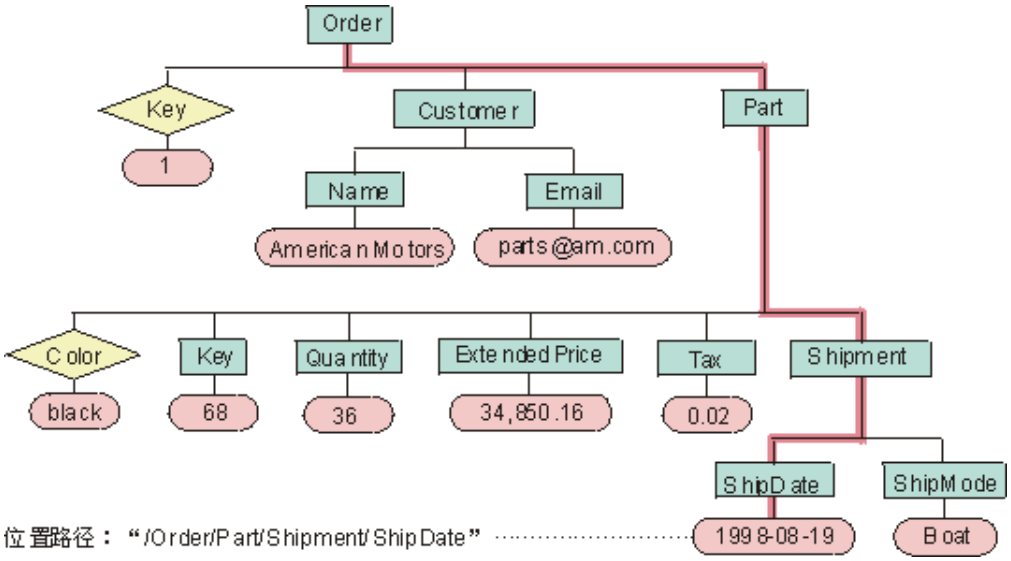


图 13. 将文档作为结构化 XML 文档存储在 DB2 UDB 表列中

相关参考:

- 第 101 页的『位置路径语法』

位置路径语法

XML Extender 使用位置路径来导航 XML 文档结构。以下列表描述了 XML Extender 支持的位置路径语法。单斜杠（/）路径指示上下文是整个文档。

1. / 表示 XML 根元素。这是包含文档中的所有其他元素的元素。
2. */tag1*
   表示根元素下面的元素 *tag1*。
3. */tag1/tag2/..../tagn*
   表示名为 *tagn* 的元素，它是从根、*tag1*、*tag2* 直到 *tagn*-1 的降序链的子代。
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* 的元素的后代，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* 的子代。
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 *tag1* 元素的子代。
8. */tag1/tag2[@attr1="5"]/.../tagn*
   表示名为 *tagn* 的元素，它是从根、*tag1*、*tag2* 至 *tagn*-1 的降序链的子代，其中 *tag2* 的属性 *attr1* 的值为 5。

简单位置路径

*简单位置路径*是在 XML 列 DAD 文件中使用的一类位置路径。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。每个属性的值是在元素类型后用方括号括起来的。表 17 总结了简单位置路径的语法。

表 17. 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* 的元素及其父代标识的元素内容
XML 属性	*/tag_1/tag_2/..../tag_n-1/tag_n/@attr1*	由 *tagn* 及其父代标识的元素的名为 *attr1* 的属性

位置路径用法

位置路径的语法取决于您访问元素或属性的位置时所处的上下文。因为 XML Extender 使用元素或属性与 DB2 列之间的一对一映射，所以它限制 DAD 文件和函数的语法规则。表 18 描述了使用语法选项时所处的上下文 。

表 18. 使用位置路径时 XML Extender 的限制

位置路径的使用	支持的位置路径
副表的 XML 列 DAD 映射中的路径属性值	3，6（表 17 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	3 - 例外：未使用斜杠指定根标记。例如：`tag1/tag2/..../tagn`

[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。

相关概念:

- 第 100 页的『位置路径』

启用 XML 集合

启用 XML 集合解析 DAD 文件，以标识与 XML 文档相关的表和列，并在 XML_USAGE 表中记录控制信息。启用 XML 集合对于以下情况是可选的:

- 分解 XML 文档并将数据存储在新的 DB2 UDB 表中
- 将多个 DB2 UDB 表中现有的数据组成 XML 文档

如果对相同的 DAD 文件进行组成和分解，则对组成和分解都可以启用集合。

可以使用 XML Extender “管理”向导、带 enable_collection 选项的 **dxxadm** 命令或者 XML Extender 存储过程 dxxEnableCollection() 启用 XML集合。

使用“管理”向导:

要使用此向导来启用 XML 集合:

1. 设置并启动“管理”向导。
2. 从“启动板”窗口单击**使用 XML 集合**。 “选择任务”窗口打开。
3. 单击**启用集合**，然后单击**下一步**。 “启用集合”窗口打开。
4. 在**集合名**字段中，选择想要启用的集合的名称。

5. 在 **DAD 文件名**字段中指定 DAD 文件名。
6. 可选: 在**表空间**字段中输入以前创建的表空间的名称。

   此表空间将包含为分解而生成的新的 DB2 UDB 表。
7. 单击**完成**来启用集合，并返回至“启动板”窗口。
   - 如果成功启用了该集合，则会显示启用集合成功消息。
   - 如果未能成功启用集合，则会显示一则错误消息。重复上述步骤，直到成功启用该集合为止。

使用 dxxadm 命令启用集合:

要启用 XML 集合，从 DB2 UDB 命令行输入 **dxxadm** 命令:

语法:

参数:

dbName
RDB 数据库的名称。

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
包含为分解生成的新的 DB2 UDB 表的现有表空间。如果未指定此项，则使用缺省表空间。

示例: 以下示例使用命令行在数据库 SALES_DB 中启用名为 sales_ord 的集合。DAD 文件使用 SQL 映射。

从 Qshell:

```
dxxadm enable_collection SALES_DB sales_ord getstart_collection.dad
```

从 OS 命令行:

```
CALL QDBXM/QZXMADM PARM(enable_collection SALES_DB sales_ord
  'getstart_collection.dad')
```

从“操作导航器”:

```
CALL MYSCHEMA.QZXMADM('enable_collection', 'SALES_DB', 'sales_ord',
  'getstart_collection.dad');
```

启用 XML 集合之后，可以使用 XML Extender 存储过程来组成或分解 XML 文档。

相关概念:

- 第 81 页的『XML 集合作为存储和访问方法』

相关任务:

- 第 104 页的『禁用 XML 集合』
- 第 81 页的『管理 XML 集合中的数据』

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想要删除集合时，应禁用集合。

可通过使用 XML Extender “管理”向导、使用带 disable_collection 选项的 **dxxadm** 命令，或者使用 XML Extender 存储过程 dxxDisableCollection() 禁用 XML 集合。

过程:

要使用“管理”向导来禁用 XML 集合:

1. 启动“管理”向导。
2. 从“启动板”窗口单击**使用 XML 集合**，以查看与 XML Extender 集合相关的任务。“选择任务”窗口打开。
3. 单击**禁用 XML 集合**，然后单击**下一步**以禁用 XML 集合。“禁用集合”窗口打开。
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“启动板”窗口。
   - 如果成功禁用了该集合，则会显示禁用集合成功消息。
   - 如果未成功禁用该集合，会显示错误消息框。重复上述步骤，直到成功禁用该集合为止。

要从命令行禁用 XML 集合，输入 **dxxadm** 命令:

语法:

参数:

dbName
RDB 数据库的名称。

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

示例:

从 Qshell:

```
dxxadm disable_collection SALES_DB sales_ord
```

从 OS 命令行:

```
CALL QDBXM/QZXMADM PARM(disable_collection SALES_DB sales_ord)
```

从“操作导航器”:

```
CALL MYSCHEMA.QZXMADM('disable_collection', 'SALES_DB', 'sales_ord');
```

相关概念:

- 第 81 页的『XML 集合作为存储和访问方法』

相关任务:

- 第 81 页的『管理 XML 集合中的数据』

相关参考:

- 第 175 页的『XML Extender 管理存储过程』

第 5 章 XML 模式

XML 模式可用于代替 DTD 来定义 XML 文档内容的规范。XML 模式使用 XML 格式或 SML 语法来定义 XML 文档的元素和属性名称，以及定义允许包含元素和属性的内容的类型。

使用 XML 模式取代 DTD 的优点

与 XML 模式相比，DTD 更易于编码和验证。然而，使用 XML 模式的优点如下表所示:

- XML 模式是有效 XML 文档，WebSphere® Studio Application Developer 中的 XSD Editor、XML Spy 或 XML Authority 之类的工具可以处理的此类 XML 文档。
- XML 模式的功能比 DTD 更强大。凡是 DTD 可以定义的，模式也可以定义，但反过来就不是这样。
- XML 模式支持一组类似于最常用的编程语言中所使用的数据类型，并提供创建其他类型的能力。 可以将文档内容限制为具有适当的类型。例如，可以复制 DB2 中的字段的属性。
- XML 模式支持正则表达式以对字符数据设置约束，如果使用 DTD 则不可能这样做。
- XML 模式提供了更好的对 XML 名称空间的支持，XML 名称空间使您能够验证使用多个名称空间的文档并重新使用其他名称空间中已定义模式的构造。
- XML 模式通过包括和导入元素提供了更好的对模块性和重新使用的支持。
- XML 模式支持元素、属性和数据类型定义的继承。

相关任务:

- 第 108 页的『模式中的数据类型、元素和属性』

相关参考:

- 第 109 页的『XML 模式的示例』

XML Extender 的 UDT 和 UDF 名称

DB2® 函数的全名是 *schema-name.function-name*，其中 *schema-name* 是一个标识，它提供了对一组 SQL 对象的逻辑分组。XML Extender UDF 和 UDT 的模式名是 DB2XML。在本文档中，仅对函数名进行引用。

如果将模式名添加至函数路径，则可以指定不带模式名的 UDT 和 UDF。函数路径是模式名的有序列表。DB2 UDB 使用列表中模式名的顺序来解析对函数和 UDT 的引用。可通过指定 SQL 语句 SET CURRENT FUNCTION PATH 来指定函数路径。此语句将在 CURRENT FUNCTION PATH 专用寄存器中设置函数路径。

XML 模式 complexType 元素

使用 XML 模式元素 complexType 来定义可以由子元素组成的元素类型。例如，下列标记显示了一个地址在 XML 文档中的设计。

```
<billTo country="US">
    <name>Dan Jones</name>
    <street>My Street</street>
    <city>My Town</city>
    <state>CA</state>
    <zip>99999</zip>
</billTo>
```

此元素的结构可在 XML 模式中作如下定义:

```
1 <xsd:element name="billTo" type="USAddress"/>
2 < xsd:complexType name="USAddress">
 3  <xsd:sequence>
 4   < xsd:element name="name" type="xsd:string"/>
 5   < xsd:element name="street" type="xsd:string"/>
 6   < xsd:element name="city" type="xsd:string"/>
 7   < xsd:element name="state" type="xsd:string"/>
 8   < xsd:element name="zip" type="xsd:decimal"/>
 9  </xsd:sequence>
 10  < xsd:attribute name="country" type="xsd:NMTOKEN" use="fixed"  value="US"/>
 12</xsd:complexType>
```

在上述示例中，假定 xsd 前缀已绑定至 XML 模式名称空间。行 2 至 5 将 complexType USAddress 定义为五个元素和一个属性的序列。元素在 sequence 标记中出现的顺序决定了元素的排列顺序。

内部元素来自数据类型 xsd:string 或 xsd:decimal。这两者都是预定义的简单数据类型。

另外，可以使用 all 标记或 choice 标记代替 sequence 标记。对于 all 标记，所有子元素都必须出现，但无需以任何特定顺序出现。使用 choice 标记，XML 文档中必须恰好出现一个子元素。

还可以使用用户定义数据类型来定义其他元素。

模式中的数据类型、元素和属性

XML 模式中的简单数据类型

XML 模式提供了一组简单内置数据类型。通过应用约束，可以从中导出其他数据类型。

在“示例 1”中，将基本类型 xsd:positiveInteger 的范围限制为 0 到 100。

示例 1

```
< xsd:element name="quantity">
   < xsd:simpleType>
    < xsd:restriction base="xsd:positiveInteger">
     < xsd:maxExclusive value="100"/>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:element>
```

在“示例 2”中，通过正则表达式限制基本类型 xsd:string。

示例 2

```
<xsd:simpleType name="SKU">
   < xsd:restriction base="xsd:string">
    < xsd:pattern value="\d{3}-[A-Z]{2}"/>
   </xsd:restriction>
  </xsd:simpleType>
```

示例 3 基于字符串内置类型来显示列举类型。

示例 3

```
<xsd:simpleType name="SchoolClass">
   < xsd:restriction base="xsd:string">
    < xsd:enumeration value="WI"/>
    < xsd:enumeration value="MI"/>
    < xsd:enumeration value="II"/>
    < xsd:enumeration value="DI"/>
    < xsd:enumeration value="AI"/>
   </xsd:restriction>
  </xsd:simpleType>
```

XML 模式中元素

要声明 XML 模式中的元素，必须将名称和类型指示为元素的属性。例如:

```
<xsd:element name="street" type="xsd:string"/>
```

此外，可以使用属性 minOccurs 和 maxOccurs 来确定元素在 XML 文档中必须出现的最大次数或最小次数。minOccurs 和 maxOccurs 的缺省值是 1。

XML 模式中的属性

属性声明出现在元素定义的末尾。例如:

```
<xsd:complexType name="PurchaseOrderType">
< xsd:sequence>
  < xsd:element name="billTo" type="USAddress"/>
< xsd:sequence>
  < xsd:attribute name="orderDate" type="xsd:date"/>
 </xsd:complexType>
```

相关概念:

- 第 107 页的『使用 XML 模式取代 DTD 的优点』

相关任务:

- 第 146 页的『验证函数』

相关参考:

- 第 109 页的『XML 模式的示例』
- 第 108 页的『XML 模式 complexType 元素』

XML 模式的示例

编写 XML 模式的良好策略是首先使用 UML 工具来设计 XML 文档的数据结构。设计结构之后，可以将此结构映射到模式文档中。以下示例显示一个 XML 模式。

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>
3
4   <xs:element name="personnel">
5    <xs:complexType>
6     <xs:sequence>
```

```
 7       <xs:element ref="person" minOccurs='1' maxOccurs='unbounded'/>
 8    </xs:sequence>
 9   </xs:complexType>
10  </xs:element>
11
12   <xs:element name="person">
13    <xs:complexType>
14     <xs:sequence>
15       <xs:element ref="name"/>
16       <xs:element ref="email" minOccurs='0' maxOccurs='4'/>
17    </xs:sequence>
18     <xs:attribute name="id"  type="xs:ID" use='required'/>
19   </xs:complexType>
20  </xs:element>
21
22   <xs:element name="name">
23    <xs:complexType>
24     <xs:sequence>
25      <xs:element ref="family"/>
26      <xs:element ref="given"/>
27    </xs:sequence>
28   </xs:complexType>
29  </xs:element>
30
31   <xs:element name="family" type='xs:string'/>
32   <xs:element name="given" type='xs:string'/>
33   <xs:element name="email" type='xs:string'/>
34 </xs:schema>
```

前两行声明此 XML 模式与 XML 1.0 兼容且使用 Unicode 8 进行解码，并指定使用 XML 模式标准名称空间，此标准名称空间使您能够访问基本 XML 模式数据类型和结构。

行 4 至 10 将 personnel 定义为由 1 至 n 个人员序列组成的 complexType。然后在行 12 至 20 中定义 complexType。它由 complexType 元素名称和元素电子邮件组成。电子邮件元素是可选的（minOcccurs = '0'），最多可以出现四次（maxOccurs = '4'）。元素出现的次数越多，验证模式所需的时间就越长。作为对比，在 DTD 中只能选择让元素出现 0 次、1 次或无限次。

行 22 至 29 定义用于 person 类型的 name 类型。name 类型由一系列 family 和 given 元素组成。

行 31 至 33 定义单一元素 *family*、*given* 和 *e-mail*，它们包含已声明的类型字符串。

使用模式的 XML 文档实例

以下示例是一个 XML 文档，它是 *personalnr.xsd* 模式的实例。

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <personnel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:noNamespaceSchemaLocation='personsnr.xsd'>
4
5   <person id="Big.Boss" >
6      <name><family>Boss</family> <given>Big</given></name>
7     <email>chief@foo.com</email>
8   </person>
9
10   <person id="one.worker">
11   <name><family>Worker</family><given>One</given></name>
12      <email>one@foo.com</email>
13   </person>
14
```

```
15    <person id="two.worker">
16   <name><family>Worker</family><given>Two</given></name>
17      <email>two@foo.com</email>
18   </person>
19 </personnel>
```

使用 DTD 的 XML 文档实例

此示例显示如何将这个 XML 模式识别为 DTD。

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <!ELEMENT email (#PCDATA)>
3   <!ELEMENT family (#PCDATA)>
4   <!ELEMENT given (#PCDATA)>
5   <!ELEMENT name (family, given)>
6   <!ELEMENT person (name, email*)>
7
8   <!ATTLIST person
9 id ID #REQUIRED>
10 <!ELEMENT personnel (person+)>
```

通过使用 DTD，您可以将电子邮件的最大出现次数设置为只出现一次或不限制出现次数。

使用此 DTD，XML 文档实例将与顶上的示例相同，除第 2 行更改为以下情况外:

```
<!DOCTYPE personnel SYSTEM "personsnr.dtd">
```

相关概念:

- 第 107 页的『使用 XML 模式取代 DTD 的优点』

相关任务:

- 第 108 页的『模式中的数据类型、元素和属性』
- 第 146 页的『验证函数』

相关参考:

- 第 108 页的『XML 模式 complexType 元素』

第 6 章 dxxadm 管理命令

dxxadm 命令概述

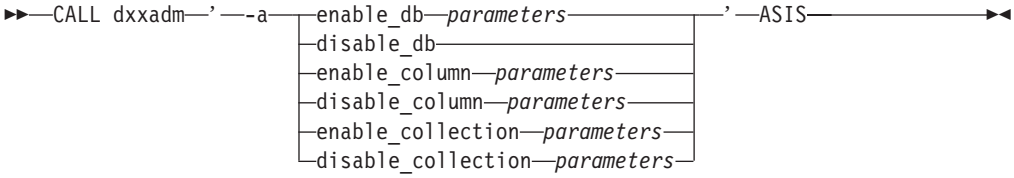
XML Extender 提供了管理命令 **dxxadm**，用于从 Q-shell 或 OS 命令行完成下列管理任务。

- 为 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

相关概念:

- 第 29 页的『XML Extender 的管理工具』
- 第 36 页的『XML Extender 管理计划』

dxxadm 管理命令的语法



参数:

表 19. dxxadm 参数

参数	描述
enable_db	为数据库启用 XML Extender 功能。
disable_db	为数据库禁用 XML Extender 功能。
enable_column	启用 XML 列以便可以将 XML 文档存储在列中。
disable_column	禁用已经启用了 XML 的列。
enable_collection	按指定的 DAD 启用 XML 集合。
enable_collection	禁用启用了 XML 的集合。

此调用假定已激活 XML Extender 装入模块库。如果尚未这样做的话，对 **dxxadm** 使用全限定名称。

用于管理命令的的子命令

系统程序员可使用下列 **dxxadm** ：

- enable_column
- enable_collection
- enable_db
- disable_column

- disable_collection
- disable_db

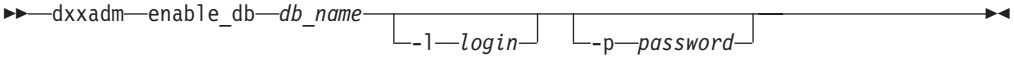
dxxadm 命令的 enable_db 选项

用途:

对数据库启用 XML Extender 功能。启用数据库时，XML Extender 将创建下列对象:

- XML Extender 用户定义的类型（UDT）。
- XML Extender 用户定义的函数（UDF）。
- XML Extender DTD 资源库表 DTD_REF，用来存储 DTD 和有关每个 DTD 的信息。
- XML Extender 用法表 XML_USAGE，它用来存储对 XML 启用的每个列以及每个集合的公共信息。

语法:



参数:

表 20. enable_db 参数

参数	描述
db_name	XML 数据驻留所在的 RDB 数据库的名称。
-l *login*	用来连接数据库的可选用户标识。如果未指定此项，则使用当前用户标识。
-p *password*	用来连接数据库的可选密码。如果未指定此项，则使用当前密码。

如果正在使用已分区的 “DB2 UDB 企业服务器版” 并且要在启用数据库时指定表空间，则需要在创建表空间时指定节点组。例如:

```
db2 "create database partition group mygroup on node (0,1)"
db2 "create regular tablespace mytb in database partition group mygroup
   managed by system using ('mytb')"
```

在上述示例中，启用数据库时将指定 mytb 表空间。

如果启用数据库时未提供表空间选项，XML Extender 将检查 DXXDTDRF 和 DXXXMLUS 表空间是否存在。如果此表空间存在，则会在 DXXDTDRF 表空间中创建 db2xml.dtd_ref 表，在 DXXXMLUS 表空间中创建 db2xml.xml_usage 表。如果 DXXDTDRF 或 DXXXMLUS 表之一不存在，则在大多数相应表空间分别创建这两个表（db2xml.dtd_ref 或 db2xml.xml_usage）。

如果启用数据库时仅提供一个 DXXDTDRF 表空间，则在指定的表空间中创建这两个表。如果启用数据库时提供两个表空间，将在列出的第一表空间中创建 db2xml.dtd_ref 表，在列出的第二个表空间中创建 db2xml.xml_usage 表。

示例: :

以下示例启用数据库 SALES_DB。

从 Qshell:

```
dxxadm enable_db SALES_DB
```

从 OS 命令行:

```
CALL QDBXM/QZXMADM PARM(enable_db SALES_DB)
```

从 “iSeries 导航器” :

```
CALL MYSCHEMA.QZXMADM('enable_db', 'SALES_DB');
```

相关参考:

- 第 113 页的『dxxadm 命令概述』

dxxadm 命令的 disable_db 选项

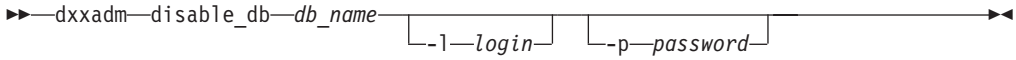
用途:

对数据库服务器禁用 XML Extender 功能; 此操作称为 “禁用数据库”。禁用数据库后, XML Extender 将不再能使用它。当 XML Extender 禁用数据库时, 它将删除下列对象:

- XML Extender 用户定义的类型（UDT）。
- XML Extender 用户定义的函数（UDF）。
- XML Extender DTD 资源库表 DTD_REF, 它用来存储 DTD 和有关每个 DTD 的信息。
- XML Extender 用法表 XML_USAGE, 它用来存储对 XML 启用的每个列以及每个集合的公共信息。

重要事项: 在尝试禁用数据库之前, 必须禁用所有 XML 列。XML Extender 不能禁用其中包含已对 XML 启用的列或集合的数据库。您还必须删除所有带有使用 XML Extender 用户定义的类型（如 XMLCLOB）定义的列的表。

语法:



参数:

表 21. disable_db 参数

参数	描述
db_name	XML 数据驻留所在的 RDB 数据库的名称
-l *login*	用来连接数据库的用户标识。如果未指定此项, 则使用当前用户标识。
-p *password*	用来连接数据库的密码。如果未指定此项, 则使用当前密码。

示例: :

以下示例禁用数据库 SALES_DB。

从 Qshell:

```
dxxadm disable_db SALES_DB
```

从 OS 命令行:

```
CALL QDBXM/QZXMADM PARM(disable_db SALES_DB)
```

从 “iSeries 导航器” :

```
CALL MYSCHEMA.QZXMADM('disable_db', 'SALES_DB');
```

相关概念:

- 第 197 页的第 11 章, 『XML Extender 管理支持表』

相关参考:

- 第 175 页的『XML Extender 管理存储过程』
- 第 xi 页的『如何阅读语法图』

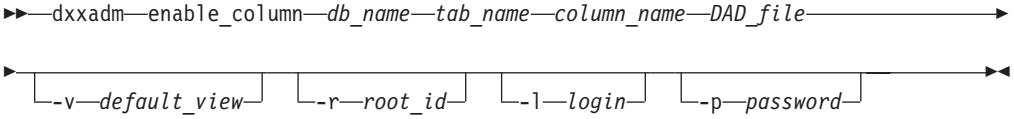
dxxadm 命令的 enable_column 选项

用途:

连接至数据库并启用 XML 列, 以使其可包含 XML Extender UDT。当启用列时, XML Extender 完成下列任务:

- 确定 XML 表是否具有主键; 如果没有, 则 XML Extender 将变更 XML 表, 并添加名为 DXXROOT_ID 的列。
- 创建 DAD 文件中指定的副表, 并且有一列包含 XML 表中每一行的唯一标识。此列可以是用户指定的根标识, 也可以是由 XML Extender 命名的 DXXROOT_ID。
- 有选择地为 XML 表及其副表创建缺省视图, 并可选择使用您指定的名称。

语法:



参数:

表 22. enable_column 参数

参数	描述
db_name	XML 数据驻留所在的 RDB 数据库的名称。
tab_name	XML 列驻留所在的表的名称。
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 列和副表。
-v *default_view*	用于连接 XML 列和副表的缺省视图的名称。
-r *root_id*	XML 列表中的主键的名称, 它要用作副表的 root_id。root_id 是可选的。
-l *login*	用来连接数据库的用户标识。如果未指定此项, 则使用当前用户标识。

表 22. enable_column 参数 (续)

参数	描述
-p *password*	用来连接数据库的密码。如果未指定此项，则使用当前密码。

如果正在使用已分区的 “DB2 UDB 企业服务器版”，并且要在启用列时指定表空间，则需要在创建表空间时指定节点组。例如:

```
db2 "create database partition group mygroup on node (0,1)"
db2 "create regular tablespace mytb in database partition group mygroup
   managed by system using ('mytb')"
```

在上述示例中，启用列时将指定 mytb 表空间:

```
dxxadm enable_column mydatabase mytable mycolumn "dad/mydad.dad" -t mytb
```

示例: :

以下示例启用一个 XML 列。

从 Qshell:

```
dxxadm enable_column SALES_DB MYSCHEMA.SALES_TAB ORDER getstart.dad
   -v sales_order_view -r INVOICE_NUMBER
```

从 OS 命令行:

```
CALL QDBXM/QZXMADM PARM(enable_column SALES_DB 'MYSCHEMA.SALES_TAB'
   ORDER 'getstart.dad' '-v' sales_order_view '-r' INVOICE_NUMBER)
```

从 “iSeries 导航器”:

```
CALL MYSCHEMA.QZXMADM('enable_column', 'SALES_DB', 'MYSCHEMA.SALES_TAB',
   'ORDER', 'getstart.dad', '-v sales_order_view', '-r INVOICE_NUMBER');
```

相关样本:

- 『dxx_xml – s-getstart_enableCol_NT-cmd.htm』
- 『dxx_xml – s-getstart_enableCol-cmd.htm』

dxxadm 命令的 disable_column 选项

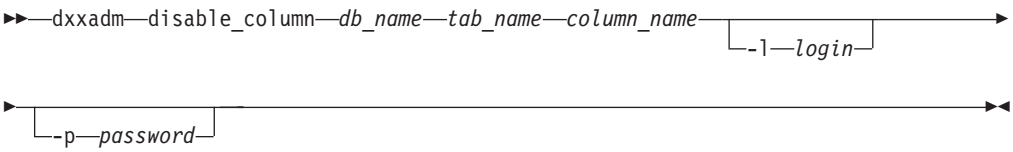
用途:

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副表。

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副表及 XML_USAGE 表中的 XML 列项。

语法:



参数:

表 23. disable_column 参数

参数	描述
db_name	数据驻留所在的 RDB 数据库的名称。
tab_name	XML 列驻留所在的表的名称。
column_name	XML 列的名称。
-l *login*	用来连接数据库的用户标识。如果未指定此项，则使用当前用户标识。
-p *password*	用来连接数据库的密码。如果未指定此项，则使用当前密码。

示例:

以下示例禁用一个启用了 XML 的列。

从 Qshell:

```
dxxadm disable_column SALES_DB MYSCHEMA.SALES_TAB ORDER
```

从 OS 命令行:

```
CALL QDBXM/QZXMADM PARM(disable_column SALES_DB 'MYSCHEMA.SALES_TAB' ORDER)
```

从“iSeries 导航器”:

```
CALL MYSCHEMA.QZXMADM('disable_column', 'SALES_DB', 'MYSCHEMA.SALES_TAB', 'ORDER');
```

相关参考:

- 第 118 页的『dxxadm 命令的 enable_collection 选项』

相关样本:

- 『dxx_xml – s-getstart_clean_NT-cmd.htm』
- 『dxx_xml – s-getstart_clean-cmd.htm』

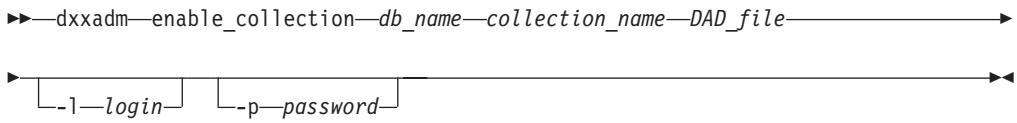
dxxadm 命令的 enable_collection 选项

用途:

连接数据库并按指定的 DAD 启用 XML 集合。当已分区的“企业服务器版”环境中运行 XML Extender 时，检查 DAD 文件中指定的所有表是否包含至少一个符合分区键的列。 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。

语法:



参数:

表 24. enable_collection 参数

参数	描述
db_name	数据驻留所在的 RDB 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。
-l *login*	用来连接数据库的用户标识。如果未指定此项，则使用当前用户标识。
-p *password*	用来连接数据库的密码。如果未指定此项，则使用当前密码。

如果正在使用已分区的 “DB2 UDB 企业服务器版” 并且要在启用集合时指定表空间，则需要在创建表空间时指定节点组。例如:

```
db2 "create database partition group mygroup on node (0,1)"
db2 "create regular tablespace mytb in database partition group mygroup
   managed by system using ('mytb')"
```

在上述示例中，启用集合时将指定 mytb 表空间。

示例:

以下示例启用一个 XML 集合。

从 Qshell:

```
dxxadm enable_collection SALES_DB sales_ord getstart_xcollection.dad
```

从 OS 命令行:

```
CALL QDBXM/QZXMADM PARM(enable_collection SALES_DB sales_ord
  'getstart_collection.dad')
```

从 “iSeries 导航器” :

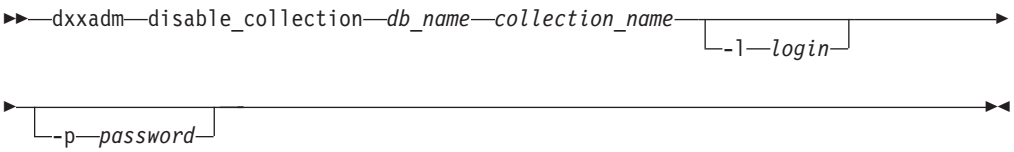
```
CALL MYSCHEMA.QZXMADM('enable_collection', 'SALES_DB', 'sales_ord',
  'getstart_collection.dad');
```

dxxadm 命令的 disable_collection 选项

用途:

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。禁用集合不会删除在使用 enable_collection 选项时创建的集合表。

语法:



参数:

表 25. disable_collection 参数

参数	描述
db_name	数据驻留所在的 RDB 数据库的名称。
collection_name	XML 集合的名称。
-l *login*	用来连接数据库的用户标识。如果未指定此项，则使用当前用户标识。
-p *password*	用来连接数据库的密码。如果未指定此项，则使用当前密码。

示例: :

下列示例禁用一个 XML 集合。

从 Qshell:

```
dxxadm disable_collection SALES_DB sales_ord
```

从 OS 命令行:

```
CALL QDBXM/QZXMADM PARM(disable_collection SALES_DB sales_ord)
```

从 “iSeries 导航器” :

```
CALL MYSCHEMA.QZXMADM('disable_collection', 'SALES_DB', 'sales_ord');
```

第 4 部分 参考

此部分提供 XML Extender 管理命令、用户定义的数据类型（UDT）、用户定义的函数（UDF）和存储过程的语法信息。还提供了消息文本，以便于确定问题。

第 7 章 XML Extender 用户定义的类型

数据类型用于定义应用程序表中将用来存储 XML 文档的列。您还可以通过指定文件名来将 XML 文档作为文件存储在文件系统上。

所有 XML Extender 的用户定义的类型都有限定符 **DB2XML**，它是 DB2 UDB XML Extender 用户定义的类型的*模式名*。例如:

```
db2xml.XMLVarchar
```

XML Extender 创建 UDT 以用于存储和检索 XML 文档。表 26 描述了 UDT。

表 26. XML Extender UDT

用户定义的类型列	源数据类型	用法描述
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。您必须确保文件内容、DB2 和为建立索引而创建的副表之间的完整性。

其中 *varchar_len* 和 *clob_len* 是特定于操作系统的。

对于 iSeries 上的 XML Extender，*varchar_len* = 3K 且 *clob_len* = 10M。

要更改 XMLVARCHAR 或 XMLCLOB UDT 的大小，请在创建 UDT 之后为 XML Extender 启用数据库。

过程:

要更改已启用数据库的 XMLVARCHAR 或 XMLCLOB UDT 的大小:

1. 备份启用 XML Extender 的数据库中的所有数据。
2. 删除所有 XML 集合表或 XML 列副表。
3. 使用 命令禁用数据库。
4. 创建 XMLVARCHAR 或 XMLCLOB 用户定义的类型。
5. 使用 命令启用数据库。
6. 重新创建并重新装入表。

这些 UDT 仅用于指定应用程序列的类型；它们不应用于 XML Extender 创建的副表。

相关概念:

- 第 67 页的『XML 列作为存储和访问方法』
- 第 81 页的『XML 集合作为存储和访问方法』
- 第 30 页的『准备管理 DB2 XML Extender』
- 第 93 页的『XML 集合的映射方案』

第 8 章 XML Extender 用户定义的函数

用户定义的函数（UDF）是对数据库管理系统定义的函数，可在 SQL 语句中引用它。本章描述由 DB2 UDB XML Extender 所使用的用户定义的函数。

XML Extender 用户定义的函数的类型

XML Extender 提供了用于存储、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 用户定义的函数（UDF）用于 XML 列，但不用于 XML 集合。

所有 UDF 都具有模式名 DB2XML。

以下列表中描述了 XML Extender 函数的类型:

存储函数
存储函数将完整的 XML 文档作为 XML 数据类型插入启用了 XML 的列中。

检索函数
检索函数从 DB2® 数据库中的 XML 列检索 XML 文档。

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 提供了一组用于各种 SQL 数据类型的抽取函数。

更新函数
更新函数修改整个 XML 文档、指定的元素内容或属性值，并返回带有位置路径指定的更新值的 XML 文档副本。

generate_unique 函数
generate_unique 函数返回唯一键。

验证函数
验证函数针对 XML 模式或 DTD 验证 XML 文档。

XML 用户定义的函数允许您对一般 SQL 数据类型执行搜索。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*搜索*。例如，这种搜索能力可用于改进特定 Web 站点的可用性，该 Web 站点发布大量可读文本（如新闻文章）或*电子数据交换（EDI）*应用程序，它们具有可频繁搜索的元素或属性。

限制: 当使用 UDF 中的参数标记时，Java™ 数据库（JDBC）限制要求必须将 UDF 的参数标记强制转型为返回数据将插入到其中的列的数据类型。

存储函数

XML Extender 中的存储函数概述

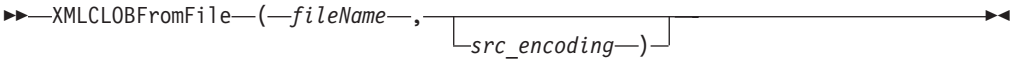
使用存储函数将 XML 文档插入 DB2 UDB 数据库。可直接在 INSERT 或 SELECT 语句中使用 UDT 的缺省数据类型转换函数。另外，XML Extender 提供了 UDF 来从除 UDT 基本数据类型之外的源获取 XML 文档并将其转换为指定的 UDT。

XMLCLOBFromFile() 函数

用途:

从服务器文件读取 XML 文档并将该文档作为 XMLCLOB 类型返回。

语法:



参数:

表 27. XMLCLOBFromFile 参数

参数	数据类型	描述
fileName	VARCHAR(512)	全限定服务器文件名。
src_encoding	VARCHAR(100)	源文件的编码。

结果:

作为 LOCATOR 的 XMLCLOB

示例:

下列示例从服务器上的文件读取 XML 文档并将其作为 XMLCLOB 类型插入到 XML 列中。服务器文件的编码显式指定为 iso-8859-1。

```
EXEC SQL INSERT INTO sales_tab(ID, NAME, ORDER)
  VALUES('1234', 'Sriram Srinivasan',
           XMLCLOBFromFile('
           dxxsamples/xml/getstart.xml', 'iso-8859-1'))
```

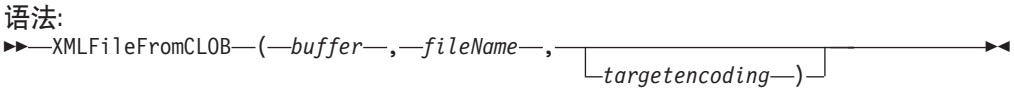
SALES_TAB 表中的列 ORDER 已定义为 XMLCLOB 类型。

XMLFileFromCLOB() 函数

用途:

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 28. 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。
targetencoding	VARCHAR(100)	输出文件的编码。

结果:

XMLFILE

示例:

以下示例以 CLOB 定位器（一个主变量，该变量的值在数据库服务器中表示单个 LOB 值）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。函数将以 ibm-808 对输出文件进行编码。

```
              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, 'dxxsamples/xml/getstart.xml', 'ibm-808'))
```

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。

XMLFileFromVarchar() 函数

用途:

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 29. XMLFileFromVarchar 参数

参数	数据类型	描述
buffer	VARCHAR(3K)	包含 XML 文档的缓冲区。
fileName	VARCHAR(512)	全限定服务器文件名。
targetencoding	VARCHAR(100)	输出文件的编码。

结果:

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。函数将以 iso-8859-1 对输出文件进行编码。

```
              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, 'dxxsample/xml/getstart.xml', 'iso-8859-1'))
```

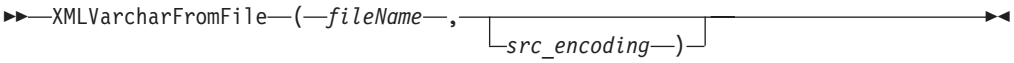
SALES_TAB 表中的列 ORDER 被定义为 XMLFILE 类型。

XMLVarcharFromFile() 函数

用途:

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 30. XMLVarcharFromFile 参数

参数	数据类型	描述
fileName	VARCHAR(512)	全限定服务器文件名。
src_encoding	VARCHAR(100)	源文件的编码。

结果:

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。服务器文件的编码显式指定为 ibm-808。

```
EXEC SQL INSERT INTO sales_tab(ID, NAME, ORDER)
   VALUES('1234', 'Sriram Srinivasan',
          XMLVarcharFromFile('dxxsample/xml/getstart.xml', 'ibm-808'))
```

在本示例中，记录被插入 SALES_TAB 表。函数 XMLVarcharFromFile() 将 XML 文档从显式指定以 ibm-808 编码的文件导入 DB2 UDB，并将其作为 XMLVARCHAR 存储。

检索函数

XML Extender 中的检索函数

XML Extender 提供了用于进行检索的重载函数 Content()。此重载函数引用一套具有相同名称的检索函数，但这些函数根据被检索数据所处位置的不同而有不同的行为。还可使用缺省数据类型转换函数来将 XML UDT 转换成基本数据类型。

Content() 函数提供了下列类型的检索:

- **从服务器的外部存储器检索至客户机的主变量。**

  当 XML 文档是作为外部服务器文件存储时，可使用 Content() 来将其检索到内存缓冲区中。可使用 “Content(): 从 XMLFILE 检索至 CLOB” 来执行此操作。

- **从内部存储器检索至外部服务器文件**

  还可以使用 Content() 来检索存储在 DB2 UDB 内的 XML 文档，并将其存储至 DB2 UDB 服务器的文件系统上的服务器文件中。下列 Content() 函数用于将信息存储在外部服务器文件中:

  – 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

以下用户定义的函数具有指定源文件或输出文件的编码的新参数。此参数的值是由 ICU 识别的任何代码页名称。

```
db2xml.XMLVarcharFromFile(filename varchar(512), src_encoding  varchar(100))
returns XMLVarchar

db2xml.XMLCLOBFromFile(filename varchar(512), src_encoding varchar(100))
returns XMLCLOB AS LOCATOR

db2xml.XMLFileFromVarchar(doc varchar(3000), targetfilename varchar(512),
targetencoding varchar(100))
returns XMLFile

db2xml.XMLFileFromCLOB(doc CLOB(2G) as LOCATOR, targetfilename varchar(512),
 targetencoding varchar(100))
returns XMLFile

db2xml.Content(doc XMLVarchar, targetfilename varchar(512),
 targetencoding varchar(100))
returns varchar(512)

db2xml.Content(doc XMLCLOB as LOCATOR, targetfilename varchar(512),
 targetencoding varchar(100))
returns varchar(512)
```

示例:

将文件 `/home/collins/xml/entail.xml` 的内容导入 varchar 缓存，并指定源文件以 iso-8859-1 编码:

```
db2xml.XMLVarcharFromFile('/home/collins/xml/entail.xml', 'iso-8859-1')
```

将文件导入 varchar，并从 iso-8859-1 转换至数据库代码页。

将 varchar 缓存导出至文件 /home/raskolnikov/xml/confession.xml，并指定输出文件应以 ibm-808 编码:

```
db2xml.Content('<sequence><thought>I did it!</thought></sequence>',
'/home/raskolnikov/xml/confession.xml', 'ibm-808')
```

将缓存的内容导出至文件，并从数据库代码页转换至 ibm-808。然后，适当更新 XML 文件的编码声明。

下一节中的示例假定您正在使用 DB2 UDB 命令 shell 程序，在此程序中，不需要在每个命令的开头输入 “DB2”。

Content(): 从 XMLFILE 检索至 CLOB

用途:

从服务器文件检索数据并将其存储在 CLOB LOCATOR 中。

语法:

参数:

表 31. 至 CLOB 参数的 XMLFILE

参数	数据类型	描述
xmlobj	XMLFILE	XML 文档。

结果:

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。

相关任务:

- 第 90 页的『更新、删除和检索 XML 集合中的数据』

Content(): 从 XMLVARCHAR 检索至外部服务器文件

用途:

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。

参数:

表 32. 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。
targetencoding	VARCHAR(100)	输出文件的编码。

结果:

VARCHAR(512)

示例:

以下示例检索作为 XMLVARCHAR 类型存储的 XML 内容，并将其存储在服务器上的外部文件中。UDF 以 “ibm-808” 对文件进行编码。

```
CREATE table app1 (id int NOT NULL, order DB2XML.XMLVarchar);
INSERT into app1 values (1, '<?xml version="1.0"?>
   <!DOCTYPE SYSTEM
"dxxsample/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>
      </Part>
    </Order>');

SELECT DB2XML.Content(order, '
dxxsamples/dad/getstart_column.dad', 'ibm-808')
  from app1 where ID=1;
```

相关任务:

- 第 72 页的『检索 XML 文档的方法』

相关参考:

- 第 129 页的『XML Extender 中的检索函数』

Content(): 从 XMLCLOB 检索至外部服务器文件

用途:

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 33. 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。
targetencoding	VARCHAR(100)	输出文件的编码。

结果:

VARCHAR(512)

示例:

下列示例检索作为 XMLCLOB 类型存储的 XML 内容，并将其存储在服务器上的外部文件中。UDF 以 “ibm-808” 对文件进行编码。

```
CREATE table app1 (id int NOT NULL, order DB2XML.XMLCLOB );

INSERT into app1 values (1, '<?xml version="1.0"?>
  <!DOCTYPE SYSTEM
"dxxsamples/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>
      </Part>
    </Order>');

SELECT DB2XML.Content(order,
'dxxsamples/xml/getstart.xml', 'ibm-808')
 from app1 where ID=1;
```

抽取函数

XML Extender 中的抽取函数

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。

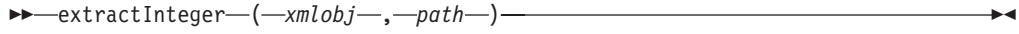
这些示例假定您正在使用 DB2 UDB 命令 shell 程序，在此程序中，不需要在每个命令的开头输入 “DB2”。

extractInteger() 和 extractIntegers()

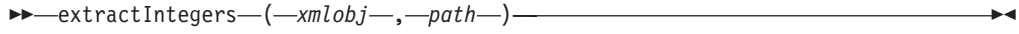
用途:

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:



表函数:



参数:

表 34. extractInteger 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。

返回的类型:

INTEGER

示例:

在以下示例中，当 key 的属性值 = “1” 时返回一个值。将该值作为 INTEGER 来进行抽取的。这些示例假定您正在使用 DB2 UDB 命令 shell 程序，在此程序中，不需要在每个命令的开头输入 “DB2”。

```
CREATE TABLE t1(key INT);
INSERT INTO t1 values (
    DB2XML.extractInteger(DB2XML.XMLFile('
    dxxsamples/xml/getstart.xml'),
    '/Order/Part[@color="black "]/key'));
SELECT * from t1;
```

表函数示例:

在以下示例中，将所有销售订单的每个订单键作为 INTEGER 来进行抽取。这些示例假定您正在使用 DB2 UDB 命令 shell 程序，在此程序中，不需要在每个命令的开头输入 “DB2”。

```
SELECT *
  FROM TABLE(
  DB2XML.extractIntegers(DB2XML.XMLFile('dxxsamples/xml/getstart.xml'),
      '/Order/Part/key'))   AS X;
```

相关概念:

- 第 107 页的『XML Extender 的 UDT 和 UDF 名称』
- 第 125 页的『XML Extender 用户定义的函数的类型』

相关参考:

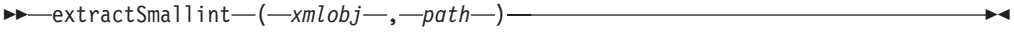
- 第 133 页的『XML Extender 中的抽取函数』

extractSmallint() 和 extractSmallints()

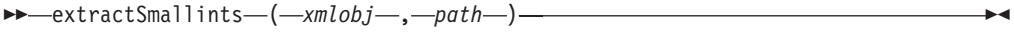
用途:

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:



表函数:



参数:

表 35. extractSmallint 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。

返回的类型:

SMALLINT

示例:

在以下示例中，将所有销售订单中的 key 的值作为 SMALLINT 来进行抽取。这些示例假定您正在使用 DB2 UDB 命令 shell 程序，在此程序中，不需要在每个命令的开头输入 “DB2”。

```
CREATE TABLE t1(key INT);
INSERT INTO t1 values (
    DB2XML.extractSmallint(db2xml.xmlfile('dxxsamples/xml/getstart.xml'),
    '/Order/Part[@color="black "]/key'));
SELECT * from t1;
```

表函数示例:

在以下示例中，将所有销售订单中的 key 的值作为 SMALLINT 来进行抽取。这些示例假定您正在使用 DB2 UDB 命令 shell 程序，在此程序中，不需要在每个命令的开头输入 “DB2”。

```
SELECT *
  FROM TABLE(
      DB2XML.extractSmallints(DB2XML.XMLFile('dxxsamples/xml/getstart.xml'),
      '/Order/Part/key'))   AS X;
```

相关概念:

- 第 69 页的『对 XML 列数据使用索引』
- 第 107 页的『XML Extender 的 UDT 和 UDF 名称』
- 第 125 页的『XML Extender 用户定义的函数的类型』

相关参考:

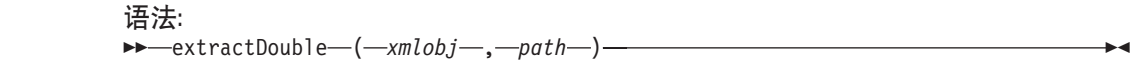
- 第 133 页的『XML Extender 中的抽取函数』
- 第 201 页的『XML Extender 存储过程返回码』

extractDouble() 和 extractDoubles()

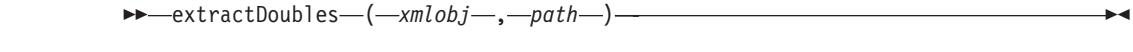
用途:

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:



表函数:



参数:

表 36. extractDouble 函数参数

参数	数据类型	描述
xmlobj	XMLVARCHAR、XMLFILE 或 XMLCLOB	列名。

表 36. extractDouble 函数参数 (续)

参数	数据类型	描述
path	VARCHAR	元素或属性的位置路径。

返回的类型:

DOUBLE

示例:

以下示例自动将订单中的价格从 DOUBLE 类型转换为 DECIMAL 类型。这些示例假定您正在使用 DB2 命令 shell 程序，在此程序中，不需要在每个命令的开头输入 “DB2”。

```
CREATE TABLE t1(price DECIMAL(9,2));
INSERT INTO t1 values (
    DB2XML.extractDouble(DB2XML.xmlfile('dxxsamples/xml/getstart.xml'),
    '/Order/Part[@color="black "]/ExtendedPrice'));
SELECT * from t1;
```

表函数示例:

在以下示例中，将销售订单每个部分中的 ExtendedPrice 的值作为 DOUBLE 来进行抽取。这些示例假定您正在使用 DB2 UDB 命令 shell 程序，在此程序中，不需要在每个命令的开头输入 DB2 UDB。

```
SELECT CAST(RETURNEDDOUBLE AS DOUBLE)
  FROM TABLE(
      DB2XML.extractDoubles(DB2XML.XMLFile('dxxsamples/xml/getstart.xml'),
      '/Order/Part/ExtendedPrice'))   AS X;
```

相关概念:

- 第 107 页的『XML Extender 的 UDT 和 UDF 名称』

相关参考:

- 第 133 页的『XML Extender 中的抽取函数』

extractReal() 和 extractReals()

用途:

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:

表函数:

参数:

表 37. extractReal 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。

返回的类型:

REAL

示例:

在以下示例中，将 ExtendedPrice 的值作为 REAL 来进行抽取。这些示例假定您正在使用 DB2 UDB 命令 shell 程序，在此程序中，不需要在每个命令的开头输入 “DB2”。

```
CREATE TABLE t1(price DECIMAL(9,2));
INSERT INTO t1 values (
    DB2XML.extractReal(DB2XML.xmlfile('dxxsamples/xml/getstart.xml'),
    '/Order/Part[@color="black"]/ExtendedPrice'));
SELECT * from t1;
```

表函数示例:

在以下示例中，将 ExtendedPrice 的值作为 REAL 来进行抽取。这些示例假定您正在使用 DB2 UDB 命令 shell 程序，在此程序中，不需要在每个命令的开头输入 “DB2”。

```
SELECT CAST(RETURNEDREAL AS REAL)
  FROM TABLE(
      DB2XML.extractReals(DB2XML.XMLFile('dxxsamples/xml/getstart.xml'),
      '/Order/Part/ExtendedPrice'))   AS X;
```

相关概念:

- 第 107 页的『XML Extender 的 UDT 和 UDF 名称』
- 第 125 页的『XML Extender 用户定义的函数的类型』

相关参考:

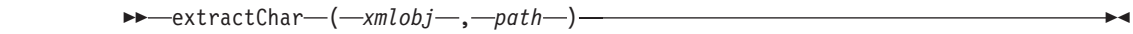
- 第 133 页的『XML Extender 中的抽取函数』
- 第 201 页的『XML Extender UDF 返回码』

extractChar() 和 extractChars()

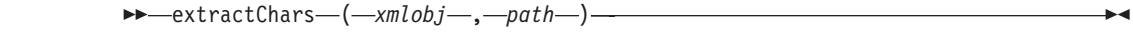
用途:

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:



表函数:



参数:

表 38. extractChar 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。

返回的类型:

CHAR

示例:

在以下示例中，将 Name 的值作为 CHAR 来进行抽取。这些示例假定您正在使用 DB2 UDB 命令 shell 程序，在此程序中，不需要在每个命令的开头输入 “DB2”。

```
CREATE TABLE t1(name char(30));
INSERT INTO t1 values (
    DB2XML.extractChar(DB2XML.xmlfile('dxxsamples/xml/getstart.xml'),
    '/Order/Customer/Name'));
SELECT * from t1;
```

表函数示例:

在以下示例中，将 Color 的值作为 CHAR 来进行抽取。这些示例假定您正在使用 DB2 UDB 命令 shell 程序，在此程序中，不需要在每个命令的开头输入 “DB2”。

```
SELECT *
  FROM TABLE(
      DB2XML.extractChars(DB2XML.XMLFile('dxxsamples/xml/getstart.xml'),
      '/Order/Part/@color'))   AS X;
```

相关参考:

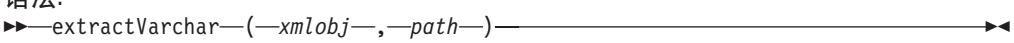
- 第 133 页的『XML Extender 中的抽取函数』
- 第 xi 页的『如何阅读语法图』

extractVarchar() 和 extractVarchars()

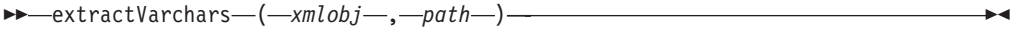
用途:

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:



表函数:



参数:

表 39. extractVarchar 函数参数

参数	数据类型	描述
xmlobj	XMLVARCHAR、XMLFILE 或 XMLCLOB	列名。

表 39. extractVarchar 函数参数 (续)

参数	数据类型	描述
path	VARCHAR	元素或属性的位置路径。

返回的类型:

VARCHAR(4K)

示例:

在带有超过 1000 个 XML 文档（这些文档存储在 SALES_TAB 表的 ORDER 列中）的数据库中，您可能想要查找所有已订购了 ExtendedPrice 大于 2500.00 的项的客户。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
```

这些示例假定您正在使用 DB2 UDB 命令 shell 程序，在此程序中，不需要在每个命令的开头输入 “DB2”。UDF extractVarchar() 采用 ORDER 列作为输入，并采用位置路径 /Order/Customer/Name 作为选择标识。此 UDF 返回客户的姓名。借助 WHERE 子句，抽取函数仅对那些 ExtendedPrice 大于 2500.00 的订单进行求值。

表函数示例:

在带有超过 1000 个 XML 文档（这些文档存储在 SALES_TAB 表的 ORDER 列中）的数据库中，您可能想要查找所有已订购了 ExtendedPrice 大于 2500.00 的项的客户。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
```

这些示例假定您正在使用 DB2 UDB 命令 shell 程序，在此程序中，不需要在每个命令的开头输入 “DB2”。UDF extractVarchar() 采用 ORDER 列作为输入，并采用位置路径 /Order/Customer/Name 作为选择标识。此 UDF 返回客户的姓名。借助 WHERE 子句，抽取函数仅对那些 ExtendedPrice 大于 2500.00 的订单进行求值。

在以下示例中，将 Name 的值作为 VARCHAR 来进行抽取。这些示例假定您正在使用 DB2 UDB 命令 shell 程序，在此程序中，不需要在每个命令的开头输入 “DB2”。

```
CREATE TABLE t1(name varchar(30));
INSERT INTO t1 values (
    DB2XML.extractVarchar(DB2XML.xmlfile('dxxsamples/xml/getstart.xml'),
    '/Order/Customer/Name'));
SELECT * from t1;
```

表函数示例:

在以下示例中，将 Color 的值作为 VARCHAR 来进行抽取。这些示例假定您正在使用 DB2 UDB 命令 shell 程序，在此程序中，不需要在每个命令的开头输入 “DB2”。

```
SELECT*
  FROM TABLE(
      DB2XML.extractVarchars(DB2XML.XMLFile('dxxsamples/xml/getstart.xml'),
      '/Order/Part/@color'))   AS X;
```

相关概念:

- 第 107 页的『XML Extender 的 UDT 和 UDF 名称』
- 第 125 页的『XML Extender 用户定义的函数的类型』

相关参考:

- 第 133 页的『XML Extender 中的抽取函数』
- 第 201 页的『XML Extender UDF 返回码』

extractCLOB() 和 extractCLOBs()

用途:

抽取 XML 文档段，带有元素和属性标记以及元素和属性的内容，包括子元素。此函数与其他抽取函数不同，其他抽取函数只返回元素和属性的内容。extractClob（s）函数用来抽取文档段，而 extractVarchar（s）和 extractChar（s）用来抽取简单的值。

语法:

表函数:

参数:

表 40. extractCLOB 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。

返回的类型:

CLOB(10K)

示例:

在此示例中，从采购订单抽取所有 name 元素内容和标记。这些示例假定您正在使用 DB2 UDB 命令 shell 程序，在此程序中，不需要在每个命令的开头输入 “DB2”。

```
CREATE TABLE t1(name DB2XML.xmlclob);
INSERT INTO t1 values (
    DB2XML.extractClob(DB2XML.xmlfile('dxxsamples/xml/getstart.xml'),
    '/Order/Customer/Name'));
SELECT * from t1;
```

表函数示例:

在此示例中，从采购订单抽取所有 color 属性。这些示例假定您正在使用 DB2 UDB 命令 shell 程序，在此程序中，不需要在每个命令的开头输入 “DB2”。

```
SELECT *
  FROM TABLE(
      DB2XML.extractCLOBs(DB2XML.XMLFile('dxxsamples/xml/getstart.xml'),
      '/Order/Part/@color'))   AS X;
```

相关概念:

- 第 125 页的『XML Extender 用户定义的函数的类型』

相关参考:

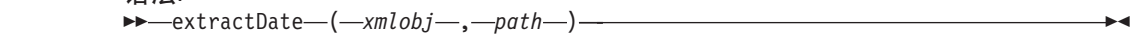
- 第 133 页的『XML Extender 中的抽取函数』

extractDate() 和 extractDates()

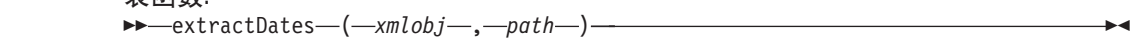
用途:

从 XML 文档抽取元素内容或属性值，并返回 DATE 类型的数据。日期必须具有此格式：YYYY-MM-DD。

语法:



表函数:



参数:

表 41. extractDate 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。

返回的类型:

DATE

示例:

在以下示例中，将 ShipDate 的值作为 DATE 来进行抽取。这些示例假定您正在使用 DB2 UDB 命令 shell 程序，在此程序中，不需要在每个命令的开头输入 “DB2”。

```
CREATE TABLE t1(shipdate DATE);
INSERT INTO t1 values (
    DB2XML.extractDate(DB2XML.xmlfile('dxxsamples/xml/getstart.xml'),
    '/Order/Part[@color="red   "]/Shipment/ShipDate'));
SELECT * from t1;
```

表函数示例:

在以下示例中，将 ShipDate 的值作为 DATE 来进行抽取。

```
SELECT *
  FROM TABLE(
      DB2XML.extractDates(DB2XML.XMLFile('dxxsamples/xml/getstart.xml'),
      '/Order/Part[@color="black "]/Shipment/ShipDate'))   AS X;
```

相关概念:

- 第 125 页的『XML Extender 用户定义的函数的类型』

相关参考:

- 第 133 页的『XML Extender 中的抽取函数』

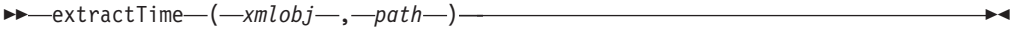
- 第 201 页的『XML Extender UDF 返回码』

extractTime() 和 extractTimes()

用途:

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:



表函数:

参数:

表 42. extractTime 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。

返回的类型:

TIME

示例:

这些示例假定您正在使用 DB2 UDB 命令 shell 程序，在此程序中，不需要在每个命令的开头输入 “DB2”。

```
CREATE TABLE t1(testtime  TIME);
INSERT INTO t1 values (
    DB2XML.extractTime(DB2XML.XMLCLOB(
         '<stuff><data>11.12.13</data></stuff>'), '//data'));
SELECT * from t1;
```

表函数示例:

```
select *
from table(
    DB2XML.extractTimes(DB2XML.XMLCLOB(
        '<stuff><data>01.02.03</data><data>11.12.13</data></stuff>'),
    '//data')) as x;
```

相关概念:

- 第 107 页的『XML Extender 的 UDT 和 UDF 名称』
- 第 125 页的『XML Extender 用户定义的函数的类型』

相关参考:

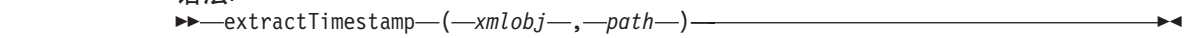
- 第 133 页的『XML Extender 中的抽取函数』

extractTimestamp() 和 extractTimestamps()

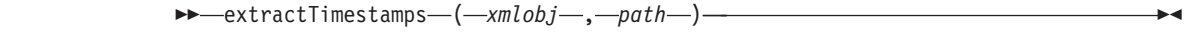
用途:

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:



表函数:



参数:

表 43. extractTimestamp 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。

返回的类型:

TIMESTAMP

示例:

这些示例假定您正在使用 DB2 UDB 命令 shell 程序，在此程序中，不需要在每个命令的开头输入 “DB2”。

```
CREATE TABLE t1(testtimestamp  TIMESTAMP);
INSERT INTO t1 values (
    DB2XML.extractTimestamp(DB2XML.XMLCLOB(
        '<stuff><data>2003-11-11-11.12.13.888888</data></stuff>'),
        '//data'));
SELECT * from t1;
```

表函数示例:

```
select * from
table(DB2XML.extractTimestamps(DB2XML.XMLClob(
    '<stuff><data>2003-11-11-11.12.13.888888
    </data><data>2003-12-22-11.12.13.888888</data></stuff>'),
    '//data')) as x;
```

XML Extender 将自动标准化从 XML 文档抽取的时间戳记以符合 DB2 时间戳记格式（如果需要）。时间戳记标准化为 yyyy-mm-dd-hh.mm.ss.nnnnnn 格式或 yyyy-mm-dd-hh mm.ss.nnnnnn 格式。例如:

```
2003-1-11-11.12.13
```

将标准化为:

```
2003-01-11-11.12.13.000000
```

相关概念:

- 第 107 页的『XML Extender 的 UDT 和 UDF 名称』
- 第 125 页的『XML Extender 用户定义的函数的类型』

相关参考:

- 第 133 页的『XML Extender 中的抽取函数』
- 第 201 页的『XML Extender UDF 返回码』

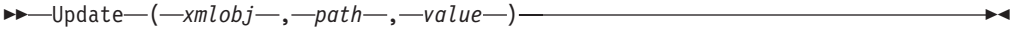
XML Extender 中的更新函数

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。还可使用缺省数据类型转换函数将 SQL 基本类型转换为 XML UDT。

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 44. 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。 **限制**: Update 函数没有禁用输出转义的选项；不能使用此函数插入 extractClob 的输出（它是带标记的片断）。仅使用文本值。

限制: 注意 Update 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，支持下列谓词:

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

不支持下列谓词:

```
'/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 = db2xml.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 Update 函数更改一个或多个 XML 文档中的值时，它替换 XML 列中的 XML 文档。基于 XML 解析器的输出，保留了原始文档的某些部分，而其他部分则丢失或被更改。下列各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。

Update() 函数如何处理 XML 文档

当 Update() 函数替换 XML 文档时，它必须基于 XML 解析器输出重新构造文档。表 45 用示例描述如何处理文档的各部分。

表 45. Update 函数规则

项或节点类型	XML 文档代码示例	更新后的状态
XML 声明	<?xml version='1.0' encoding='utf-8' standalone='yes' >	保留 XML 声明。
DOCTYPE 声明 <!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] >		保留文档类型声明:
注释	<!-- comment -->	注释保留在根元素之外。 废弃根元素内部的注释。
元素	<books> *content* </books>	保留元素。
属性	id='1' date="01/02/2003"	保留元素的属性。 • 更新之后，使用双引号来描述值。 • 属性中的数据丢失。 • 替换实体。

表 45. Update 函数规则 (续)

项或节点类型	XML 文档代码示例	更新后的状态
文本节点	`This chapter is about my dog &mydog;.`	保留文本节点（元素内容）。 • 文本节点内的数据丢失。 • 替换实体。

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。这表示如果文档具有多次出现的位置路径，则 Update() 函数使用 *value* 参数中所提供的值替换现有的值。

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。Update() 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。

生成唯一函数

用途

生成唯一函数返回一个字符串，该字符串与相同函数的任何其他执行结果比较都是唯一的。此函数没有自变量（必须指定空的括号）。该函数的结果是一个唯一值。结果不能为空值。

语法

返回值

VARCHAR(13)

示例

以下示例使用 db2xml.generate_unique() 对要为其创建索引的列生成唯一键。

```
<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,
 (select db2xml.generate_unique()
  as ship_id, date, mode, part_key from ship_tab) as 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>
```

验证函数

DB2 XML Extender 提供两个用户定义的函数（UDF）来针对 XML 模式或 DTD 验证 XML 文档。

如果满足相关联的元素类型规则，则 XML 文档中的元素根据给定模式是有效的。如果所有元素都有效，则整个文档有效。然而，在 DTD 的情况下，无法要求特定的根元素。如果文档有效，验证函数返回 1，或者如果文档无效，返回 0 并在跟踪文件中写入错误消息。函数有:

db2xml.svalidate:
针对指定模式，验证 XML 文档实例。

db2xml.dvalidate:
针对指定的 DTD，验证 XML 文档实例。

SVALIDATE() 函数

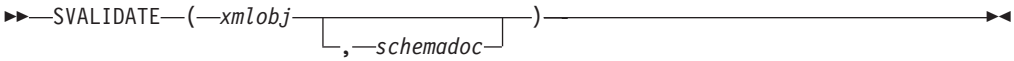
此函数针对指定的模式（或在 XML 文档中命名的模式）验证 XML 文档，并且如果文档有效则返回 1，否则返回 0。此函数假定文件系统上存在 XML 文档和模式，或者作为 CLOB 存在于 DB2 中。

在运行 SVALIDATE 函数之前，通过运行下列命令，确保通过 数据库启用了 XML Extender:

```
CALL QDBXM/QZXMADM PARM(enable_db mydbname)
```

如果 XML 文档未能通过验证，则会将错误消息写入 XML Extender 跟踪文件中。执行 SVALIDATE 命令之前，启用跟踪。有关启用跟踪的信息，请参阅第 199 页的『为 XML Extender 启动跟踪』。

语法



参数

表 46. SVALIDATE 参数

参数	数据类型	描述
xmlobj	VARCHAR(256)	要验证的 XML 文档的文件路径。
	CLOB(2G)	包含要验证文档的 XML 列。
schemadoc	VARCHAR(256)	模式文档的文件路径。
	CLOB(2G)	包含模式的 XML 列。

示例

示例 1: 此示例针对文档中指定的模式验证 equiplog2001.xml。

```
select db2xml.svalidate('/dxxsamples/xml/equiplog2001..xml') from db2xml.onerow
```

示例 2: 本示例使用指定模式验证 XML 文件，并且将文档和模式都存储在 DB2 UDB 表中。

```
select db2xml.svalidate(doc,schema) from db2xml.onerow
```

DVALIDATE() 函数

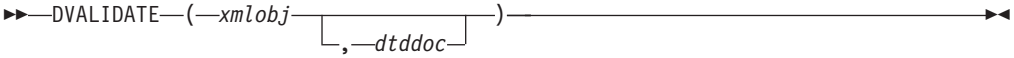
此函数针对指定的 DTD（或在 XML 文档中指定的模式）验证 XML 文档，如果文档有效，则返回 1，否则返回 0。此函数假定文件系统上存在 XML 文档和 DTD，或者作为 CLOB 存在于 DB2 中。

执行 DVALIDATE 函数之前，确保通过运行以下命令对您的数据库启用了 XML Extender:

```
CALL QDBXM/QZXMADM PARM(enable_db mydbname)
```

如果 XML 文档未能通过验证，则会将错误消息写入 XML Extender 跟踪文件中。执行 SVALIDATE 命令之前，启用跟踪。有关启用跟踪的信息，请参阅第 199 页的『为 XML Extender 启动跟踪』。

语法



参数

表 47. DVALIDATE 参数

参数	数据类型	描述
xmlobj	VARCHAR(256)	要验证的 XML 文档的文件路径。
	CLOB(2G)	包含要验证文档的 XML 列。
dtddoc	VARCHAR(256)	DTD 文档的文件路径。
	CLOB(2G)	包含 DTD 的 XML 列，此 DTD 可能出自 DTD_REF 表，也可能出自常规表。

示例

示例 1: 本示例针对文档中指定的 DTD 验证 equiplog2001.xml。

```
select db2xml.dvalidate('/dxxsamples/xml/equiplog2001.xml') from db2xml.onerow
```

示例 2: 本示例使用指定的 DTD 验证 XML 文档，并且文档和 DTD 都位于文件系统中。

```
select db2xml.dvalidate('/dxxsamples/xml/equiplog2001.xml',
'/dxxsamples/dtd/equip.dtd') from db2xml.onerow
```

示例 3: 本示例使用指定的 DTD 验证 XML 文档，并且文档和 DTD 都存储在 DB2 UDB 表中。

```
select db2xml.dvalidate (doc,dtdid) from equiplogs, db2xml.dtd_ref \
    where dtdid='equip.dtd'
```

第 9 章 文档访问定义（DAD）文件

为 XML 列创建 DAD 文件

此任务是更大型的定义和启用 XML 列任务的一部分。

有关 DAD 文件的最新信息，请访问 XML Extender Web 站点，网址为 www.ibm.com/software/data/db2/extenders/xmlext/downloads.html。

要访问 XML 数据并对 XML 表中的 XML 数据启用列，需要定义文档访问定义（DAD）文件。此文件定义需要在列中搜索的数据的属性和关键元素。对于 XML 列，DAD 文件主要指定如何为存储在 XML 列中的文档建立索引。DAD 文件还指定 DTD 或模式以用于验证插入 XML 列的文档。将 DAD 文件存储为 CLOB 数据类型，其大小限制为 100 KB。

先决条件:

在创建 DAD 文件之前，需要:

- 决定您期望在搜索中经常使用哪些元素或属性。XML Extender 将这些指定的元素或属性抽取到副表中，以便快速搜索。
- 定义位置路径以表示副表中的每个已建立索引的元素或属性。还必须指定要将元素或属性转换成什么数据类型。

过程:

要创建 DAD 文件:

1. 在文本编辑器中创建新文档并输入以下语法:

   ```
   <?XML version="1.0"?>
   <!DOCTYPE DAD SYSTEM <"path/dtd/dad.dtd">.
   ```

   "path/dtd/dad.dtd " 是用于 DAD 文件的 DTD 的路径和文件名。在 dxx_install\samples\db2xml\dtd 中提供了 DTD

2. 在步骤 1 的行后面插入 DAD 标记。

   ```
   <DAD>
   </DAD>
   ```

   此元素将包含所有其他元素。

3. 对文档和列指定验证:
   - 如果您想在将文档插入数据库之前，再次针对 DTD 或模式验证整个 XML 文档:
     - 插入正确的标记以指定您想如何验证文档:

       ```
       <dtdid>path/dtd_name.dtd</dtdid>
       ```

     - 插入以下标记来使用模式验证文档:

       ```
       <schemabindings>
       <nonamespacelocation location="path/schema_name.xsd"/>
       </schemabindings>
       ```

     - 通过插入以下标记来验证列:

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

- 如果不想验证文档，则使用以下标记:

```
<validation>NO</validation>
```

4. 插入 <Xcolumn> </Xcolumn> 标记以指定您正在将 XML 列用作 XML 数据的访问和存储方法。
5. 指定副表。对于要创建的每个副表:
   a. 指定 <table></table> 标记。例如:

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

   b. 在表标记中，对于要让副表包含的每个列插入 <column> 标记。每个列有四个属性: name、type、path 和 multi_occurrence。

   示例:

```
<table name="person_names">>
<column name ="fname"
     type="varchar(50)"
     path="/person/firstName"
        multi_occurrence="NO"/>
<column name ="lname"
     type="varchar(50)"
     path="/person/lastName"
        multi_occurrence="NO"/>
</table>
```

   其中:

   name 指定在副表中创建的列的名称。

   type 指示副表中每个已建立索引的元素或属性的 SQL 数据类型。

   path 指定 XML 文档中每个将要为其建立索引的元素或属性的位置路径。

   multi_occurrence
   指示 path 属性引用的元素或属性能否在 XML 文档中出现多次。可能的 **multi_occurrence** 的值是 **YES** 或 **NO**。如果值为 **NO**，则可以对每个表指定多个列。如果值为 **YES**，则只能指定副表中的一个列。
6. 使用 DAD 扩展名来保存文件。

以下示例显示一个完整的 DAD 文件:

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "dxxsamples\dtd\dad.dtd">
<DAD>
<dtid>C:\SG246130\code\person.dtd</dtdid>
<validation>YES</validation>
   <Xcolumn>
  <table name="person_names">
   <column name="fname"
     type="varchar(50)"
     path="/person/firstName"
        multi_occurrence="NO"/>
   <column name="lname"
     type="varchar(50)"
     path="/person/lastName"
        multi_occurrence="NO"/>
     </table>
<table name="person_phone_number">
   <column name="pnumber"
```

```
     type="varchar(20)"
     path="/person/phone/number"
     multi_occurrence="YES"/>
     </table>
<table name="person_phone_number">
   <column name="pnumber"
     type="varchar(20)"
     path="/person/phone/number"
     multi_occurrence="YES"/>
     </table>
<table name="pesron_phone_type">
   <column name="ptype"
     type="varchar(20)"
     path="/person/phone/type"
     multi_occurrence="YES"/>
     </table>
   <Xcolumn>
</DAD>
```

既然创建了 DAD 文件，用于定义和启用 XML 列的下一步骤是创建要在其中存储 XML 文档的表。

相关概念:

- 第 81 页的『XML 集合作为存储和访问方法』
- 第 151 页的『XML 集合的 DAD 文件』
- 第 165 页的『Dad 检查程序』

相关任务:

- 第 165 页的『使用 DAD 检查程序』

XML 集合的 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，您根据这些表来组成文档。还可使用 DAD 文件将文档分解到 DB2 UDB 表中。

例如，如果 XML 文档中有一个名为 <Tax> 的元素，则需要将 <Tax> 映射至一个名为 TAX 的列。使用 DAD 文件来定义 XML 数据与关系数据之间的关系。

必须在启用集合，或在 XML 集合的存储过程中使用 DAD 文件时指定此 DAD 文件。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，请在 DAD 文件中使用 <Xcollection> 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成 XML 文档。

XML 集合是一组包含 XML 数据的表。应用程序可启用任何用户表的 XML 集合。这些用户表可以是现有业务数据表，也可以是 XML Extender 最近创建的表。

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

标识元素，该元素可以是根元素或子元素。

text_node

表示元素的 CDATA 文本。

attribute_node

表示元素的属性。

图 14 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 ""dxxsamples\dtd\dad.dtd">
<DAD>
  ...
<Xcollection>
<SQL_stmt>
       ...
</SQL_stmt>
<prolog>?xml version="1.0"?</prolog>
<doctype>!DOCTYPE Order SYSTEM
                  ""dxxsamples\dtd\getstart.dtd""</doctype>
<root_node>
   <element_node name="Order">         --> Identifies the element <Order>
     <attribute_node name="key">       --> Identifies the attribute "key"
        <column name="order_key"/>     --> Defines the name of the column,
                                       "order_key", to which the
                                       element and attribute are
                                       mapped
     </attribute_node>
     <element_node name="Customer">    --> Identifies a child element of
                                              <Order> as <Customer>
   <text_node>                    --> Specifies the CDATA text for
                                      the element <Customer>
    <column name="customer">      --> Defines the name of the column,
                                      "customer", to which the child
                                                  element is mapped
        </text_node>
   </element_node>
        ...
   </element_node>

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

图 14. 映射至 XML 集合表的 XML 文档的节点定义

在本示例中，前两列将元素和属性映射至这些节点。

可使用 XML Extender 管理向导或编辑器来创建和更新 DAD 文件。

相关概念:

- 第 93 页的『XML 集合的映射方案』

SQL 组合

可以使用同名的列组成 XML 文档。必须使用唯一别名来标识所选择的同名列（即使它们来自不同的表），以使 SQL 语句的 SELECT 子句中的每个变量都不相同。以下示例显示了如何对同名的列指定唯一的别名。

```
<SQL_stmt>select o.order_key as oorder_key,      key customer_name, customer_email,
                 p.part_key p.order_key as porder_key,
                 color, qty, price, tax, ship_id, date, mode
            from order_tab o.part_tab p
            order by order_key, part_key</SQL_stmt>
```

还可以使用具有已生成随机值的列组成 XML 文档。如果 DAD 文件中的 SQL 语句具有随机值，则必须对随机值函数指定别名才能在 ORDER BY 子句中使用它。因为该值不与给定表中的任何列相关联，所以这是必需的要求。请查看下列示例中 ORDER BY 子句末尾的 generate_unique 的别名。

```
<SQL_stmt>select o.order_key, customer_name,customer_email,
                 p.part_key,color,qty,price,tax,ship_id,
                 date, mode
          from order_tab o,part_tab p,

table (select db2xml.generate_unique()
as ship_id, date, mode,
                           part_key
                      from ship_tab) s
          where o.order_key=1 and p.price>2000 and
                o.order_key=o.order_key and s.part_key
          order by order_key, part_key,ship_id</SQL_stmt>
```

RDB 节点组合

下列限制适用于 RDB 节点组合:

- 与任何非 root_node RDB 节点 DAD 文件相关联的条件都必须与文字作比较。
- 与顶级 RDB_node 相关联的条件中的每个等式都指定两个表的列之间的连接关系，并且独立于其他等式进行应用。也就是说，通过 AND 连接的所有谓词并不是同时对单个连接条件应用；当组成文档时，它们模拟外连接。每一对表之间的父代 - 子代关系由它们在 DAD 文件中的相对嵌套确定。例如:

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

由具有空值的行组成

可以使用具有空值的列来组成 XML 文档。

以下示例说明如何能够由表 *MyTable* 生成 XML 文档，此表具有列 *Col 1* 包含空值的行。此示例中使用的 DAD 是 *nullcol.dad*。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:\dxx\dtd\dad.dtd">
<DAD>
<validation>NO validation>NO>
<Xcollection>
<SQL_stmt>SELECT 1 as X, Col1 FROM MyTable order by X, Col1<\SQL_stmt>
<prolog>?xml version="1.0"?prolog>?xml version="1.0"?>
<doctype>!DOCTYPE Order SYSTEM "e:\t3xml\x.dtd">
<root_node>
<element_node name="MyColumn">
<element_node name="Column1" multi_occurrence="YES">
```

```
  <text_node>
  <column name="Col1"/>
  </text_node>
 </element_node>
   </element_node>
</roott_node>
</Xcollection>
</DAD>

MyTable

_________
| Col 1 |
|_______|
| 1 |
|_______|
| 3 |
|_______|
| - |
|_______|
```

运行 tests2x mydb nullcol.dad result_tab 或使用 dxxGenXML 来生成以下文档: 请注意, 第三列元素表示空值。

```
<?xml version="1.0"?>
<!DOCTYPE Order SYSTEM "e:\t3xml\x.dtd">
<MyColumn>
 <Column1>1</Column1>
 <Column1>3</Column1>
 <Column1></Column1>
</MyColumn>
```

- 与任何非 root_node RDB 节点 DAD 文件相关联的条件都必须与文字作比较。
- 与任何 DAD 中较低级别 RDB 节点相关联的条件都必须与文字作比较。
- 与 root_node 相关联的条件描述 RDB 节点组合中涉及的表之间的关系。例如, 主外键关系。
- 与顶层 RDB_node 相关联的条件中的每个等式都指定两个表的列之间的连接关系, 并且独立于其他等式进行应用。也就是说, 通过 AND 连接的所有谓词并不是同时对单个连接条件应用, 当组成文档时, 它们模拟外连接。每一对表之间的父代 - 子代关系由它们在 DAD 文件中的相对嵌套确定。例如:

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

用于 DAD 文件的 DTD

本主题描述用于文档访问定义（DAD）文件的文档类型声明（DTD）。DAD 文件本身是树形结构的 XML 文档, 并且需要 DTD。DTD 文件名为 dad.dtd。以下示例显示用于 DAD 文件的 DTD。

```
<?xml encoding="US-ASCII"?>

  <!ELEMENT DAD ((schemabindings | dtdid)?, validation,
  (Xcolumn | Xcollection))>
  <!ELEMENT dtdid (#PCDATA)>
  <!ELEMENT schemabindings (nonamespacelocation)>
  <!ELEMENT nonamespacelocation (empty)>
  <!ATTLIST nonamespacelocation location CDATA #REQUIRED>
  <!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?, prolog, doctype, root_node)>
  <!ELEMENT SQL_stmt (#PCDATA)>
  <!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 process_instruction_node (#PCDATA)>
```

DAD 文件有四个主要元素:

- DTDID
- validation
- 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。语法示例摘自前述示例。

DTDID 元素

提供给 XML Extender 的 DTD 存储在 DTD_REF 表中。每个 DTD 由 DAD 文件的 DTDID 标记中提供的唯一标识来标识。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。对于 XML 集合，仅当验证输入和输出 XML 文档时才需要此元素。 对于 XML 列，仅在验证输入 XML 文档时需要此元素。DTDID 必须与在 XML 文档的 doctype 中指定的 SYSTEM 标识相同。

语法: `<!ELEMENT dtdid (#PCDATA)>`

validation 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 XML 列的索引方案。它由零个或多个表组成。

语法: `<!ELEMENT Xcolumn (table*)>Xcolumn 有一个子元素 table。`

table 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>
```

table 元素有一个强制属性和两个隐含属性:

name 属性
指定副表的名称。

key 属性
表的单个主键。

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排序顺序。

table 元素有一个子元素:

column 元素
将 CDATA 节点的属性从输入 XML 文档映射至表中的列。

语法:

```
<!ATTLIST column
                    name CDATA #REQUIRED
                    type  CDATA #IMPLIED
                    path  CDATA #IMPLIED
                    multi_occurrence CDATA #IMPLIED>
```

column 元素有下列属性:

name 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。

type 属性
定义列的数据类型。它可以是任何 SQL 数据类型。

path 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 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。

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 用于指定结果文档的 DOCTYPE。定义显式值。此值是对整个集合中的所有文档提供的。

doctype 有一个子元素:

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)>`

RDB_node

定义 DB2 UDB 表，XML 元素的内容或 XML 属性的值将存储在此表中或从中进行检索。rdb_node 是 element_node、text_node 和 attribute_node 的子元素，并且具有以下子元素:

table 指定存储元素或属性内容的表。

column

指定存储元素或属性内容的列。

condition

对该列指定条件。可选。

element_node

表示 XML 元素。它必须在对集合指定的 DAD 中定义。对于 RDB_node 映射，根 element_node 必须具有 RDB_node，以指定包含用于其自身及其所有子节点的 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。因为元素内容是由 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 还可以有下列子节点:

element_node
表示当前 XML 元素的子元素。

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。它是定义 XML 属性与关系表中列数据之间映射的一个节点。

语法:

attribute_node 必须具有对 name 属性、column 或 RDB_node 子元素的定义。 attribute_node 具有以下属性:

name 属性的名称。

attribute_node 具有下列子元素:

column
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。它是定义 XML 元素内容与关系表中列数据之间映射的一个节点。

语法: 它必须由 column 或 RDB_node 子元素来定义:

column
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 table 和 column。该条件是可选的。

相关概念:

• 第 151 页的『XML 集合的 DAD 文件』

相关任务:

• 第 159 页的『动态覆盖 DAD 文件中的值』

动态覆盖 DAD 文件中的值

过程:

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

此参数是标志 *override* 参数的类型所必需的输入参数（IN）。*overrideType* 参数具有下列值:

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。

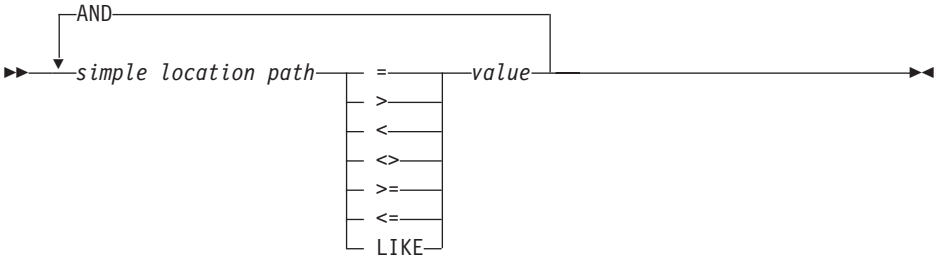
override

此参数是一个可选输入参数（IN），它为 DAD 文件指定覆盖条件。输入值的语法与 *overrideType* 参数上指定的值相对应:

- 如果指定 NO_OVERRIDE，则输入值为 NULL 字符串。
- 如果指定 SQL_OVERRIDE，则输入值为一个有效的 SQL 语句。

  如果使用 SQL_OVERRIDE 作为 SQL 语句，则必须在 DAD 文件中使用 SQL 映射方案。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。
- 如果指定 XML_OVERRIDE，则输入值是包含一个或多个表达式的字符串。

  如果使用 XML_OVERRIDE 和一个表达式，则必须在 DAD 文件中使用 RDB_node 映射方案。输入 XML 表达式将覆盖 DAD 文件中所指定的 RDB_node 条件。表达式使用下列语法:



此语法具有下列组件:

simple location path

使用 XPath 定义的语法来指定简单位置路径。

运算符

语法图中显示的 SQL 运算符可使用空格来将运算符与表达式的其他部分隔开。

运算符两边的空格是可选的。LIKE 运算符两边的空格是必需的。

value
括在单引号中的数值或字符串。

AND
将 AND 视为同一位置路径上的逻辑运算符。如果在覆盖字符串中多次指定简单位置路径，则同时应用该简单位置路径的所有谓词。

如果指定 XML_OVERRIDE，则指定的表达式将覆盖与简单位置路径相匹配的 text_node 或 attribute_node 中的 RDB_node 的条件。

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。

下列示例使用 SQL_OVERRIDE 和 XML_OVERRIDE 来显示动态覆盖。

示例 1: 一个使用 SQL_OVERRIDE 的存储过程。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* 参数覆盖 <SQL_stmt> 的值。

```
#include "dxx.h"
#include "dxxrc.h"

             EXEC SQL INCLUDE SQLCA;
              EXEC SQL BEGIN DECLARE SECTION;
char    collectionName[32];    /* name of an XML collection */
             char    result_tab[32];    /* name of the result table */
char    result_colname[32];    /* name of the result column */
char    valid_colname[32];     /* name of the valid column, will set to NULL*/
char    override[512];      /* override */
         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   collectionName_ind;
             short            rtab_ind;
short   rcol_ind;
short   vcol_ind;
             short            ovtype_ind;
short   ov_ind;
short   maxrow_ind;+
             short            numrow_ind;
             short       returnCode_ind;
             short       returnMsg_ind;
EXEC SQL END DECLARE SECTION;
         float    price_value;

         /* create table */
         EXEC SQL CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

/* initial host variable and indicators */
strcpy(collection, "sales_ord");
         strcpy(result_tab,"xml_order_tab");
strcpy(result_col,"xmlorder");
valid_colname[0] = '\0';

/* get the price_value from some place, such as from data */
price_value = 1000.00        /* for example */

/* specify the override */
sprintf(override,
  " SELECT o.order_key, customer, p.part_key,
    quatity, price, tax, ship_id, date, mode
```

```
        FROM order_tab o, part_tab p,
    table(select db2xml.generate_unique()
    as ship_id, date, mode from ship_tab) 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);

        overrideType = SQL_OVERRIDE;
max_row = 0;
        num_row = 0;
       returnCode = 0;
        msg_txt[0] = '\0';
collectionName_ind = 0;
        rtab_ind = 0;
rcol_ind = 0;
vcol_ind = -1;
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 "DB2XML.DXXRETRIEVEXML" (:collectionName:collectionName_ind,
                :result_tab:rtab_ind,
                :result_colname:rcol_ind,
                :valid_colname:vcol_ind,
                                   :overrideType:ovtype_ind,:override:ov_ind,
                                   :max_row:maxrow_ind,:num_row:numrow_ind,
                              :returnCode:returnCode_ind,
                              :returnMsg:returnMsg_ind);

if (SQLCODE < 0) {
    EXEC SQL ROLLBACK;
  else
    EXEC SQL COMMIT;
}
```

示例 2: 一个使用 XML_OVERRIDE 的存储过程。在此示例中，DAD 文件中的 <collection> 元素具有根 element_node 的 RDB_node。*override* 值基于 XML 内容。XML Extender 将简单位置路径转换为所映射的 DB2 UDB 列。

```
#include "dxx.h"
#include "dxxrc.h"

             EXEC SQL INCLUDE SQLCA;
              EXEC SQL BEGIN DECLARE SECTION;
char    collectionName[32];    /* name of an XML collection */
             char    result_tab[32];    /* name of the result table */
char    result_colname[32];    /* name of the result column */
char    valid_colname[32];     /* name of the valid column, will set to NULL*/
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   collectionName_ind;
             short            rtab_ind;
short   rcol_ind;
short   vcol_ind;
             short            ovtype_ind;
short   ov_ind;
             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);

/* initial host variable and indicators */
strcpy(collection, "sales_ord");
         strcpy(result_tab,"xml_order_tab");
strcpy(result_col,"xmlorder");
valid_colname[0] = '\0';
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';
collectionName_ind = 0;
         rtab_ind = 0;
rcol_ind = 0;
vcol_ind = -1;
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 "DB2XML.DXXRETRIEVE" (:collectionName:collectionName_ind,
                 :result_tab:rtab_ind,
                 :result_colname:rcol_ind,
                 :valid_colname:vcol_ind,
            :overrideType:ovtype_ind,:override:ov_ind,
                 :max_row:maxrow_ind,:num_row:numrow_ind,
                :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
if (SQLCODE < 0) {
    EXEC SQL ROLLBACK;
  else
    EXEC SQL COMMIT;
}
```

多次覆盖

XML Extender 支持相同路径上的多次覆盖。将接受对 RDB 节点指定的所有覆盖。

可以在相同位置路径上指定多次 XML 覆盖以优化搜索中的设置条件。在以下示例中，XML 文档由两个使用 test.dad 文件的表组成。

表 48. 部门表

部门编号	部门名称
10	工程部
20	运营部
30	市场部

表 49. 员工表

员工编号	部门编号	工资
123	10	$98,000.00
456	10	$87,000.00
111	20	$65,000.00
222	20	$71,000.00
333	20	$66,000.00
500	30	$55,000.00

以下列举的 DAD 文件 test.dad 包含将变量 *deptno* 与值 10 进行比较的条件。要将搜索扩展至大于 10 而小于 30，必须覆盖此条件。在调用 dXXGenXML 时，必须如下设置覆盖参数:

```
/ABC.com/Department>10 AND /ABC.com/Department<30

<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "C:\dxx_xml\test\dtd\dad.dtd">
<DAD>
<dtdid>E:\dtd\lineItem.dtd</dtdid>
<validation>NO</validation>   <Xcollection>
<porlog>?xmol version="1.0"?</prolog>
<doctype>!DOCTYPE Order SYSTEM "C:\dxx_xml\test\dtd\LineItem.dtd"</doctype>
 <root_node>
<element_node name="ABC.com">
<TDB_node>
<table name="dept" key="deptno"/>
<table name="empl" key="emplno"/>
<condition>dept deptno=empl.deptno</condition>
          </RDB_node>

<element_node name="Department" multi_occurrence="YES">
          <text_node>
       <RDB_node>
<table name="dept"/>
<column name="deptno">
<condition>deptno=10</condition><RDB_node></RDB_node><text_node></text_node>
<element_node name="Employees" multi_occurrence="YES">

          <text_node>

       <RDB_node>

<table name="dept"><column name="deptnot"><condition>deptno=10</condition>
</table></RDB_node></text_node>
<element_node name="Employees" multi_occurence="YES">

<element_node name="EmployeeNo">

          <text_node>

       <RDB_node>

<table name="empl"><column name="emplno"><condition>emplno<500</condition>
</table></RDB_node></text_node></element_node>
<element_node name="Salary">

          <text_node>

       <RDB_node>
```

```
<table name="empl"><column name="salary"><condition>salary>5000.00</condition>
</table></RDB_node></text_node></element_node></element_node></element_node>
```

要组成不含覆盖的 XML 文档，输入 tests2x mydb test.dad result_tab，也可以在不设置覆盖的情况下调用 dxxGenXML。这将生成与以下内容相似的文档:

```
<?xml version="1.0">
<!DOCTYPE Order SYSTEM "C:\dxx_xml\test\dtd\LineItem.dtd">
<ABC.com>
<Department>10
<Employees>
<EmployeeNo>123</EmployeeNO>
<Salary>98,000.00</Salary>
</Employees>
<Employees>
<EmployeeNo>456</EmployeeNo>
<Salray>87,000.00</Salary>
</Employees>
</Department>
</ABC.COM>
```

要覆盖 DAD 文件，可以按上述方式调用 dxxGenXML，也可以在指定条件下运行 test2x 程序:

```
tests2x mydb test.dad result_tab -o 2 ″/ABC.com/Department>10 AND
/ABC.com/Department<30″
<?xml version="1.0">
<!DOCTYPE Order SYSTEM "C:\dxx_xml\test\dtd\LineItem.dtd">
<ABC.com>
<Department>20
<Employees>
<EmployeeNo>111</EmployeeNo>
<Salary>65,000.00</Salary>
</Employees>
<EmployeeNo>222</EmployeeNo>
<Salary>71,000.00</Salary>
</Employees>
<Employees>
<EmployeeNo>333</EmployeeNo>
<Salary>66,000.00</Salary>
</Employees>
</Department>
</ABC.com>
```

相关概念:

- 第 151 页的『XML 集合的 DAD 文件』
- 第 165 页的『Dad 检查程序』

相关任务:

- 第 149 页的『为 XML 列创建 DAD 文件』
- 第 165 页的『使用 DAD 检查程序』

相关参考:

- 第 154 页的『用于 DAD 文件的 DTD』

Dad 检查程序

DAD 检查程序可以用来验证使用 XML 集合存储方法的 DAD 文件的有效性。在每个 DAD 文件中，指定了映射方案，此方案用于指定表之间的关系以及 XML 文档的结构。

与用来验证 XML 文档语法的“文档类型描述”（DTD）很相似，使用 DAD 检查程序来确保 DAD 文件在语义上是正确的。可以在不连接数据库的情况下执行此项验证。DAD 检查程序的使用有助于最大程度地减少在将文件提交至 XML Extender 以进行处理时发生的错误。DAD 检查程序是一个从命令行调用的 Java™ 应用程序。在调用时，它生成两个输出文件，这两个文件包含错误、警告和成功指示符。这两个文件是等效的；其中一个是用来检查错误或警告的纯文本文件；另一个是 XML 文件 `errorsOutput.xml`，它将 DAD 检查程序应用程序的结果传达给其他应用程序。输出文本文件的名称是用户定义的。如果没有指定名称，则使用标准输出。

相关概念:

- 第 151 页的『XML 集合的 DAD 文件』

相关任务:

- 第 159 页的『动态覆盖 DAD 文件中的值』
- 第 149 页的『为 XML 列创建 DAD 文件』
- 第 165 页的『使用 DAD 检查程序』

使用 DAD 检查程序

先决条件:

系统上必须已安装 JRE 或 JDK 版本 1.3.1 或更新版本。

过程:

要使用 DAD 检查程序:

1. 下载 DADChecker.zip 文件，并将所有文件抽取到您选择的目录中。
2. 从命令行，转至安装 DAD 检查程序的目录中的 `/bin` 子目录。
3. 通过运行位于 `/bin` 目录的 setCP.bat 文件来设置类路径。
4. 运行以下命令:

   ```
   java dadchecker.Check_dad_xml [-dad | -xml] [-all][-tag tagname]
   [-out outputFile] fileToCheck
   ```

   其中:

   -dad
   指示要检查的文件是 DAD 文件。这是缺省选项。

   -xml
   指示要检查的文件是 XML 文档而不是 DAD 文件。对于大型 XML 文档，Java 虚拟机可能会耗尽内存，这将生成 java.lang。OutOfMemoryError 例外。在这种情况下，可使用 -Xmx 选项来分配更多的内存给“Java 虚拟机”。有关详情，请参阅 JDK 文档。

-all
指示输出将显示所有出错标记的出现。

-tag
指示只显示其名称属性值为 *tagname* 的重复标记。对于 XML 文档，只显示其名称为 tagname 的重复标记。

-out
outputFile 指定输出文本文件名。如果省略此项，则使用标准输出。还会在 DAD 文件所在相同目录中创建第二个输出文件 `errorsOutput.xml`。始终生成此文件，它包含输出文本文件中的那些信息（解析器警告和错误除外）的 XML 格式。

要显示命令行选项，请输入 `java dadchecker.Check_dad_xml help`。

要显示版本信息，请输入 `java dadchecker.Check_dad_xml version`。

Dad 检查程序的样本文件:

可以在 `samples` 目录中找到下列样本文件:

bad_dad.dad
用于演示所有可能的语义错误的样本 DAD 文件。

bad_dad.chk
DAD 检查程序对 bad_dad.dad 生成的输出文本文件。

bad_dad.chk
DAD 检查程序对 bad_dad.dad 生成的输出文本文件。

errorsOutput.xml
DAD 检查程序对 bad_dad.dad 生成的输出 XML 文件。

dup.xsl
用于将 errorsOutput.xml 文件转换成只显示重复标记的 HTML 文件的 XSL 样式表。

dups.html
生成的 HTML 文件，它只显示 bad_dad.dad 中包含的重复标记。

输出文本文件中的错误和警告:

错误和警告由标记出现指示。在下列情况下，将两个标记视为同一标记的两次出现:

- 它们的 name 属性具有相同的值。
- 它们具有相同数目个祖先。
- 它们的对应祖先标记的 name 属性具有相同的值。

同一标记的各次出现可能潜在地具有不同的子标记。

在输出文本文件中，采用以下方法来指示不符合 DAD 语义规则的标记出现:

- 按顺序显示所有祖先标记及其属性。
- 显示出错标记，由指示该标记在 XML 树中的深度的数字引出。标记名后面是行号列表，该标记出现在 DAD 文件中的那些行上。通过使用 *-all* 命令行选项，可单独地显示每个错误出现。
- 显示第一个标记出现的直接子标记。对于那些指定了数据映射的子标记，还显示数据映射标记。可使用 *-all* 命令行选项来单独地显示每个错误出现。

DAD 检查程序错误报告的样本:

在此示例中，name 属性具有 “Password” 值的 element_node 标记有错误。在 DAD 文件中，此标记出现了两次，一次出现在第 49 行上，另一次出现在第 75 行上。通过定位标记的深度指示符（在本示例中是 4），可以将出错标记与祖先及子标记列表隔离开。祖先及子标记的列表有助于建立发生错误所在的上下文。

```
<DAD>
<Xcollection>
 <root_node>
      <element_node name="Advertiser" multi_occurrence="YES">
4       <element_node name="Password">  line(s): 49 75
        <element_node name="Pswd1">
     <element_node name="Pswd2">
```

如果已使用了 all 选项，则输出文本文件将类似于:

```
<DAD>
<Xcollection>
 <root_node>
      <element_node name="Advertiser" multi_occurrence="YES">
4         <element_node name="Password">  line: 49
        <element_node name="Pswd1">
        <element_node name="Pswd2">


<DAD>
<Xcollection>
 <root_node>
      <element_node name="Advertiser" multi_occurrence="YES">
4      <element_node name="Password">  line: 75
        <element_node name="Pswd1">
    <element_node name="Pswd3">
```

在本示例中，两次出现具有完全相同的祖先和 name 属性值，但具有不同的子元素。

DAD 检查程序执行的检查

在调用 DAD 检查程序时，您将接收到以下消息:

```
Checking DAD document: file_path
```

其中，*file_path* 是正在验证的 DAD 文件的路径。

DAD 检查程序执行下列验证检查:

1. 优良结构检查和 DTD 验证。
2. 检测 <attribute_node> 和叶 <element_node> 是否重复（RDB_node 映射）。
3. 检测是否遗漏了 type 属性。
4. 检测是否遗漏了表声明。
5. 检测是否遗漏了 <text_node> 或 <attribute_node>。
6. 检查 <attribute_node> 和 <element_node> 映射顺序。
7. 对具有完全相同的 name 属性值的标记执行的数据映射一致性检查。
8. 对带有映射子代的父 <element_node> 执行的 multi_occurrence 属性值检查（RDB_node 映射）。
9. 属性和元素潜在命名冲突检查（XML 文档）。

下列各节对这些验证检查作了描述。

优良结构和 DTD 验证

必须针对 DAD DTD 验证 DAD 文件，DAD DTD 位于 ″dxxsamples\dtd\dad.dtd″ 中如果 DAD 文件不具有优良的结构，或者找不到 DTD，则会发生致命错误，此错误导致 DAD 检查程序终止，并在输出文本文件中指示此错误。例如:

```
org.xml.sax.SAXException: Stopping after fatal error,
line 1, col 22. The XML declaration must end with "?>".
```

还会在输出文本文件中报告验证错误和警告，但它们不会导致 DAD 检查程序终止。以下示例是输出文本文件的一个片段，它显示了分析 DAD 文件时可能会遇到的两个可能验证错误:

```
 ** The document is not valid against the DTD, line 5, col 15. Element type
     "XCollection" must be declared

** The document is not valid against the DTD, line 578, col 21. The content of
      element type "text_node" must match "(column|RDB_node)".
```

检测 <attribute_node> 和叶 <element_node> 是否重复（RDB_node 映射）

此项检查只与使用 RDB_node 映射的 DAD 文件相关。

如果两个或多个 <attribute_node> 或 <element_node> 标记在它们的名称属性中具有相同的值，而且有相同的祖先，则认为两种元素是重复的。

如果两个或更多个标记的对应祖先标记的 name 属性具有相同的值，则将这些标记视为具有相同的祖先。

叶 <element_node> 是一个 element_node，它用来映射在 XML 文档树中没有子代的标记。因此，叶 <element_node> 标记必须有一个文本节点标记作为它的其中一个直接子代。没有任何其他 <element_node> 标记可以将文本节点标记作为直接子代。

此项冲突可能会发生在两个或多个叶 <element_node> 标记之间，两个或多个 <attribute_node> 标记之间，或者叶 <element_node> 标记和 <attribute_node> 标记之间。

示例:

示例 1:

叶 <element_node> 冲突:

```
<element_node name = "A1">
    <element_node name = "B">
       <element_node name = "C">
       <text_node
          ....
<element_node name = "A2">
    <element_node name = "B">
       <element_node name = "C">
                 <text_node>
       ....
              </element_node>
```

在本示例中，由于通过两个不同的路径（Å1\B\C 和 \A2\B\C）映射 <element_node name = ″C″>，所以它是重复的。注意，由于 <element_node name=″B″> 不是叶 <element_node>，所以不认为它是重复的。

示例 2:

本示例显示 <attribute node> 冲突。

```
<element_node name = "A1">
<attribute_node name = "B">
                  ....
<element_node name = "A2">
<attribute_node name = "B">
 /element_node>               ....
<
```

在本示例中，由于通过两个不同的路径（Å1\B 和 \A2\B）映射 <attribute_node name = ″B″>，所以它是重复的。

示例 3:

本示例显示叶 <element_node> 和 <attribute_node> 冲突。

```
<element_node name = "A">
    <element_node name = "B">
                  <text_node>
             ....
               </element_node>
               </element_node>
            ....
<attribute_node name = "B">
            ....
<attribute_node name = "A">
            ....
```

在本示例中，<element_node name = ″B″> 与 <attribute_node name = ″B″> 冲突。注意，由于 <element_node name = ″A″> 不是叶 <element_node>，所以 <element_node name = ″A″> 与 <attribute_node name = ″A″> 不冲突。

如果发生冲突，则必须修订 XML 文档 DTD 来消除冲突。还需要修订 XML 文档和 DAD 文件来反映 DTD 更改。

示例 4:

```
7 duplicate naming conflicts were found
A total of 16 tags are in error (cumulate occurrences of these tags: 20)

The following tags are duplicates:

<DAD>
  <Xcollection>
 <root_node>
      <element_node name="Advertiser" multi_occurrence="YES">
4       <element_node name="Country">  line(s): 127 135
                <text_node>
            <RDB_node>
             <table name="advertiser">
               <column type="VARCHAR(63)" name="country">

------------
<DAD>
<Xcollection>
 <root_node>
      <element_node name="Advertiser" multi_occurrence="YES">
     <element_node name="Campaign" multi_occurrence="YES">
      <element_node name="Target" multi_occurrence="YES">
       <element_node name="Location" multi_occurrence="YES">
7         <element_node name="Country">  line(s): 460
                <text_node>
                   <RDB_node>
```

```
              <table name="target_location">
              <column type="VARCHAR(63)" name="country">
-----------------------------------------------------------
```

出错的标记按命名冲突分组。各个组由线分隔，各个标记由短线分隔。通过使用 *all* 命令行选项，还可显示所有错误出现。

如果 DAD 文件中没有重复，则将以下消息写入输出文本文件:

```
No duplicated tags were found.
```

检测是否遗漏了 type 属性

当使用 DAD 文件来启用集合或用于分解时，必须对每个 <column> 标记指定 type 属性。例如:

```
<column name="email" type="varchar(20)">
```

enable_collection 命令使用列类型规范来在集合中创建表（如果那些表不存在的话）。如果那些表存在，则 DAD 中指定的类型必须与数据库中的实际列类型相匹配。

示例:

以下示例是输出文本文件的一个片段，它显示了不带 type 属性的 <column> 标记:

```
If this DAD is to be used for decomposition or for enabling a collection,
the type attributes are missing for the following <column> tag(s):
<DAD>
  <Xcollection>
 <root_node>
      <element_node name="Advertiser" multi_occurrence="YES">
      <element_node name="Address">
                  <text_node>
            <RDB_node>
7           <column name="address">  line: 86
```

如果没有遗漏 type 属性，则将以下消息写入输出文本文件:

```
No type attributes are missing for <column> tags.
```

检测是否遗漏了表声明

DAD 文件中的第一个 <RDB_node> 标记必须将表声明括起来，这包括所有声明用于数据映射的关系表的 <table> 标记。必须将此标记括在第一个 <element_node> 标记中。必须将所有后续 <RDB_node> 标记都括在 <text_node> 标记中。

如果遇到的第一个 <RDB_node> 标记包含 <column> 标记，则也会将错误添加至输出文件。此错误指示遗漏表声明，或者表声明错误地包含 <column> 标记。

检测是否遗漏了 <text_node> 或 <attribute_node>

每个 <RDB_node> 标记（第一个除外，它用于表声明）都必须括在 <attribute_node> 或 <text_node> 标记中。

示例:

示例 1:

```
 <element_node name ="amount">
   <text_node>
            <RDB_node>
```

```
            <table name="fakebank.payments"/>
           <column name="amount" type="decimal(8,2)"/>
                    </RDB_node>
              </element_node>
```

示例 2:

以下示例是输出文本文件的一个片段，它显示了遗漏 <text_node> 或 <attribute_node> 标记:

```
<DAD>
  <Xcollection>
 <root_node>
      <element_node name="Advertiser" multi_occurrence="YES">
      <element_node name="PostalCode">
5         <RDB_node>  line: 107
             <table name="advertiser">
        <column type="VARCHAR(10)" name="postal_code">
```

检查 <attribute_node> 和 <element_node> 映射顺序

此项检查是“修订包 3”或更旧版本所必需的。在将任何 <element_node> 标记映射至表之前，需要将 <attribute_node> 标记映射至表。

示例:

以下示例显示了需要映射至表的标记。

```
<element_node name="payment-request"
multi_occurrence="YES">
  <element_node name="payment-request-id">
                  <text_node>
            <RDB_node>
            <table name="fakebank.payments"/>
            <column name="statement_id" type="varchar(30)"/>
                  ....
<element_node name="bank-customer-info">
    <element_node name="account">
          <attribute_node name="type">
                 <text_node>
              <RDB_node>
              <table name="fakebank.payments"/>
                 <column name="payor_account" type="char(6)"/
```

在本示例中，<attribute_node name=″type″> 与 <element_node name = ″payment-request-id″> 映射至同一个表（fakebank.payments）。<attribute_node> 的映射必须位于 <element_node> 的映射之前。

对具有完全相同的 name 属性值的标记执行的数据映射一致性检查

在 DAD 文件中，所有的 <element_node> 标记以及映射的（并由相异 name 属性值标识的）所有 <attribute_node> 标记都只应映射一次。如果 <element_node> 标记或 <attribute_node> 标记的两次或更多次出现映射至不同的列，则应对它们的 name 属性指定不同的值。

示例:

示例 1: 在本示例中，<element_node name=″type″> 标记的第二次出现与第一次出现具有不同的映射。此项检查的结果并不显示重复的 <attribute_node> 和重复的叶 <element_node> 标记。

```
<element_node name="bank-customer-info">
    <element_node name="account">
      <element_node name="type">
                 <text_node>
              <RDB_node>
              <table name="fakebank.payments"/>
              <column name="payor_account" type="char(20)
            </RDB_node>
                 </text_node>
              </element_node>
    <element_node>
<element_node>
<element_node name="bank-customer-info">
    <element_node name="account">
      <element_node name="type">
                 <text_node>
              <RDB_node>
              <table name="fakebank.payments"/>
              <column name="payto_account" type="char(20)"/>
            </RDB_node>
                 </text_node>
              </element_node>
              </element_node>
<element_node>
```

可以通过创建新元素来与第二个映射配合使用来修正此错误。还需要更改 DTD、XML 文档和 DAD 文件。

示例 2: 本示例是输出文本文件的一个片段，它指示具有相同名称和祖先，但不具有相同映射的 <element_node> 标记。

```
<DAD>
  <Xcollection>
 <root_node>
      <element_node name="Advertiser" multi_occurrence="YES">
4      <element_node name="PostalCode">  line(s): 127
                 <text_node>
               <RDB_node>
             <table name="advertiser">
        <column type="VARCHAR(10)" name="postal_code">
----------------
<DAD>
  <Xcollection>
   <root_node>
      <element_node name="Advertiser" multi_occurrence="YES">
4         <element_node name="PostalCode">  line(s): 135 143
                 <text_node>
              <RDB_node>
             <table name="advertiser">
              <column type="VARCHAR(10)" name="postal_code2">
```

在本示例中，<element_node name=″PostalCode″> 在第 127 行上的一次出现映射至 “postal_code” 列，而同一标记的另外两次出现（分别在第 135 和 143 行上）映射至 “postal_code2” 列。

对带有映射子代的父 <element_node> 执行的 multi_occurrence 属性值检查

此项检查只与使用 RDB_node 映射的 DAD 文件相关。

multi_occurrence 属性的缺省值是 NO。对于每个带有直接子代 <attribute_node> 标记的 <element_node> 标记，或者两个或更多个符合以下两个条件之一或两者的 <element_node> 标记，应对 multi_occurrence 属性分配 YES 值:

- 映射 <element_node>（它将 <text_node> 作为其直接子代）。
- <element_node> 将至少一个 <attribute_node> 作为直接子代。

示例:

示例 1: 在以下示例中，payment-request-id 和 amount 映射至 DB2 UDB 表。Sender 将一个 <attribute_node> 作为直接子代。Payment-request-id、amount 和 sender 全都是 payment-request 的直接子代:

```
<element_node name="payment-request" multi_occurrence="YES">
  <element_node name="payment-request-id">
                <text_node>
              <RDB_node>
              <table name="fakebank.payments"/>
            <column name="statement_id" type="varchar(30)"/>
            </RDB_node>
                 </text_node>
              </element_node>
 <element_node name ="amount">
                <text_node>
              <RDB_node>
              <table name="fakebank.payments"/>
           <column name="amount" type="decimal(8,2)"/>
            </RDB_node>
                 </text_node>
              </element_node>
  <element_node name ="sender">
     <attribute_node name ="ID">
              <RDB_node>
              <table name="fakebank.payments"/>
            <column name="sender_ID" type="decimal(8,2)"/>
            </RDB_node>
              </attribute_node>
              </element_node>
              </element_node>
```

DAD 检查程序将指示 multi_occurrence 属性设置为 NO 的所有 <element_node> 标记。

示例 2: 以下示例是输出文本文件的一个片段，它给出哪些 <element_node> 标记的 multi_occurrence 属性应设置为 YES 的建议。

```
<DAD>
<Xcollection>
 <root_node>
      <element_node name="Advertiser" multi_occurrence="YES">
4       <element_node name="Password">  line(s): 49 75
        <element_node name="Pswd1">
        <element_node name="Pswd2">
```

属性和元素命名冲突

在 XML 文档中，同名元素可能出现在不同的上下文中，如具有不同祖先元素。属性和元素可以具有完全相同的名称。由于这些命名冲突通常会导致 DAD 文件中出现重复标记，所以 XML Extender 当前无法解决这些命名冲突。因此，所有属性以及将要映射的所有具有相同祖先的元素都必须具有唯一的名称。

DAD 检查程序可用来检查 XML 文档是否存在命名冲突。如果需要映射多个相冲突的元素或属性，则应对文档和 DTD 进行命名更改。

最好在创建 DAD 文件之前检查 XML 文档。DAD 检查程序不会针对 XML 文档的 DTD 来对其进行验证。

示例:

以下示例是 XML 文档的一个片段，该处发生了命名冲突:

```
<A1>
   <B>
      <C>
          ....
<A2>
   <B>
      <C>
          ....
<D C="attValue">
.....
```

如果将要映射 <C> 元素和 C 属性，则生成的 DAD 文件将带有下列重复冲突:

```
<element_node name = "A1">
    <element_node name = "B">
       <element_node name = "C">
                 <text_node>
            .....
<element_node name = "A2">
    <element_node name = "B">
       <element_node name = "C">
                 <text_node>
            ....
      <element_node name = "D">
         <attribute_node name = "C">
          ....
              </element_node>
```

在 DAD 中，两个 <element_node name = ″C″> 标记与 <attribute_node name = ″C″> 标记重复。

第 10 章 XML Extender 存储过程

XML Extender 存储过程

XML Extender 提供了存储过程（又称*过程*）来对 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细信息，请参阅《*DB2 UDB iSeries 版 SQL 编程*》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:

管理存储过程
帮助用户完成管理任务

组合存储过程
使用现有数据库表中的数据生成 XML 文档

分解存储过程
拆散或分割入局 XML 文档，并将数据存储在新的或现有数据库表中

在调用存储过程之前，必须遵循所有环境的设置过程。调用存储过程的样本程序位于 DXXSAMPLES/QCSRC 中。

确保将 XML Extender 外部头文件包括在需要调用存储过程的程序中。头文件位于 ″dxxsamples\include″ 目录中。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

用于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。

XML Extender 管理存储过程

这些存储过程用于诸如启用或禁用 XML 列或集合之类的管理任务。它们由 XML Extender 管理向导和管理命令 **dxxadm** 调用。

- dxxEnableDB()
- dxxDisableDB()
- dxxEnableColumn()
- dxxDisableColumn()
- dxxEnableCollection()
- dxxDisableCollection()

dxxEnableDB() 存储过程

用途:

启用数据库。启用数据库时，XML Extender 将创建下列对象:

- XML Extender 用户定义的类型（UDT）。
- XML Extender 用户定义的函数（UDF）。
- XML Extender DTD 资源库表 DTD_REF，它用来存储 DTD 和有关每个 DTD 的信息。
- XML Extender 用法表 XML_USAGE，它用来存储对 XML 启用的每个列以及每个集合的公共信息。

语法:

```
QDB2XML.dxxEnableDB(char(dbName) dbName,       /* input */
            long          returnCode,     /* output */
            varchar(1024) returnMsg)      /* output */
```

参数:

表 50. 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消息文本。	OUT

相关概念:

- 第 197 页的第 11 章，『XML Extender 管理支持表』

相关任务:

- 第 48 页的『对 XML 启用数据库』
- 第 180 页的『调用 XML Extender 组合存储过程』

相关参考:

- 第 175 页的『XML Extender 管理存储过程』
- 第 xi 页的『如何阅读语法图』
- 第 229 页的附录 C，『XML Extender 限制』

dxxDisableDB() 存储过程

用途:

禁用数据库。当 XML Extender 禁用数据库时，它将删除下列对象:

- XML Extender 用户定义的类型（UDT）。
- XML Extender 用户定义的函数（UDF）。
- XML Extender DTD 资源库表 DTD_REF，它用来存储 DTD 和有关每个 DTD 的信息。

- XML Extender 用法表 XML_USAGE，它用来存储对 XML 启用的每个列以及每个集合的公共信息。

重要事项: 在尝试禁用数据库之前，必须禁用所有 XML 列。XML Extender 不能禁用其中包含已对 XML 启用的列或集合的数据库。

语法:

```
QDB2XML.dxxDisableDB(char(dbName)     dbName,       /* input */
            long          returnCode,     /* output */
            varchar(1024) returnMsg)      /* output */
```

参数:

表 51. 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消息文本。	OUT

相关概念:

- 第 197 页的第 11 章，『XML Extender 管理支持表』

相关任务:

- 第 180 页的『调用 XML Extender 组合存储过程』

相关参考:

- 第 175 页的『XML Extender 管理存储过程』
- 第 xi 页的『如何阅读语法图』
- 第 229 页的附录 C，『XML Extender 限制』

dxxEnableColumn() 存储过程

用途:

启用 XML 列。当启用列时，XML Extender 完成下列任务:

- 确定 XML 表是否具有主键；如果没有，则 XML Extender 将变更 XML 表，并添加名为 DXXROOT_ID 的列。
- 创建 DAD 文件中指定的副表，并且有一列包含 XML 表中每一行的唯一标识。此列是由用户指定的 root_id，或是由 XML Extender 命名的 DXXROOT_ID。
- 为 XML 表及其副表创建缺省视图，可选择使用您指定的名称。

语法:

```
DB2XML.dxxEnableColumn(char(dbName) dbName,      /* input */
                char(tbName) tbName,      /* input */
                char(colName) colName,     /* input */
                    CLOB(100K)     DAD,         /* input */
                                char(defaultView) defaultView, /* input */
                char(rootID) rootID,      /* input */
            long          returnCode,     /* output */
            varchar(1024) returnMsg)      /* output */
```

参数:

表 52. 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
defaultView	用来连接应用程序表和副表的缺省视图的名称。	IN
rootID	应用程序表中要用作副表的 root 用户标识的单个主键的名称。	IN
returnCode	存储过程的返回码。	OUT
returnMsg	在发生错误时返回的消息文本。	OUT

相关概念:

- 第 67 页的『XML 列作为存储和访问方法』

相关任务:

- 第 180 页的『调用 XML Extender 组合存储过程』

相关参考:

- 第 175 页的『XML Extender 管理存储过程』
- 第 xi 页的『如何阅读语法图』
- 第 229 页的附录 C, 『XML Extender 限制』

dxxDisableColumn() 存储过程

用途:

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。

语法:

```
DB2XML.dxxDisableColumn(char(dbName) dbName,      /* input */
                char(tbName) tbName,      /* input */
                 char(colName) colName,     /* input */
            long          returnCode,     /* output */
            varchar(1024) returnMsg)      /* output */
```

参数:

表 53. 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消息文本。	OUT

相关参考:

- 第 229 页的附录 C, 『XML Extender 限制』

dxxEnableCollection() 存储过程

用途:

启用与应用程序表相关联的 XML 集合。

语法:

```
dxxEnableCollection(char(dbName) dbName,      /* input */
                    char(colName) colName,     /* input */
                    CLOB(100K)     DAD,         /* input */
                      long           returnCode,  /* output */
            varchar(1024) returnMsg)      /* output */
```

参数:

表 54. 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
returnCode	存储过程的返回码。	OUT
returnMsg	在发生错误时返回的消息文本。	OUT

相关概念:

- 第 81 页的『XML 集合作为存储和访问方法』

相关任务:

- 第 180 页的『调用 XML Extender 组合存储过程』

相关参考:

- 第 175 页的『XML Extender 管理存储过程』
- 第 xi 页的『如何阅读语法图』
- 第 229 页的附录 C, 『XML Extender 限制』

dxxDisableCollection() 存储过程

用途:

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。

语法:

```
dxxDisableCollection(char(dbName) dbName,      /* input */
                     char(colName) colName,     /* input */
            long          returnCode,     /* output */
            varchar(1024) returnMsg)      /* output */
```

参数:

表 55. 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消息文本。	OUT

相关参考:

- 第 229 页的附录 C, 『XML Extender 限制』

XML Extender 组合存储过程

组合存储过程 dxxGenXML()、dxxRetrieveXML()、dxxGenXMLCLOB() 和 dxxRetrieveXMLCLOB() 用于通过使用现有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。

已对组成存储过程作了下列性能改进。

- 在 iSeries 和 zSeries 操作系统上，覆盖参数的长度已经增加到 16KB。在 iSeries 和 zSeries 操作系统上，覆盖参数的长度已经增加到 16KB。
- 已除去中间结果表需求。
- 通过使用这些存储过程:
  - 由于不需要创建结果表，所以缩短了指令路径长度。
  - 简化了编程。
- 如果要生成多个文档，则使用需要中间结果表的存储过程。
- 已增强了 XML 列用户定义的函数的性能
- 在处理 XML 文档时，DB2 UDB XML Extender 用户定义的函数现在将把小型（512KB）的 XML 文档保存在内存中。这将减少输入／输出活动和对用于临时文件的磁盘的争用。
- 已更改 DB2 UDB XML Extender 标量（非表）用户定义的函数的定义，因此它们能够并行运行。对于多次引用用户定义的函数的查询的执行，此项更改显著提高了性能。必须运行迁移脚本程序才能获得标量 UDF 的并行能力。如果已使用标量 UDF 启用了列，则必须禁用所有的列，运行迁移脚本，然后重新启用列。

调用 XML Extender 组合存储过程

通过以大写和小写两种方式编写存储过程名称，可以从单一客户机应用程序使用不同操作系统中的 XML Extender。要以这种方式调用存储过程，请使用组合存储过程的 `result_colname` 和 `valid_colname` 版本。使用这种方法有下列好处:

- 由于可以将许多列包括在结果表中，所以可以在所有 “DB2 通用数据库” 环境中使用这些存储过程。不支持 `result_colname` 和 `valid_colname` 的存储过程版本要求结果表中刚好有一列。

- 可使用已声明临时表来作为结果表。临时表由已设置为“会话”的模式标识。已声明临时表使您能够支持多用户客户机环境。

在调用 DB2 XML Extender 存储过程时使用大写，以便在各平台之间一致地访问存储过程。

过程:

使用下列语法调用 XML Extender:

```
CALL DB2XML.function_entry_point
```

其中:

function_entry_point
指定函数的名称。

在 CALL 语句中，传送给存储过程的自变量必须是主变量，而不能是常量或表达式。主变量可以带有空指示符。

请参阅 DXXSAMPLES/QCSRC 源文件、 中的调用存储过程样本。在 DXXSAMPLES/QCSRC 源目录 中，提供 SQX 代码文件来使用嵌入式 SQL 调用 XML 集合存储过程。

dxxGenXML() 存储过程

用途:

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，并访存结果集。

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* 用来区分 *override* 的类型。

语法:

```
dxxGenXML(CLOB(100K)     DAD,       /* input */
              char(resultTabName) resultTabName,    /* input */

          char(resultColumn) result_column,
         char(validColumn) valid_column,
          integer        overrideType  /* input */
   varchar(varchar_value) override,

           integer        maxRows,      /* input */
               integer        numRows,          /* output */
                 long           returnCode,  /* output */
               varchar(1024)  returnMsg)        /* output */
```

其中，varchar_value 对于 Windows 和 UNIX 是 32672，对于 iSeries 和 z/OS 是 16366。

参数:

表 56. 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
result_column	结果表中用于存储已组成 XML 文档的列的名称。	IN
valid_column	一个列的名称，该列用于指示 XML 文档在对文档类型定义（DTD）进行验证时是否有效。	IN
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
resultDoc	包含已组成 XML 文档的 CLOB。	OUT
valid	valid 的设置如下: • 如果 VALIDATION=YES，则 valid=1 表示验证成功，valid=0 表示验证不成功。 • 如果 VALIDATION=NO，则 valid=NULL。	OUT
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消息文本。	OUT

示例:

下例的片断假定: 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。完整的工作样本位于 `DXXSAMPLES/QCSRC(GENX)` 中。

```
#include "dxx.h"
#include "dxxrc.h"

              EXEC SQL INCLUDE SQLCA;
               EXEC SQL BEGIN DECLARE SECTION;
     SQL TYPE is CLOB(100K) dad;              /* DAD */

              char    result_tab[32];    /* name of the result table */
char    result_colname[32];    /* name of the result column */
char    valid_colname[32];     /* name of the valid column, will set to NULL */
              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   rcol_ind;
short   vcol_ind;
              short            ovtype_ind;
short   ov_ind;
              short            maxrow_ind;
              short            numrow_ind;
              short       returnCode_ind;
              short       returnMsg_ind;
               EXEC SQL END DECLARE SECTION;

FILE   *file_handle;
long    file_length=0;

         /* create table */
         EXEC SQL CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

/* initialize the DAD CLOB object. */
file_handle = fopen( "/dxx/dad/getstart_xcollection.dad", "r" );
if ( file_handle != NULL ) {  file_length = fread ((void *) &dad.data
, 1, FILE_SIZE, file_handle);
  if (file_length == 0) {
    printf ("Error reading dad file
             /dxx/dad/getstart_xcollection.dad\n");
      rc = -1;
      goto exit;
 } else
    dad.length = file_length;
}
else {
  printf("Error opening dad file  \n", );
      rc = -1;
      goto exit;
}
        /* initialize host variable and indicators */
         strcpy(result_tab,"xml_order_tab");
strcpy(result_colname, "xmlorder")
valid_colname = '\0';
              override[0] = '\0';
              overrideType = NO_OVERRIDE;
         max_row = 500;
         num_row = 0;
        returnCode = 0;
         msg_txt[0] = '\0';
              dad_ind = 0;
         rtab_ind = 0;
rcol_ind = 0;
vcol_ind = -1;
              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 "DB2XML.DXXGENXML" (:dad:dad_ind;
                :result_tab:rtab_ind,
                :result_colname:rcol_ind,
                :valid_colname:vcol_ind,
                :overrideType:ovtype_ind,:override:ov_ind,
                :max_row:maxrow_ind,:num_row:numrow_ind,
                             :returnCode:returnCode_ind,
                             :returnMsg:returnMsg_ind);

if (SQLCODE < 0) {
    EXEC SQL ROLLBACK;
  else
    EXEC SQL COMMIT;
}

exit:
  return rc;
```

相关任务:

- 第 55 页的『通过使用 SQL 映射来组成 XML 文档』
- 第 58 页的『通过使用 RDB_node 映射来组成 XML 集合』
- 第 180 页的『调用 XML Extender 组合存储过程』

相关参考:

- 第 180 页的『XML Extender 组合存储过程』
- 第 xi 页的『如何阅读语法图』

---

dxxRetrieveXML() 存储过程

用途:

存储过程 dxxRetrieveXML() 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、错误代码和消息文本。

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* 用来区分 *override* 的类型。

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。

语法:

```
dxxRetrieveXML(char(collectionName) collectionName,   /* input */
               char(resultTabName) resultTabName,    /* input */

              char(resultColumn) result_column,
       char(validColumn) valid_column,
               integer        overrideType,    /* input */
     varchar(varchar_value) override,
```

```
            integer        maxRows,          /* input */
integer        numRows,          /* output */
  long           returnCode,  /* output */
varchar(1024)  returnMsg)        /* output */
```

其中，*varchar_value* 对于 Windows 和 UNIX 是 32672，对于 iSeries 和 z/OS 是 16366。

参数:

表 57. 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
result_column	结果表中用于存储已组成 XML 文档的列的名称。	IN
valid_column	一个列的名称，该列用于指示 XML 文档在对文档类型定义（DTD）进行验证时是否有效。	IN
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消息文本。	OUT

示例:

以下的片断是调用 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。完整的工作样本位于DXXSAMPLES/QCSRC(RTRX) 中。

```
#include "dxx.h"
#include "dxxrc.h"

              EXEC SQL INCLUDE SQLCA;
               EXEC SQL BEGIN DECLARE SECTION;
char    collectionName[32];    /* name of an XML collection */
              char    result_tab[32];    /* name of the result table */
char    result_colname[32];    /* name of the result column */
char    valid_colname[32];     /* name of the valid column, will set to NULL*/
              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   collectionName_ind;
              short            rtab_ind;
short   rcol_ind;
short   vcol_ind;
              short            ovtype_ind;
short   ov_ind;
              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);

/* initial host variable and indicators */
strcpy(collection, "sales_ord");
         strcpy(result_tab,"xml_order_tab");
strcpy(result_col,"xmlorder");
valid_colname[0] = '\0';
              override[0] = '\0';
              overrideType = NO_OVERRIDE;
         max_row = 500;
         num_row = 0;
        returnCode = 0;
         msg_txt[0] = '\0';
collectionName_ind = 0;
         rtab_ind = 0;
rcol_ind = 0;
vcol_ind = -1;
              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 "DB2XML.DXXRETRIEVE" (:collectionName:collectionName_ind,
                 :result_tab:rtab_ind,
                 :result_colname:rcol_ind,
                 :valid_colname:vcol_ind,
                 :overrideType:ovtype_ind,:override:ov_ind,
                 :max_row:maxrow_ind,:num_row:numrow_ind,
          :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
if (SQLCODE < 0) {
```

```
    EXEC SQL ROLLBACK;
  else
    EXEC SQL COMMIT;
}
```

相关任务:

- 第 55 页的『通过使用 SQL 映射来组成 XML 文档』
- 第 58 页的『通过使用 RDB_node 映射来组成 XML 集合』
- 第 180 页的『调用 XML Extender 组合存储过程』

相关参考:

- 第 180 页的『XML Extender 组合存储过程』
- 第 xi 页的『如何阅读语法图』
- 第 229 页的附录 C, 『XML Extender 限制』

dxxGenXMLClob 存储过程

用途:

dxxGenXMLClob 采用包含 DAD 的缓冲区来作为输入。它使用 DAD 中的 <Xcollection> 指定的 XML 集合表中存储的数据来构造 XML 文档，并将生成的第一个（通常是唯一的一个）XML 文档返回到 *resultDoc* CLOB 中。

语法:

```
dxxGenXMLClob(CLOB(100k)                  DAD             /*input*/
      integer        overrideType,    /*input*/
              varchar(varchar_value)    override,        /*input*/
              CLOB(1M)                  resultDoc,      /*output*/
              integer                   valid,          /*output*/
              integer                   numDocs,        /*output*/
              long                      returnCode,     /*output*/
              varchar(1024)             returnMsg),     /*output*/
```

其中，*varchar_value* 对于 Windows 和 UNIX 是 32672，对于 iSeries 和 z/OS 是 16366。

参数:

表 58. dxxGenXMLClob 参数

参数	描述	IN/OUT 参数
DAD	包含 DAD 文件的 CLOB。	IN
overrideType	一个标志，用来指示 *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

表 58. *dxxGenXMLClob 参数 (续)*

参数	描述	IN/OUT 参数
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，需要在 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** 一个字符串，它包含一个或多个由 AND 一词分隔并括在双引号中的表达式。使用此 *overrideType* 时，需要在 DAD 文件中使用 RDB_node 映射。	IN
resultDoc	包含已组成 XML 文档的 CLOB。	OUT
valid	valid 的设置如下: • 如果 VALIDATION=YES，则 valid=1 表示验证成功，valid=0 表示验证不成功。 • 如果 VALIDATION=NO，则 valid=NULL。	OUT
numDocs	根据输入数据将要生成的 XML 文档的数目。 **注:** 当前，只返回第一个文档。	OUT
returnCode	存储过程的返回码。	OUT
returnMsg	在发生错误时返回的消息文本。	OUT

CLOB 参数大小是 1 MB。如果 CLOB 文件大于 1 MB，则 XML Extender 将提供一个命令文件来重新定义存储过程参数。请从 DB2 UDB XML Extender Web 站点下载 `crtgenxc.zip` 文件。此压缩文件包含下列程序:

crtgenxc.db2
用于在 “XML Extender V7.2 修订包 5 UNIX 版和 Windows 版” 和更新版本上使用。

crtgenxc.iseries
用于与 XML Extender for iSeries 配合使用

对于 iSeries，将此文件作为成员放到一个文件中。(例如，将此文件放到 DXXSAMPLES/SQLSTMT 中)。

要指定 CLOB 长度: 在编辑器中打开该文件并修改 *resultDoc* 参数，如以下示例所示。

```
out    resultDoc    clob(clob_size),
```

大小建议: *resultDoc* 参数的大小限制取决于系统设置，但您要知道，此参数中指定的数量是 JDBC 分配的数量，与文档的大小无关。此大小应当能够容纳最大的 XML 文件，但不应超过 1.5 吉字节。

要在 iSeries 上运行命令文件，请从命令行输入:

```
RUNSQLSTM SRCFILE(DXXSAMPLES/SQLSTMT) SRCMBR(CRTGENXC) NAMING(*SQL)
```

其中，*DXXSAMPLES/SQLSTMT* 与您将文件下载到其中的“库”和“文件”的名称相匹配。

相关任务:

- 第 55 页的『通过使用 SQL 映射来组成 XML 文档』
- 第 58 页的『通过使用 RDB_node 映射来组成 XML 集合』
- 第 180 页的『调用 XML Extender 组合存储过程』

相关参考:

- 第 180 页的『XML Extender 组合存储过程』
- 第 xi 页的『如何阅读语法图』
- 第 229 页的附录 C, 『XML Extender 限制』

dxxRetrieveXMLClob 存储过程

用途:

dxxRetrieveXMLClob 存储过程启用源自关系数据的文档组合。

使用 dxxRetrieveXMLClob 时的需求与 dxxGenXMLClob 的需求相同。唯一的区别就是 DAD 不是 dxxRetrieveXMLClob 的输入参数，但它是启用的 XML 集合的名称。

语法:

```
dxxRetrieveXMLClob(varchar(collectionName)          collelctionName  /*input*/
      integer        overrideType,    /*input*/
              varchar(varchar_value)   override,       /*input*/
              CLOB(1M)                 resultDoc,      /*output*/
              integer                  valid,          /*output*/
              integer                  numDocs,        /*output*/
              long                     returnCode,     /*output*/
              varchar(1024)            returnMsg),     /*output*/
```

参数:

表 59. dxxRetrieveXMLClob 参数

参数	描述	IN/OUT 参数
collectionName	启用的 XML 集合的名称。	IN
overrideType	一个标志，用来指示 *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

表 59. *dxxRetrieveXMLClob 参数 (续)*

参数	描述	IN/OUT 参数
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，需要在 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** 一个字符串，它包含一个或多个由 AND 一词分隔并括在双引号中的表达式。使用此 *overrideType* 时，需要在 DAD 文件中使用 RDB_node 映射。	IN
resultDoc	结果表中的最大行数。	IN
valid	valid 的设置如下: • 如果 VALIDATION=YES，则 valid=1 表示验证成功，valid=0 表示验证不成功。 • 如果 VALIDATION=NO，则 valid=NULL。	OUT
numDocs	根据输入数据将要生成的 XML 文档的数目。注意: 当前只返回第一个文档。	OUT
returnCode	存储过程的返回码。	OUT
returnMsg	在发生错误时返回的消息文本。	OUT

CLOB 参数大小是 1 MB。如果 CLOB 文件大于 1 MB，则 XML Extender 将提供一个命令文件来重新定义存储过程参数。请从 DB2 UDB XML Extender Web 站点下载 `crtgenxc.zip` 文件。此压缩文件包含下列程序:

crtgenxc.db2
用于在 UNIX 和 Windows 上的 “XML Extender V7.2 修订包 5” 和更新版本上使用。

crtgenxc.iseries
用于与 XML Extender for iSeries 配合使用

对于 iSeries，将此文件作为成员放到一个文件中。(例如，将此文件放到 DXXSAMPLES/SQLSTMT 中)。

要指定 CLOB 长度: 在编辑器中打开该文件并修改 *resultDoc* 参数，如以下示例所示。

```
out    resultDoc     clob(clob_size),
```

大小建议: *resultDoc* 参数的大小限制取决于系统设置，但您要知道，此参数中指定的数量是 JDBC 分配的数量，与文档的大小无关。此大小应当能够容纳最大的 XML 文件，但不应超过 1.5 吉字节。

要在 iSeries 上运行命令文件，请从命令行输入:

```
RUNSQLSTM SRCFILE(DXXSAMPLES/SQLSTMT) SRCMBR(CRTGENXC) NAMING(*SQL)
```

其中，*DXXSAMPLES/SQLSTMT* 与您将文件下载到其中的“库”和“文件”的名称相匹配。

相关任务:

- 第 55 页的『通过使用 SQL 映射来组成 XML 文档』
- 第 58 页的『通过使用 RDB_node 映射来组成 XML 集合』
- 第 180 页的『调用 XML Extender 组合存储过程』

相关参考:

- 第 180 页的『XML Extender 组合存储过程』
- 第 xi 页的『如何阅读语法图』
- 第 229 页的附录 C, 『XML Extender 限制』

XML Extender 分解存储过程

分解存储过程 dxxInsertXML() 和 dxxShredXML() 用来拆散或分割入局 XML 文档，并将数据存储在新的或现有的数据库表中。dxxInsertXML() 存储过程将已启用的 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() 存储过程

用途:

基于 DAD 文件映射分解 XML 文件，将 XML 元素和属性的内容存储在指定的 DB2 UDB 表中。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有表一致。此存储过程要求 DAD 中的连接条件中指定的列与现有表中的主键 – 外键关系相对应。在根 element_node 的 RDB_node 中指定的连接条件列必须存在于这些表中。

本节中存储过程的片断是用于解释的样本。完整的工作样本位于 DXXSAMPLES/QCSRC(SHDX) 中。

语法:

```
dxxShredXML(CLOB(100K)    DAD,           /* input */
            CLOB(1M)      xmlobj,        /* input */
                 long           returnCode,  /* output */
               varchar(1024)  returnMsg)       /* output */
```

参数:

表 60. 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

表 60. dxxShredXML() 参数 (续)

参数	描述	IN/OUT 参数
returnMsg	在发生错误时返回的消息文本。	OUT

示例:

以下片断是调用 dxxShredXML() 的一个示例。完整的工作样本位于 DXXSAMPLES/QCSRC(SHDX) 中。

```
#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(100K) xmlDoc; /* input xml document */

long    returnCode;            /* return 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;

FILE   *file_handle;
long    file_length=0;

/* initialize the DAD CLOB object. */
file_handle = fopen( "/dxxsamples/dad/getstart_xcollection.dad", "r" );
if ( file_handle != NULL ) {
  file_length = fread ((void *) &dad.data
, 1, FILE_SIZE, file_handle);
  if (file_length == 0) {
    printf ("Error reading dad file getstart_xcollection.dad\n");
      rc = -1;
      goto exit;
 } else
    dad.length = file_length;
}
else {
  printf("Error opening dad file  \n");
      rc = -1;
      goto exit;
}

/* Initialize the XML CLOB object. */
  file_handle = fopen( "/dxxsamples/xml/getstart_xcollection.xml", "r" );
  if ( file_handle != NULL ) {
    file_length = fread ((void *) &xmlDoc.data
, 1, FILE_SIZE,
                                  file_handle);
    if (file_length == 0) {
      printf ("Error reading xml file getstart_xcollection.xml \n");
      rc = -1;
      goto exit;
    } else
        xmlDoc.length = file_length;
  }
  else {
    printf("Error opening xml file \n");
      rc = -1;
      goto exit;
```

```
  }

         /* initialize host variable and indicators */
        returnCode = 0;
         msg_txt[0] = '\0';
              dad_ind = 0;
              xmlDoc_ind = 0;
        returnCode_ind = -1;
        returnMsg_ind = -1;

        /* Call the store procedure */
EXEC SQL CALL "DB2XML.DXXSHRED" (:dad:dad_ind;
                                 :xmlDoc:xmlDoc_ind,
                                 :returnCode:returnCode_ind,
                                 :returnMsg:returnMsg_ind);

if (SQLCODE < 0) {
    EXEC SQL ROLLBACK;
  else
    EXEC SQL COMMIT;
}

exit:
  return rc;
```

相关任务:

- 第 60 页的『通过使用 RDB_node 映射来分解 XML 集合』
- 第 86 页的『将 XML 文档分解为 DB2 UDB 数据』
- 第 180 页的『调用 XML Extender 组合存储过程』

相关参考:

- 第 191 页的『XML Extender 分解存储过程』
- 第 xi 页的『如何阅读语法图』
- 第 229 页的附录 C, 『XML Extender 限制』

dxxInsertXML() 存储过程

用途:

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: 返回码和返回消息。

语法:

```
dxxInsertXML(char(collectionName) collectionName, /*input*/
            CLOB(1M)      xmlobj,         /* input */
                 long           returnCode,  /* output */
               varchar(1024)  returnMsg)        /* output */
```

参数:

表 61. 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

表 61. *dxxInsertXML() 参数 (续)*

参数	描述	IN/OUT 参数
returnMsg	在发生错误时返回的消息文本。	OUT

示例:

在下列片段示例中，dxxInsertXML() 调用分解输入 XML 文档 dxx_install/xml/order1.xml，并根据启用的 DAD 文件中所指定的映射将数据插入 SALES_ORDER 集合表中。完整的工作样本位于 DXXSAMPLES/QCSRC(INSX) 中。

```
#include "dxx.h"
#include "dxxrc.h"

              EXEC SQL INCLUDE SQLCA;
               EXEC SQL BEGIN DECLARE SECTION;
char    collectionName[32];    /* name of an XML collection */
SQL TYPE is CLOB(100K) xmlDoc; /* input xml document */
              long    returnCode;        /* return error code */
              char        returnMsg[1024];    /* error message text */
short   collectionName_ind;
              short       xmlDoc_ind;
              short       returnCode_ind;
              short       returnMsg_ind;
               EXEC SQL END DECLARE SECTION;

FILE   *file_handle;
long    file_length=0;

/* initialize the DAD CLOB object. */
file_handle = fopen( "/dxxsamples/dad/getstart_xcollection.dad", "r" );
if ( file_handle != NULL ) {
  file_length = fread ((void *) , &dad.data;
1, FILE_SIZE, file_handle);
  if (file_length == 0) {
    printf ("Error reading dad file getstart_xcollection.dad\n");
      rc = -1;
      goto exit;
 } else
    dad.length = file_length;
}
else {
  printf("Error opening dad file  \n");
      rc = -1;
      goto exit;
}

        /* initialize host variable and indicators */
strcpy(collectionName, "sales_ord");
        returnCode = 0;
         msg_txt[0] = '\0';
collectionName_ind = 0;
              xmlDoc_ind = 0;
        returnCode_ind = -1;
        returnMsg_ind = -1;

        /* Call the store procedure */
EXEC SQL CALL "DB2XML.DXXINSERTXML" (:collection_name:collection_name_ind,
                                :xmlDoc:xmlDoc_ind,
                                :returnCode:returnCode_ind,
                                :returnMsg:returnMsg_ind);

if (SQLCODE < 0) {
    EXEC SQL ROLLBACK;
```

```
  else
    EXEC SQL COMMIT;
}

exit:
  return rc;
```

相关任务:

- 第 60 页的『通过使用 RDB_node 映射来分解 XML 集合』
- 第 86 页的『将 XML 文档分解为 DB2 UDB 数据』
- 第 180 页的『调用 XML Extender 组合存储过程』

相关参考:

- 第 191 页的『XML Extender 分解存储过程』
- 第 xi 页的『如何阅读语法图』
- 第 229 页的附录 C, 『XML Extender 限制』

第 11 章 XML Extender 管理支持表

当启用数据库时，创建 DTD 资源库表（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。每个列都是用特定 PUBLIC 特权创建的。

DTD 引用表

XML Extender 还用作 XML DTD 资源库。当数据库为启用了 XML 的数据库时，创建 DTD 资源库表 DTD_REF。此表的每一行都表示具有附加元数据信息的 DTD。您可以访问此表，并插入您自己的 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 62 中所显示的列。

表 62. 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 DTD 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。此信息是可选的。
CREATOR	VARCHAR(128)	进行首次插入的用户标识。此列是可选的。
UPDATOR	VARCHAR(128)	进行最后更新的用户标识。此列是可选的。

仅当 USAGE_COUNT 为零时，才能由应用程序修改 DTD。

为 PUBLIC 授予的特权

为 PUBLIC 授予了 INSERT、UPDATE、DELETE 和 SELECT 特权。

XML 使用表（XML_USAGE）

XML_USAGE 表存储了每个启用了 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*)。此表的只读特权被授予 PUBLIC。启用数据库的同时，创建 XML_USAGE 表。表 63 中显示 XML_USAGE 表中的列。

表 63. XML_USAGE 表

列名	描述
table_schema	对于 XML 列，它是包含 XML 列的用户表的模式名。对于 XML 集合，它是作为缺省模式名的 DXX_COLL 的值。

表 63. XML_USAGE 表 (续)

列名	描述
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，它是将实体标识为集合的值 DXX_COLLECTION。
col_name	XML 列或 XML 集合的名称。它是组合键和 table_name 的一部分。
DTDID	使插入至 DTD_REF 中的 DTD 与 DAD 文件中指定的 DTD 产生关联的字符串；此值必须与 DAD 中的 DTDID 元素的值匹配。此列是外键。
DAD	与 XML 列或 XML 集合相关联的 DAD 文件的内容。
access_mode	指定使用哪种访问方式：1 表示 XML 集合，0 表示 XML 列
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。
validation	1 表示“是”，0 表示“否”

不要从 XML_USAGE 表中添加、修改或删除项；该表仅供 XML Extender 内部使用。

为 PUBLIC 授予的特权

对于 XML_USAGE，为 PUBLIC 授予了 SELECT 特权。为 DB2XML 授予了 INSERT、DELETE 和 UPDATE 特权。

第 12 章 故障诊断

故障诊断 XML_Extender

程序中的所有嵌入式 SQL 语句和 DB2 UDB 命令行接口（CLI）调用（包括那些调用 DB2 UDB XML Extender 用户定义的函数（UDF）的命令行接口调用）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 UDB 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对程序中的问题进行隔离和修正。

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 “IBM 软件支持”，以隔离和修正问题。XML Extender 包括用于记录 XML Extender 活动的跟踪设施。跟踪信息对于 “IBM 软件支持” 来说可能很有价值。您只应该在 “IBM 软件支持” 的指示下使用跟踪设施。

本章描述跟踪设施、错误代码和消息。

相关参考:

- 第 201 页的『XML Extender 的 SQLSTATE 代码和相关联的消息号』
- 第 205 页的『XML Extender 消息』
- 第 200 页的『停止跟踪』
- 第 199 页的『为 XML Extender 启动跟踪』

为 XML Extender 启动跟踪

用途:

记录 XML Extender 服务器活动。要启动跟踪，将 on 选项应用于 **dxxtrc**，并输入用户概要文件以及要包含跟踪文件的现有目录的名称。当跟踪处于打开状态时，将把文件 dxx*INSTANCE*.trc 放在指定的目录中。 *INSTANCE* 是赋给对其启动跟踪的 “用户概要文件” 的数字 UID 值。跟踪文件大小没有限制。

语法:

从 Qshell 启动跟踪:

```
►►—dxxtrc—on—user_profile—trace_directory———————————————————————►◄
```

从 iSeries 导航器启动跟踪:

```
call schema.QZXMTRC('on', 'user_profile', 'trace_directory');
```

从 OS 命令行启动跟踪:

```
call QDBXM/QZXMTRC PARM(on user_profile 'trace_directory')
```

参数:

表 64. 跟踪参数

参数	描述
user_profile	与 XML Extender 正在其中运行的作业关联的用户概要文件的名称。
trace_directory	现有 路径和目录的名称，在其中放置了 dxx*INSTANCE*.trc。它是必需的，无缺省值。

示例:

下列示例显示如何启动跟踪，它使用 /u/user1/dxx/trace 目录中的 dxxdb2inst1.trc 文件。

从 Qshell:

```
dxxtrc on user1 /u/user1/trace
```

从 iSeries 导航器:

```
call myschema.QZXMTRC('on', 'user1', '/u/user1/trace');
```

从 OS 命令行:

```
call QDBXM/QZXMTRC PARM(on user1 '/u/user1/trace')
```

停止跟踪

用途:

关闭跟踪。不再记录跟踪信息。

建议: 因为运行跟踪记录文件大小不受限制但会影响性能，所以在生产环境中关闭跟踪。

语法:

从 Qshell停止跟踪:

从 iSeries 导航器停止跟踪:

```
call schema.QZXMTRC('off', 'user_profile');
```

从 OS 命令行停止跟踪:

```
call QDBXM/QZXMTRC PARM(off user_profile)
```

参数:

表 65. 跟踪参数

参数	描述
user_profile	与 XML Extender 正在其中运行的作业关联的用户概要文件的名称。

示例:

下列示例演示如何停止跟踪。

从 Qshell:

```
dxxtrc off user1
```

从 iSeries 导航器:

```
call myschema.QZXMTRC('off', 'user1');
```

从 OS 命令行:

```
call QDBXM/QZXMTRC PARM(off user1)
```

XML Extender UDF 返回码

嵌入式 SQL 语句在 SQLCA 结构的 SQLCODE、SQLWARN 和 SQLSTATE 字段中返回代码。此结构是在 SQLCA INCLUDE 文件中定义的。（有关 SQLCA 结构和 SQLCA INCLUDE 文件的更多信息，请参阅 *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更多信息，请参阅 *CLI Guide and Reference*。）

SQLCODE 值 0 表示语句运行成功（可能会有警告条件）。正数 SQLCODE 值表示语句运行成功，但发出了警告。（嵌入式 SQL 语句返回关于警告的信息，该警告与 SQLWARN 字段中的 0 或正数 SQLCODE 值相关联。）负数 SQLCODE 值表示发生了错误。

DB2 将消息与每一个 SQLCODE 值相关联。如果 XML Extender UDF 遇到警告或错误状态，它会将相关联的信息传送至 DB2 UDB 以便包括在 SQLCODE 消息中。

调用 DB2 XML Extender UDF 的嵌入式 SQL 语句和 DB2 UDB 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 UDB 返回这些值的方式与它对其他嵌入式 SQL 语句或其他 DB2 UDB 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 UDB XML Extender UDF 的嵌入式 SQL 语句或 DB2 UDB CLI 调用的方式相同。

XML Extender 存储过程返回码

XML Extender 提供了返回码来帮助解决与存储过程相关的问题。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相匹配。

```
dxx_install/include/dxxrc.h
```

相关参考:

- 第 201 页的『XML Extender 的 SQLSTATE 代码和相关联的消息号』

XML Extender 的 SQLSTATE 代码和相关联的消息号

表 66. 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。

表 66. SQLSTATE 代码和相关联的消息号 (续)

SQLSTATE	消息号	描述
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。
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会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	在试图启用该列时，XML Extender 找不到 DTD 标识，它是在文档访问定义（DAD）文件中为 DTD 指定的标识。
	DXXQ060E	XML Extender 在试图启用列时，未能找到 SCHEMA 标识。SCHEMA 标识对应于 nonamespacelocation 标记的位置属性值，此标记位于 DAD 文件的 schemabindings 标记内。
38X14	DXXD000E	试图将无效文档存储到表中。验证失败。
38X15	DXXA056E	文档访问定义（DAD）文件中的验证元素错误或丢失。

表 66. SQLSTATE 代码和相关联的消息号 (续)

SQLSTATE	消息号	描述
	DXXA057E	文档访问定义（DAD）文件中的副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。
	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。
	DXXQ056E	无法将指定元素／属性映射至指定作为外键的一部分的列。外键的数据值由主键的数据值决定; XML 文档中的指定元素／属性对表和列的映射不是必需的。
	DXXQ057E	schemabindings 和 DTD ID 标记不能同时存在于 DAD 文件中。
	DXXQ058E	schemabindings 标记内的 nonamespacelocation 标记在 DAD 文件中丢失。
	DXXQ059E	无法在用于模式验证的 DAD 中的 XCollection 标记内找到 doctype 标记。
	DXXQ062E	此错误状态通常由给定元素或属性的父代 element_node 上丢失了 multi_occurrence = YES 规范而导致。
	DXXQ063E	文档访问定义（DAD）文件中指定 element_node 上的 multi_occurrence 属性值错误或丢失。此值必须为 “yes” 或 “no”，且不区分大小写。
	DXXQ064E	在连接状态中指定的键列未映射至任何元素或属性节点。
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）文件中的 SQL 语句或覆盖此语句的语句无效。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。

表 66. SQLSTATE 代码和相关联的消息号 (续)

SQLSTATE	消息号	描述
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 DXXQ040E	attribute_node 或 text_node 的 RDB_node 必须有一个表。
	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。
	DXXQ040E	文档访问定义（DAD）文件中的指定元素名是错误的。
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	文档访问定义（DAD）文件中条件元素中的条件具有无效的格式。
	DXXQ061E	字符串表达式的格式无效。如果字符串是日期、时间或时间戳记值，则此语法不符合其数据类型。
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 子句从将 SQL 映射至 XML 的文档访问定义（DAD）文件中的 SQL 语句中丢失。

表 66. SQLSTATE 代码和相关联的消息号 (续)

SQLSTATE	消息号	描述
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。
38x33	DXXG005E	此发行版中不支持该参数。以后的发行版中将支持此参数。
38x34	DXXG000E	指定了无效文件名。
38X36	DXXA073E	用户尝试启用数据库时，该数据库未绑定。
38X37	DXXG007E	服务器操作系统语言环境与 DB2 UDB 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。
38X41	DXXQ048E	样式表处理器返回内部错误。XML 文档或样式表可能无效。
38X42	DXXQ049E	此目录中已存在指定的输出文件。
38X43	DXXQ050E	UDF 无法为指定目录中的输出文档创建唯一的文件名，原因是它不具有访问权。所有可生成的文件名都已在使用中，或者目录可能不存在。
38X44	DXXQ051E	一个或多个输入或输出参数具有无效值。
38X45	DXXQ055E	转换操作期间发生 ICU 错误。

XML Extender 消息

DXXA000I 正在启用列 <*column_name*>。请稍候。

解释: 这是参考消息。

用户回答: 不需要任何操作。

DXXA001S 构件 <*build_ID*>、文件 <*file_name*> 和行 <*line_number*> 中出现了意外错误。

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，一定要包括所有的消息文本、跟踪文件和如何再现该问题的说明。

DXXA002I 连接到数据库 <*database*>。

解释: 这是参考消息。

用户回答: 不需要任何操作。

DXXA003E 无法连接到数据库 <*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 无法启用数据库 <*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: 这是参考消息。

用户回答: 不需要任何操作。

DXXA006I 已成功启用数据库 <*database*>。

解释: 这是参考消息。

用户回答: 不需要任何操作。

DXXA007E 无法禁用数据库 <*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: 这是参考消息。

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: 此存储过程仅用于 “XML 列”。

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 标识 <*dtdid*> 失败。

解释: 在试图启用该列时，XML Extender 找不到 DTD 标识，它是在文档访问定义（DAD）文件中为 DTD 指定的标识。

用户回答: 确保在 DAD 文件中指定了 DTD 标识的正确值。

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 试图变更表 <*table_name*> 失败。

解释: 当试图启用该列时，XML Extender 未能变更指定的表。

用户回答: 检查变更该表所需的特权。

DXXA014E 指定的根标识列 <*root_id*> 不是表 <*table_name*> 的单一主键。

解释: 指定的根标识不是键，或不是表 *table_name* 的单个键。

用户回答: 确保指定的根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根标识选项指定主列。

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 在表 <*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 无法创建缺省视图 <*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: 这是参考消息。

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称、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 无法删除缺省视图 <*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 的管理用户标识具有删除该缺省视图所需的特权。

DXXA026E 无法删除副表 <*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:

- 系统内存不足。
- 具有此名称的触发器不存在。

用户回答: 使用跟踪设施创建一个跟踪文件并尝试更正该问题。如果问题仍存在，请与软件服务提供者联系，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:

- 系统内存不足。
- 具有此名称的触发器不存在。

用户回答: 使用跟踪设施创建一个跟踪文件并尝试更正该问题。如果问题仍存在，请与软件服务提供者联系，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:

- 系统内存不足。
- 具有此名称的触发器不存在。

用户回答: 使用跟踪设施创建一个跟踪文件并尝试更正该问题。如果问题仍存在，请与软件服务提供者联系，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:

- 系统内存不足。
- 具有此名称的触发器不存在。

用户回答: 使用跟踪设施创建一个跟踪文件并尝试更正该问题。如果问题仍存在，请与软件服务提供者联系，并将跟踪文件提供给他们。

DXXA031E 无法将应用程序表中的 DXXROOT_ID 列值复位为 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变更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 表中的关联行。

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 已成功地禁用了列 <*column_name*>。

解释: 这是参考消息

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: 这是参考消息。

用户回答: 不需要任何操作。

DXXA036I XML Extender 已成功禁用了数据库 <*database*>。

解释: 这是参考消息。

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 无法创建副表 <*side_table*>。

解释: 当试图启用一列时，XML Extender 无法创建指定的副表。

用户回答: 确保 XML Extender 的管理员用户标识具有创建副表所需的特权。

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:

- DAD 文件的语法不正确。
- 系统内存不足。
- 存在另一个具有相同名称的触发器。

用户回答: 使用跟踪设施创建一个跟踪文件并尝试更正该问题。如果问题仍存在，请与软件服务提供者联系，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:

- DAD 文件的语法不正确。
- 系统内存不足。
- 存在另一个具有相同名称的触发器。

用户回答: 使用跟踪设施创建一个跟踪文件并尝试更正该问题。如果问题仍存在，请与软件服务提供者联系，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:

- DAD 文件的语法不正确。
- 系统内存不足。
- 存在另一个具有相同名称的触发器。

用户回答: 使用跟踪设施创建一个跟踪文件并尝试更正该问题。如果问题仍存在，请与软件服务提供者联系，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:

- DAD 文件的语法不正确。
- 系统内存不足。
- 存在另一个具有相同名称的触发器。

用户回答: 使用跟踪设施创建一个跟踪文件并尝试更正该问题。如果问题仍存在，请与软件服务提供者联系，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:

- 系统内存不足。
- 具有此名称的触发器不存在。

用户回答: 使用跟踪设施创建一个跟踪文件并尝试更正该问题。如果问题仍存在，请与软件服务提供者联系，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:

- DAD 文件的语法不正确。
- 系统内存不足。
- 存在另一个具有相同名称的触发器。

用户回答: 使用跟踪设施创建一个跟踪文件并尝试更正该问题。如果问题仍存在，请与软件服务提供者联系，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:

- DAD 文件的语法不正确。
- 系统内存不足。
- 存在另一个具有相同名称的触发器。

用户回答: 使用跟踪设施创建一个跟踪文件并尝试更正该问题。如果问题仍存在，请与软件服务提供者联系，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:

- DAD 文件的语法不正确。
- 系统内存不足。
- 存在另一个具有相同名称的触发器。

用户回答: 使用跟踪设施创建一个跟踪文件并尝试更正该问题。如果问题仍存在，请与软件服务提供者联系，并将跟踪文件提供给他们。

DXXA056E DAD 文件中的验证值 <*validation_value*> 无效。

解释: 文档访问定义（DAD）文件中的验证元素错误或已丢失。

用户回答: 确保在 DAD 文件中正确地指定了验证元素。

DXXA057E DAD 中的副表名 <*side_table_name*> 无效。

解释: 文档访问定义（DAD）文件中的副表的名称属性错误或已丢失。

用户回答: 确保在 DAD 文件中正确地指定了副表的名称属性。

DXXA058E DAD 文件中的列名 <*column_name*> 无效。

解释: 文档访问定义（DAD）文件中的列的名称属性错误或已丢失。

用户回答: 确保在 DAD 文件中正确地指定了某列的名称属性。

DXXA059E DAD 文件中的列 <*column_name*> 的类型 <*column_type*> 无效。

解释: 文档访问定义（DAD）文件中的列的类型属性错误或已丢失。

用户回答: 确保在 DAD 文件中正确地指定了某列的类型属性。

DXXA060E DAD 文件中 <*column_name*> 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*> 无效。

解释: 文档访问定义（DAD）文件中的列的 multi_occurrence 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 multi_occurrence 属性。

DXXA062E 无法检索表 <*table_name*> 中 <*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: 这是参考消息。

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: 这是参考消息。

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: 这是参考消息。

用户回答: 不需要任何响应。

DXXA067I XML Extender 已成功启用了集合 <*collection_name*>。

解释: 这是参考消息。

用户回答: 不需要任何响应。

DXXA068I XML Extender 成功打开跟踪。

解释: 这是参考消息。

用户回答: 不需要任何响应。

DXXA069I XML Extender 成功关闭跟踪。

解释: 这是参考消息。

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: 将输入参数声明为 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: 将输入参数声明为 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 iSeries 用户标识。

DXXA077E 许可密钥无效。有关更多的详细信息，请参阅服务器错误记录。

解释: 软件许可证已到期或不存在。

用户回答: 与服务供应商联系以获取新的软件许可证。

DXXC000E 无法打开指定文件。

解释: XML Extender 无法打开指定的文件。

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 找不到指定的文件。

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 无法读取指定的文件中的数据。

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 无法将数据写入文件中。

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 无法运行指定的定位器。

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: 文件大小大于 XMLVarchar 的大小，XML Extender 无法导入文件中的所有数据。

用户回答: 使用 XMLCLOB 列类型。

DXXC006E 输入文件超过 DB2 UDB LOB 的限制。

解释: 文件大小大于 XMLCLOB 的大小，XML Extender 无法导入文件中的所有数据。

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 无法除去文件 <*file_name*>。

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。

DXXC013E 抽取 UDF 的结果超出 UDF 返回类型的大小限制。

解释: 抽取 UDF 返回的数据必须适合该 UDF 返回类型的大小限制，此限制在《DB2 UDB XML Extenders 管理与编程指南》中定义。例如，extractVarchar 的结果一定不能超过 4000 个字节（包括终止 NULL）。

用户回答: 请使用具有更大返回类型大小限制的抽取 UDF: extractChar() 是 254 个字节，extractVarchar() 是 4 KB，而 extractClob() 是 2 GB。

DXXD000E 拒绝无效的 XML 文档。

解释: 试图将无效文档存储到表中。验证已失败。

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。标量函数仅可使用不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 在搜索路径中位置 <*position*> **附近发生语法错误。**

解释: 路径表达式在语法上不正确。

用户回答: 更正查询的搜索路径自变量。参阅文档以了解路径表达式的语法。

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 构件 <*build_ID*>、文件 <*file_name*> 和行 <*line_number*> 中出现内部错误。

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，一定要包括所有的消息、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，则参阅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更正问题。

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: 检查参数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*>、构件 <*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消息、跟踪作业记录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 UDB 代码页 <*code_page*> 不一致。

解释: 服务器操作系统的语言环境与 DB2 UDB 代码页不一致。

用户回答: 更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: 更正服务器操作环境的语言环境然后重新启动 DB2。

DXXG017E 在构件 *build_ID*、文件 *file_name* 以及行 *line_number* 中超过了 *XML_Extender_constant* 的限制。

解释: 检查《XML Extender 管理与编程指南》以了解应用程序是否超过了限制表中的值。如果未超过任何限制，则与“软件服务供应商”联系。报告错误时，要包括所有的输出消息、跟踪文件和有关如何再现该问题的信息，如输入 DAD、XML 文档和表定义。

用户回答: 更正服务器操作环境的语言环境然后重新启动 DB2。

DXXM001W 发生 DB2 UDB 错误。

解释: DB2 遇到指定的错误。

用户回答: 有关进一步的解释，请参阅任何伴随的消息，并参阅操作系统的 DB2 UDB 消息和代码文档。

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 对于 XML 生成的 SQL 语句无效。

解释: 文档访问定义（DAD）中的 SQL 语句或覆盖该语句的语句是无效的。SELECT 语句对于生成 XML 文档是必需的。

用户回答: 更正 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。

DXXQ005E 错误的关系映射。元素 <*element_name*> 处于比其子列 <*column_name*> 更低的级别。

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自顶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 UDB 内置函数 generate_unique() 为该表生成一个键。

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*> 不具有列元素或 RDB_node。

解释: 文档访问定义（DAD）中的 attribute_node 元素没有列元素或 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 元素没有列元素。

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。请参阅 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: 文档访问定义（DAD）文件中 element_node 元素没有名称属性。

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: 文档访问定义（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 文件的顶层元素中定义。

解释: 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。

用户回答: 确保 RDB 节点的表是在 DAD 文件的顶层元素中定义的。

DXXQ017E 结果表 <*table_name*> 中的列太小。

解释: 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。

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 子句从将 SQL 映射至 XML 的文档访问定义（DAD）文件中的 SQL 语句中丢失。

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*> 应该具有类型 <*type_name*>。

解释: 列类型是错误的。

用户回答: 更正文档访问定义（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: 更正文档访问定义（DAD）中的列长度。

DXXQ024E 无法创建表 <*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*> 太大，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: 更正文档访问定义（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: 更正文档访问定义（DAD）中的列名的长度。

DXXQ033E 以 <*identifier*> 开头的标识无效

解释: 该字符串不是有效的 DB2 UDB SQL 标识。

用户回答: 更正 DAD 中的字符串以符合 DB2 UDB SQL 标识的规则。

DXXQ034E DAD 的顶部 RDB_node 中的无效条件元素: <*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: 请参阅 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 中的无效连接条件: <*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: 请参阅 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: 更正文档访问定义（DAD）中条件标记的文本。

DXXQ037E 无法生成多次出现的 <*element*>。

解释: 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，以更正 DAD。

DXXQ038E SQL 语句太长: SQL_statement

解释: DAD 的 <*SQL_stmt*> 元素中指定的 SQL 语句超过了允许的字节数。

用户回答: 将 SQL 语句的长度缩短为小于或等于 32765 个字节（对于 Windows 和 UNIX）或 16380 个字节（对于 OS/390 和 iSeries）。

DXXQ039E 对 DAD 文件中的表指定了太多的列。

解释: 用于分解或 RDB 组合的 DAD 文件最多可以有 100 个用于指定同一个表中的唯一列的 text_node 和 attribute_node 元素。

用户回答: 将引用同一个表中的唯一列的 text_node 和 attribute_node 元素的总数减少到 100 或更少。

DXXQ040E DAD 文件中的元素名称 <*element_name*> 无效。

解释: 文档访问定义（DAD）文件中的指定元素名是错误的。

用户回答: 确保在 DAD 文件中正确输入了元素名。请参阅 DAD 文件的 DTD。

DXXQ041W 已成功生成 XML 文档。在指定的覆盖路径中，有一个或多个是无效的，已将它们忽略。

解释: 只指定一个覆盖路径。

用户回答: 确保在 DAD 文件中正确输入了元素名。请参阅 DAD 文件的 DTD。

DXXQ043E 未在元素 <*elem_name*> 下找到属性 <*attr_name*>。

解释: 属性 <*attr_name*> 未存在于元素 <*elem_name*> 或它的其中一个子元素中。

用户回答: 确保 XML 文档中在 DAD 需要的地方都出现属性。

DXXQ044E 元素 <*elem_name*> 没有祖先元素 <*ancestor*>。

解释: 根据 DAD，<*ancestor*> 是 <*elem_name*> 的祖先元素。在 XML 文档中，一个或多个元素 <*elem_name*> 没有这样的祖先。

用户回答: 确保 XML 文档中的元素嵌套符合相应 DAD 中的指定。

DXXQ045E 元素 <*elem_name*> 下面的子树包含多个名为 <*attrib_name*> 的属性。

解释: 在 XML 文档中，<*elem_name*> 下面的子树包含 <*attrib_name*> 属性的多个实例，根据 DAD，该属性将要分解到同一行中。将要分解的元素或属性必须具有唯一的名称。

用户回答: 确保子树中的元素或属性具有唯一的名称。

DXXQ046W DAD 中未找到 DTD 标识。

解释: 在 DAD 中，VALIDATION 设置为 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。

DXXQ047E 在行 <mv> linenumber</mv> 列 *colnumber* 处发生解析器错误: *msg*

解释: 由于发生报告的错误，解析器未能解析文档。

用户回答: 更正文档中的错误，如有必要的话，查阅 XML 规范。

DXXQ048E 内部错误 – 请参阅跟踪文件。

解释: 样式表处理器返回内部错误。XML 文档或样式表可能无效。

用户回答: 确保 XML 文档和样式表有效。

DXXQ049E 输出文件已存在。

解释: 此目录中已存在指定的输出文件。

用户回答: 将输出文档的输出路径或文件名更改为唯一的名称或删除现有文件。

DXXQ050E 无法创建唯一的文件名。

解释: UDF 无法为指定目录中的输出文档创建唯一的文件名，原因如下：它不具有访问权、所有可生成的文件名都已在使用中或者目录不存在。

用户回答: 确保 UDF 对指定的目录具有访问权，转至带有可用文件名的目录。

DXXQ051E 没有输入或输出数据。

解释: 一个或多个输入或输出参数具有无效值。

用户回答: 检查语句，了解是否遗漏了必需的参数。

DXXQ052E 访问 DB2XML.XML_USAGE 表时出错。

解释: 尚未启用数据库，或者已删除 DB2XML.XML_USAGE 表。

用户回答: 确保已启用数据库并且 DB2XML.XML_USAGE 表可访问。

DXXQ053E SQL 语句失败: *msg*

解释: XML Extender 处理期间生成的 SQL 语句未能执行。已删除 DB2XML.XML_USAGE。

用户回答: 检查跟踪以获取更多的详细信息。如果不能更正错误状态，则与软件服务供应商联系。在报告错误时，一定要包括所有的消息、跟踪文件和如何再现该错误的说明。

DXXQ054E 无效输入参数: *param*

解释: 存储过程或 UDF 的指定输入参数无效。

用户回答: 检查相关存储过程或 UDF 的特征符，确保实际输入参数正确。

DXXQ055E ICU 错误: *uerror*

解释: 转换操作期间遇到 ICU 错误。

用户回答: 向软件服务提供者报告错误。包括跟踪文件、错误消息和指令以再现错误。

DXXQ056E 无法将元素 / 属性 *xmlname* 映射至已指定作为外键的一部分的列 (表 *table* 中的列 *column*)。

解释: 无法将指定元素 / 属性映射至指定作为外键的一部分的列。外键的数据值由主键的数据值决定；XML 文档中的指定元素 / 属性对表和列的映射不是必需的。

用户回答: 除去 RDB_node 对 DAD 中指定列和表的映射。

DXXQ057E schemabindings 和 dtdid 标记不能同时存在于 DAD 文件中。

解释: schemabindings 和 dtdid 标记不能同时存在于 DAD 文件中。

用户回答: 检查 schemabindings 标记或 dtdid 标记是否存在于 DAD 文件中，但不是同时存在。

DXXQ058E schemabindings 标记内的 nonamespacelocation 标记在 DAD 文件中丢失。

解释: schemabindings 标记内的 nonamespacelocation 标记在 DAD 文件中丢失。

用户回答: 将 nonamespacelocation 标记添加至 schemabindings 标记。

DXXQ059E 无法在用于模式验证的 DAD 中的 XCollection 标记内找到 doctype 标记。

解释: 无法在用于模式验证的 DAD 中的 XCollection 标记内找到 doctype 标记。

用户回答: 除去用于模式验证的 Xcollection 标记内的 doctype 标记。

DXXQ060E 试图查找 SCHEMA 标识 *schemaid* 失败。

解释: XML Extender 在试图启用列时，找不到 SCHEMA 标识。SCHEMA 标识对应于 nonamespacelocation 标记的位置属性值，此标记位于 DAD 文件的 schemabindings 标记内。

用户回答: 检查是否在 DAD 文件中指定了 SCHEMA 标识的正确值。

DXXQ061E 字符串的格式无效。

解释: 字符串表达式的格式无效。如果字符串是日期、时间或时间戳记值，则此语法不符合其数据类型。

用户回答: 检查日期、时间或时间戳记值的格式是否符合其数据类型的格式。

DXXQ062E 没有留下 *table* 的结果集的任何行以便为 *element* 产生 XML 值。

解释: 此错误状态通常由给定元素或属性的父代 element_node 上丢失了 multi_occurrence = YES 规范而导致。

用户回答: 检查 DAD 父代 element_node 上 multi_occurrence 的值是否正确地反映了子 element_nodes 的多样性。

DXXQ063E DAD 文件中的 *elementname* 上的 multi_occurrence 属性值无效。

解释: 文档访问定义（DAD）文件中指定 element_node 上的 multi_occurrence 的属性值错误或丢失。此值必须为“yes”或“no”，且不区分大小写。

用户回答: 确保在 DAD 文件中正确地指定了 multi_occurrence 属性。

DXXQ064E 在外键表 *table* 中未找到列 *column*。

解释: 在连接状态中指定的键列将不会被映射至任何元素或属性节点。

用户回答: 检查以确保在 DAD 文件中指定的连接条件正确，而且将所有键列均映射至元素或属性节点。

DXXQ065I 已成功重新生成所有与启用 XML 的列相关的触发器。

解释: 这只是参考消息。

用户回答: 不需要任何操作。

DXXQ066E 表 *tablename* 的主键不存在。

解释: XML Extender 无法确定表 *tablename* 的主键。在为 XML 启用列之后，可能已删除了表的主键。

用户回答: 变更表以在为 XML 启用列时，添加指定为 ROOT ID 的主键。

DXXQ067E 试图*操作*失败。

解释: 试图*操作*时，出现一个 SQL 错误。

用户回答: 与“软件服务供应商”联系。在报告错误时，确保包括 XML Extender 跟踪文件。

第 5 部分 附录

附录 A. 样本

此附录显示在本书中使用的示例的样本对象。

- 『XML DTD 样本』
- 『XML 文档样本: getstart.xml』
- 第 222 页的『文档访问定义文件』
  - 第 222 页的『样本 DAD 文件: XML 列』
  - 第 223 页的『样本 DAD 文件: XML 集合: SQL 映射』
  - 第 224 页的『样本 DAD 文件: XML: RDB_node 映射』

XML DTD 样本

以下 DTD 用于整本书所引用的 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 15. 样本 XML DTD: getstart.dtd

XML 文档样本: 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: 它包含要构成采购订单的 XML 标记。

```
<?xml version="1.0"?>
<!DOCTYPE Order SYSTEM "dxxsamples/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 16. 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 UDB 关系表。

- 『样本 DAD 文件: XML 列』
- 第 223 页的『样本 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 224 页的『样本 DAD 文件: XML: RDB_node 映射』显示使用 RDB_node 映射的 XML 集合的 DAD。

样本 DAD 文件: XML 列

此 DAD 文件包含用于 XML 列的映射，并且定义要包含 XML 数据的表、副表和列。

```
<?xml version="1.0"?>
<!DOCTYPE Order SYSTEM  "dxxsamples/dtd/dad.dtd">
<DAD>
   <dtdid>
    "dxxsamples/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 17. XML 列的样本 DAD 文件: getstart_xcolumn.dad

样本 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 XML 数据的 DB2 UDB 表、列和条件。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "dxxsamples/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,
   (select db2xml.generate_unique()
  as ship_id, date, mode, part_key from ship_tab) as s
              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 "
dxxsamples/dtd/getstart.dtd"</doctype>
```

图 18. 使用 SQL 映射的 XML 集合的样本 DAD 文件: order_sql.dad (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_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>
              </element_node>
              </element_node>
     </root_node>
     </Xcollection>
</DAD>
```

图 18. 使用 SQL 映射的 XML 集合的样本 DAD 文件: order_sql.dad (2/2)

样本 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 UDB 表、列和条件。

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "SQLLIB/samples/db2xml/dtd/dad.dtd>
<DAD>
 <dtdid>E:\dtd\lineItem.dtd</dtdid>
   <validation>YES</validation>
<Xcollection>
<prolog>?xml version="1.0"?</prolog>
 <doctype>!DOCTYPE Order SYSTEM
                  "SQLLIB/samples/db2xml/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">

    <element_node name="Name">
            <text_node>
                    <RDB_node>
                    <table name="order_tab"/>
                  <column name="customer_name"/>
                    </RDB_node>
                </text_node>
              </element_node>
     <element_node name="Email">
            <text_node>
                    <RDB_node>
                    <table name="order_tab"/>
                   <column name="customer_email"/>
                    </RDB_node>
                </text_node>
              </element_node>
              </element_node>
           <element_node name="Part">
          <attribute_node name="Key">
                    <RDB_node>
                    <table name="part_tab"/>
                    <column name="part_key"/>
                    </RDB_node>
              </attribute_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>
```

图 19. 使用 RDB_node 映射的 XML 集合的样本 DAD 文件: order_rdb.dad (1/2)

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

    <element_node name="Quantity">
                <text_node>
                   <RDB_node>
                   <table name="part_tab"/>
                      <column name="qty"/>
                   </RDB_node>
                </text_node>
             </element_node>
      <element_node name="Shipment" multi_occurrence="YES">
             <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 19. 使用 RDB_node 映射的 XML 集合的样本 DAD 文件: order_rdb.dad (2/2)

附录 B. 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，必须正确编码这些文件。XML Extender 会在处理文件时做出某些假定，您需要了解它是如何处理代码页转换的。主要的注意事项有:

- 确保从 DB2 UDB 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 解析器处理文档时，该 XML 文档的编码声明也与文档的实际编码一致。
- 确保正确配置语言环境。

对于 iSeries，作业、DB2 UDB 和 XML 文档都必须具有相同的 CCSID。下节描述确保 CCSID 一致的方法。

配置语言环境设置

XML Extender 基于您的语言环境设置从消息编目中选择完成和错误消息。要以您的语言接收消息，必须安装 XML Extender 消息编目，并正确地设置语言环境。XML Extender 在 IFS 目录 /QIBM/ProdData/DB2Extenders/XML/MRI*xxxx* 中安装您的语言的消息编目，其中 *xxxx* 是语言代码。

例如，英语 2924 的消息编目安装在目录: /QIBM/ProdData/DB2Extenders/XML/MRI2924/dxx.cat。要让 XML Extender 选择“英语 2924”消息编目，请使用 **WRKUSRPRF** 命令来设置用户概要文件:

```
语言标识 . . . . . . . . . . . . LANGID    ENU
国家或地区标识 . . . . . . . . . CNTRYID   US
编码字符集标识 . . . . . . . . . CCSID     037
```

在此用户概要文件下运行的 XML Extender 的所有实例都将使用 MRI2924 消息编目。

为 XML Extender 编码声明注意事项

*编码声明*指定 XML 文档编码的代码页并出现在 XML 声明语句中。当使用 XML Extender 时，重要的是确保文档的编码与作业和 DB2 相匹配。。

一致的编码和编码声明

当处理 XML 文档或与另一个系统交换 XML 文档时，编码声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（如解析器）会对包括编码声明的实体而不是在声明中命名的实体产生错误。

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。
- XML 文档的声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声明编码不同）。

声明编码

编码声明的缺省值是 UTF-8，无编码声明意味着文档的编码声明为 UTF-8。

要声明编码值:

在 XML 文档声明中，使用客户机代码页的名称来指定编码声明。例如:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

防止不一致 XML 文档的建议

在将文档交给 XML 处理器（如解析器）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）:
  - 将文档转换为声明编码代码页
  - 覆盖声明编码（如果该工具具有覆盖设施的话）
  - 以手工方式将导出的文档的编码声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已组成文档的结果表）:
  - 将文档转换为声明编码代码页
  - 覆盖声明编码（如果该工具具有覆盖设施的话）
  - 在运行存储过程之前，使客户机设置 CCSID 变量，以强制客户机代码页转换为与 XML 文档的编码声明兼容的代码页。
  - 以手工方式将导出的文档的编码声明更改为文档的实际编码（即客户机代码页）

附录 C. XML Extender 限制

本主题描述下列项的限制:

- XML Extender 对象
- 由用户定义的函数返回的值
- 存储过程参数
- 管理支持表列
- 组合和分解

下表描述了 XML Extender 对象的限制。

表 67. XML Extender 对象的限制

对象	限制
分解 XML 集合中的表中的最大行数	来自每个分解的 XML 文档的 10240 行
缺省视图中指定的列名的最大字符数	10 个字符
作为参数值指定的 XML 文件路径名的最大字节数	512 个字节
用于 SQL 组成的 DAD 文件中的 sql_stmt 元素的长度	Windows 和 UNIX 操作系统: 32,765 个字节。OS/390 和 iSeries 操作系统: 16,380 个字节。
一个表的最大列数, 对用于 RDB_node 分解的 DAD 文件中的一个表指定这些列	500 列（表的列）, 由 DAD 文件中的 text_node 和 attribute_node 元素指定。

下表描述了 XML Extender 用户定义的函数返回的限制值。

表 68. 用户定义的函数值的限制

用户定义的函数返回的值	限制
由 extractCHAR UDF 返回的最大字节数	254 个字节
由 extractCLOB UDF 返回的最大字节数	2 吉字节
由 extractVARCHAR UDF 返回的最大字节数	4 千字节

下表描述了 XML Extender 存储过程参数的限制。

表 69. 存储过程参数的限制

存储过程参数	限制
XML 文档 CLOB 的最大大小[1]	1 MB
“文档访问定义”（DAD）CLOB 的最大大小[1]	100 KB
collectionName 的最大大小	30 个字节
colName 的最大大小	30 个字节
dbName 的最大大小	8 个字节
defaultView 的最大大小	128 个字节
rootID 的最大大小	30 个字节

表 69. 存储过程参数的限制 (续)

存储过程参数	限制
resultTabName 的最大大小	18 个字节
tablespace 的最大大小	8 个字节
tbName 的最大大小[2]	18 个字节
resultColumn 的最大大小	30 个字节
validColumn 的最大大小	30 个字节
varchar_value 的最大大小	16366 个字节

注:

1. 可对 dxxGenXMLClob 和 dxxRetrieveXMLCLOB 更改此大小。
2. 如果通过模式名限定 *tbName* 参数的值，则整个名称（包括分隔符）不得超过 128 个字节。

下表描述了 DB2XML.DTD_REF 表的限制。

表 70. XML Extender 限制

DB2XML.DTD_REF 表列	限制
AUTHOR 列的大小	128 个字节
CREATOR 列的大小	128 个字节
UPDATOR 列的大小	128 个字节
DTDID 列的大小	128 个字节
CLOB 列的大小	100 KB

当 DB2 UDB 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后的名称时会超过限制。

下表描述了组合和分解限制。

表 71. XML Extender 组合和分解限制

对象	限制
插入分解 XML 集合中的表中的最大行数	来自每个分解的 XML 文档的 10240 行
DAD 中的 elements_node 或 attribute_node 中的名称属性的最大长度	63 个字节
缺省视图中指定的列名的最大字符数	10 个字符
作为参数值指定的 XMLFile 路径名的最大字节数	512 个字节

DB2DXX_MIN_TMPFILE_SIZE 环境变量:

XML Extender 可以将大量文档放置在临时文件中，以避免处理期间使用过多内存。在具有大量物理内存的系统上，可以避免将文档移到临时文件中，从而减少输入／输出活动量。环境变量 DB2DXX_MIN_TMPFILE_SIZE 指示 XML Extender 使用内存缓冲区（而不是临时文件）来处理小于指定值的文档。此变量仅适用于服务器，不适用于客户机。如果多个物理节点参与多节点分区，可在每个节点上对此变量进行不同地设

置，以准确地反映每台机器上安装的内存量。如果未设置环境变量，处理期间系统自动将大于 128KB 的文档放置到临时文件中，在内存中处理小于 128K 的文档。

XML Extender 词汇表

〔B〕

本地文件系统（local file system）： 存在于 DB2 中的文件系统。

边界搜索（bound search）： 在与字边界有关的“韩国语”文档中的一种搜索方式。

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）： 一个抽象的容器集合，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。

布尔搜索（Boolean search）： 一种搜索方式，在该搜索方式中，有一个或多个搜索术语通过“布尔”运算符组合在一起。

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。

〔C〕

查询对象（query object）： 指定了功能部件、特征、值和特征权重，以便进行 QBIC 查询的对象。可以对该对象命名并保存它，以便在后续 QBIC 查询中使用。与查询字符串（query string）对照。

触发器（trigger）： 一种机制，一旦在文本列中添加、更改或删除了文档，就自动将有关需要索引的文档的信息添加到*记录表*中。

触发器（trigger）： 更改一个表时要执行的一组操作的定义。触发器可用来执行下列操作，例如: 验证输入数据、为最近插入的行生成一个值、从其他表中读取数据以进行交叉引用、或者将数据写入其他表以进行审计。通常将触发器用来进行完整性检查，或强制执行商务规则。

存储过程（stored procedure）： 过程构造与嵌入式 SQL 语句的块，存储在数据库中，可以通过名称调用。存储过程允许应用程序分成两部分运行。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。

〔D〕

大对象（LOB）（large object（LOB））： 字节序列，其长度最大可达 2 GB。LOB 可以具有三种类型: *二进制大对象*（BLOB）、*字符大对象（CLOB）*或*双字节字符大对象（DBCLOB）*。

代码页（code page）： 对所有代码点指定的图形字符和控制功能含义。例如，对 256 个 8 位代码的代码点指定字符和含义。

单值类型（distinct type）： 请参阅*用户定义的类型（user-defined type）*。

电子数据交换（EDI）（Electronic Data Interchange（EDI））： 企业对企业（B2B）应用程序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）： 在面向对象的编程中，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。

〔E〕

二进制大对象（BLOB）（binary large object（BLOB））： 一种二进制字符串，其最大长度可达 2 GB。图像、音频和视频对象都作为 BLOB 存储在 DB2 数据库中。

〔F〕

访问函数（access function）： 用户提供的函数，它将存储在一列中的文本的数据类型转换为 Text Extender 可以处理的类型。

访问和存储方法（access and storage method）： 通过下列两种主要的访问和存储方法，将 XML 文档与 DB2 UDB 数据库关联: XML 列和 XML 集合。另见 *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。

分析（analyze）： 计算图像的特征的数值，并将该值添加到 QBIC 目录中。

副表（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。

〔G〕

根元素（root element）： XML 文档的顶层元素。

跟踪（tracing）： 一种存储文件中的信息的操作，文件中的信息稍后可用于查找发生错误的原因。

关系数据库节点（RDB_node）（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 table）： 一种表，DB2 UDB Extender 使用此表来处理用户对图像、音频和视频对象的请求。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。也称为*元数据表（metadata table）*。

管理支持表（administrative support tables）： 一种表，DB2 UDB Extender 使用此表来处理用户对 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。

管理（administration）： 是指一系列任务，包括准备文本文档以进行搜索、维护索引以及获取状态信息。

过程（procedure）： 请参阅*存储过程（stored procedure）*。

过载函数（overloaded function）： 存在多个函数实例的函数名。

〔H〕

函数（function）： 请参阅*访问函数（access function）*。

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。

〔J〕

吉字节（GB）（gigabyte）： 十亿个（10^9）字节。指内存容量时，表示 1 073 741 824 个字节。

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。

结构化文本索引（structural text index）： 通过使用 DB2 UDB Text Extender，根据 XML 文档的树结构来对文本键建立索引。

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。

禁用（disable）： 通过除去在启用进程期间所创建的项，将数据库、文本表或文本列恢复为对 XML Extender 启用它之前的状态。

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 语句指定的主变量值会更改，但是该静态 SQL 语句不更改。

镜头目录（shot catalog）： 用来存储关于镜头的数据（例如，在视频剪辑中，镜头的开始帧号和结束帧号）的数据库表或文件。用户可以通过 SQL 查询来访问表视图，或者访问文件中的数据。

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 “root” 元素开始， “root” 元素是由正斜杠（/）或反斜杠（\）字符标识的。

〔K〕

开放式数据库连接（Open Database Connectivity）： 一种标准的应用程序编程接口（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编程接口，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 的调用级接口（CLI）规范，它是由 Digital Equipment Corporation（DEC）、Lotus、Microsoft 和 Sybase 公司开发的。与 *Java 数据库连接（Java Database Connectivity）* 对照。

可扩展样式表语言转换（XSLT）（Extensible Stylesheet Language Transformation（XSLT））： 用来将 XML 文档转换为其他 XML 文档的语言。XSLT 被用作 XSL 的一部分，它是 XML 的样式表语言。

可扩展样式表语言（XSL）（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。

〔L〕

连接视图（joined view）： 由 “CREATE VIEW” 语句创建的 DB2 UDB 视图，它将多个表连接在一起。

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。

列数据（column data）： 存储在 DB2 UDB 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）： 一种 Text Extender 功能，使您能够在计算机监视器上显示文本。请参阅 *Web 浏览器（Web browser）*。

浏览（browse）： 查看在计算机监视器上显示的文本。

路径表达式（path expression）： 请参阅*位置路径（location path）*。

〔M〕

命令行处理器（command line processor）： 即 DB2TX 程序，它:

允许您输入 Text Extender 命令

处理命令

显示结果。

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。

目录视图（catalog view）： 为了便于管理，Text Extender 所创建的系统表的视图。目录视图包含有关已经启用、以便供 Text Extender 使用的表和列的信息。

〔Q〕

启用（enable）： 准备数据库、文本表或文本列以供 XML Extender 使用。

千字节（KB）（kilobyte（KB））： 一千个（10^3）字节。指内存容量时，表示 1024 个字节。

嵌入式 SQL（embedded SQL）： 在应用程序中编码的 SQL 语句。请参阅*静态 SQL（static SQL）*。

强制类型转换函数（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，强制类型转换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。

缺省视图（default view）： 一种数据表示法，在这种表示法中，XML 表与其所有相关副表都互相连接。

缺省数据类型转换函数（default casting function）： 将 SQL 库基本类型强制转型为 UDT。

〔S〕

视频剪辑（video clip）： 用胶片拍摄下的或用录像磁带录制下的资料的一部分。

视频索引（video index）： 一个文件，Video Extender 用它来查找视频剪辑中特定的镜头或帧。

视频（video）： 属于记录下的信息中可以看的部分。

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。

数据库分区服务器（database partition server）： 管理*数据库分区（database partition）*。数据库分区服务器由数据库管理器，以及它管理的数据集合和系统资源组成。通常，会为每个机器指定一个数据库分区服务器。

数据库分区（database partition）： 数据库的一部分，由它自己的用户数据、索引、配置文件和事务作业记录组成。有时称为节点或数据库节点。

数据类型（data type）： 列和文字的属性。

数据流（data stream）： API 函数返回的信息，该信息中包含了其中有要搜索的术语的文本（至少一个段落），以及用来突出显示在该文本中找到的术语的信息。

数据源（data source）： 本地或远程关系或非关系数据管理器，它能够支持通过支持 ODBC API 的 ODBC 驱动程序来进行数据访问。

属性（attribute）： 请参阅 *XML 属性（XML attribute）*。

双字节字符大对象（DBCLOB）（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 UDB 数据库中。

搜索变元（search argument）： 在创建索引时指定的条件，由一个或几个搜索项以及搜索参数组成。

索引（index）： 从文本中抽取有效项，并将它们存储在文本*索引*。按键值逻辑顺序的一个指针集。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。

〔T〕

太字节（terabyte）： 一万亿（10^{12}）个字节。10 的 12 次方个字节。指内存容量时，表示 1 099 511 627 776 个字节。

条件（condition）： 选择 XML 数据的条件的规范，或连接 XML 集合表的方法的规范。

通配符字符（wildcard character）： 请参阅*屏蔽字符（masking character）*。

统一资源定位器（URL）（uniform resource locator（URL））： 一个地址，用来指定 HTTP Server 的名称并有选择地指定目录和文件的名称，例如，http://www.ibm.com/software /data/db2/extenders。

图像（image）： 图形的电子表示法。

〔W〕

外部文件（external file）： 一个文本文档，以文件的格式存储在操作系统的文件系统中，而不使用 DB2 控制之下的表单元的格式。文件系统中存在于 DB2 外部的文件。

外键（foreign key）： 一种键，它是引用约束的定义的一部分、并且由从属表的一列或多列组成。

谓词（predicate）： 搜索条件的一个元素，表示或暗示一个比较运算。

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 UDB 列（当定义 XML 列的创建索引方案时）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。

文本表（text table）： 包含*文本列*的一个 DB2 UDB 表。

文档访问定义（DAD）（Document Access Definition（DAD））： 用来定义 XML 列的创建索引方案或 XML 集合的映射方案。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（DTD）（Document type definition（DTD））： 对 XML 元素和属性的一组声明。DTD 定义在 XML 文档中使用哪些元素、可以按什么顺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。

文档（document）： 请参阅*文本文档（text document）*。

文件引用变量（file reference variable）： 一种编程变量，对于将 LOB 移至客户机工作站上的一个文件，或者从客户机工作站上的一个文件移出 LOB 是很有用的。

〔Y〕

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 资源库中。

应用程序编程接口（API）（application programming interface（API））：

（1）操作系统或可单独订购的许可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许可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）DB2 UDB Extender 提供了一些 API，用于请求用户定义的函数、管理操作、显示操作和视频画面切换检测。DB2 Text extender 提供了用于请求用户定义的函数、管理操作和信息检索服务的 API。在 DB2 中，函数位于接口中。例如，获取错误消息 API。

映射方案（mapping scheme）： 定义在关系数据库中如何表示 XML 数据。映射方案是在 DAD 中指定的。XML Extender 提供两种类型的映射方案: *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。

用户定义的单值类型（UDT）（user-defined distinct type（UDT））： 由 DB2 用户创建的数据类型，与 DB2 UDB 提供的数据类型（例如，LONG VARCHAR）相对。

用户定义的函数（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编程语言编写的，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 已知的另一内置函数或用户定义的函数来实现。此函数可以是标量函数或列（聚集）函数，并从一组值返回单一值（例如，MAX 或 AVG）。

用户定义的函数（UDF）（user-defined function（UDF））： 由 DB2 用户创建的 SQL 函数，与由 DB2 提供的 SQL 函数形成对照。Text Extender 以 UDF 格式提供了搜索函数，例如 CONTAINS。

用户定义的函数（UDF）（user-defined function（UDF））： 由 DB2 用户定义的函数。一旦定义，该函数就可以在 SQL 查询和视频对象中使用。例如，可以创建 UDF，以便获得视频的压缩格式或者返回音频的采样率。这提供了一种方法来定义特殊类型的对象的行为。

用户定义的类型（UDT）（user-defined type（UDT））： 一种数据类型，它对于数据库管理器来说，不是本机数据类型，该类型是用户创建的。请参阅*单值类型（distinct type）*。

用户定义的类型（UDT）（user-defined type（UDT））： 由 DB2 用户定义的一种数据类型。UDT 是用于区分 LOB 的。例如，可以为图像对象创建一个 UDT，而为音频对象创建另一个 UDT。尽管图像和音频对象都是作为 BLOB 来存储的，但是它们仍被作为不同于 BLOB 并且互不相同的类型来对待。

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。

语义索引（linguistic index）： 一种*文本索引*，它包含通过语义处理，已缩减至它们的基本部件格式的术语。例如，为 “Mice” 建立的索引为 “mouse”。另见*精确索引（precise index）*、*Ngram 索引（Ngram index）*和*双重索引（dual index）*。

元数据表（metadata table）： 请参阅*管理支持表（administrative support table）*。

元素（element）： 请参阅 *XML 元素（XML element）*。

〔Z〕

展开（expand）： 添加到从同义字派生的搜索项和附加项的操作。

兆字节（MB）（megabyte（MB））： 一百万个（10^6）字节。指内存容量时，表示 1 048 576 个字节。

主变量（host variable）： 嵌入式 SQL 语句中可以引用的应用程序中的变量。主变量是在数据库与应用程序工作区之间传输数据的主要机制。

主键（primary key）： 作为表定义一部分的唯一键。主键是引用约束定义的缺省父键。

转义字符（escape character）： 一个字符，指示不将后续的字符解释为*屏蔽字符*。

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（CLOB）（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 UDB 数据库中。

组成（compose）： 从 XML 集合中的关系数据中生成 XML 文档。

A

API: 请参阅*应用程序编程接口（application programming interface）*。

attribute_node: 元素的属性的一种表示法。

B

B 型树索引（B-tree indexing）： DB2 UDB 引擎提供本机索引方案。它构建采用 B 型树结构的索引项。支持 DB2 基本数据类型。

C

CCSID: 编码字符集标识。

CLOB: 字符大对象。

D

DAD: 请参阅*文档访问定义（Document access definition）*。

DBCLOB: 双字节字符大对象。

DBCS: 双字节字符支持。

DTD: (1) . (2) 请参阅*文档类型定义（Document type definition）*。

DTD 引用表（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 资源库（DTD repository）： 一种称为 DTD_REF 的 DB2 UDB 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。

DTD_REF 表（DTD_REF table）： DTD 引用表。

E

EDI: 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。

J

Java 数据库连接（JDBC）（Java Database Connectivity（JDBC））： 与开放式数据库连接（ODBC）具有相同特性的应用程序编程接口（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 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 公司以及各个合作伙伴和供应商开发的。

JDBC: Java 数据库连接。

L

LOB: 大对象。

LOB 定位器（LOB locator）: 存储在主变量中的小型（4 个字节）值，在程序中可以使用它来引用 DB2 UDB 数据库中的更大型 LOB。通过使用 LOB 定位器，用户可以操作 LOB，就象它存储在常规主变量中一样，而不需要在客户机和数据库服务器上的应用程序之间传输 LOB。

O

ODBC: 开放式数据库连接。

Q

QBIC 目录（QBIC catalog）: 一个资源库，用来保存有关图像的视觉特征的数据。

R

RDB_node: 关系数据库节点。

RDB_node 映射（RDB_node mapping）: XML 元素的内容或 XML 属性的值的位置，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）: 用来将所有副表与应用程序表关联起来的唯一标识。

S

SBCS: 单字节字符支持。

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。

T

text_node: 元素的 CDATA 文本的表示法。

U

UDF: 请参阅*用户定义的函数（user-defined function）*。

UDT: 请参阅*用户定义的类型（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。

URL: 统一资源定位器。

W

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。

X

XML: 可扩展标记语言。

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，它提供数据以组成 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。每个位置路径都可用为 XPath 定义的语法来表示。

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。

XML UDF: 由 XML Extender 提供的 DB2 UDB 用户定义的函数。

XML UDT: 由 XML Extender 提供的 DB2 UDB 用户定义的类型。

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 样式表语言转换。

索引

[F]

[G]

[H]

[J]

[K]

[L]

[M]

[Q]

[R]

[S]

[T]

[W]

[X]

E

F

G

I

J

M

O

R

S

T

U

W

X

声明

IBM 可能在所有国家或地区中不提供本文档中讨论的产品、服务或功能特性。有关您当前所在区域的产品和服务的信息，请向您当地的 IBM 代表咨询。任何对 IBM 产品、程序或服务的引用并非意在明示或暗示只能使用 IBM 的产品、程序或服务。只要不侵犯 IBM 的知识产权，任何同等功能的产品、程序或服务，都可以代替 IBM 产品、程序或服务。但是，评估和验证任何非 IBM 产品、程序或服务，则由用户自行负责。

IBM 公司可能已拥有或正在申请与本文档内容有关的各项专利。提供本文档并未授予用户使用这些专利的任何许可证。您可以用书面方式将许可查询寄往:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

有关双字节（DBCS）信息的许可查询，请与您所在国家或地区的 IBM 知识产权部门联系，或用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

本条款不适用于英国或任何这样的条款与当地法律不一致的国家或地区: 国际商业机器公司以“按现状”的基础提供本出版物，不附有任何形式的（无论是明示的，还是默示的）保证，包括（但不限于）对非侵权性、适销性和适用于某特定用途的默示保证。某些国家或地区在某些交易中不允许免除明示或默示的保证，因此本条款可能不适用于您。

本信息中可能包含技术方面不够准确的地方或印刷错误。此处的信息将定期更改；这些更改将编入本出版物的新版本中。IBM 可以随时对本出版物中描述的产品和／或程序进行改进和／或更改，而不另行通知。

本信息中对非 IBM Web 站点的任何引用都只是为了方便起见才提供的，不以任何方式充当对那些 Web 站点的保证。那些 Web 站点中的资料不是 IBM 产品资料的一部分，使用那些 Web 站点带来的风险将由您自行承担。

IBM 可以按它认为适当的任何方式使用或分发您所提供的任何信息而无须对您承担任何责任。

本程序的被许可方如果要了解有关程序的信息以达到如下目的: （i）允许在独立创建的程序和其他程序（包括本程序）之间进行信息交换，以及（ii）允许对已经交换的信息进行相互使用，请与下列地址联系:

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario

L6G 1C7
CANADA

只要遵守适当的条件和条款，包括某些情形下的一定数量的付费，都可获得这方面的信息。

本资料中描述的许可程序及其所有可用的许可资料均由 IBM 依据 IBM 客户协议、IBM 国际程序许可协议或任何同等协议中的条款提供。

此处包含的任何性能数据都是在受控环境中测得的。因此，在其他操作环境中获取的数据可能会有明显的不同。有些测量可能是在开发级的系统上进行的，因此不保证与一般可用系统上进行的测量结果相同。此外，有些测量是通过推算而估计的，实际结果可能会有差异。本文档的用户应当验证其特定环境的适用数据。

涉及非 IBM 产品的信息可从这些产品的供应商、其出版说明或其他可公开获得的资料中获取。IBM 没有对这些产品进行测试，也无法确认其性能的精确性、兼容性或任何其他关于非 IBM 产品的声明。有关非 IBM 产品性能的问题应当向这些产品的供应商提出。

所有关于 IBM 未来方向或意向的声明都可随时更改或收回，而不另行通知，它们仅仅表示了目标和意愿而已。

本信息可能包含日常商业运作所使用的数据和报告的示例。为了尽可能全面地作举例说明，这些示例包含个人、公司、品牌和产品的名称。所有这些名称都是虚构的，如与实际商业企业所使用的名称和地址有任何雷同，纯属巧合。

版权许可证:

本信息可能包括源语言形式的样本应用程序，这些样本说明不同操作平台上的编程方法。如果是为按照在编写样本程序的操作平台上的应用程序编程接口（API）进行应用程序的开发、使用、经销或分发为目的，您可以任何形式对这些样本程序进行复制、修改、分发，而无须向 IBM 付费。这些示例并未在所有条件下作全面测试。因此，IBM 不能担保或暗示这些程序的可靠性、可维护性或功能。

凡这些实例程序的每份拷贝或其任何部分或任何衍生产品，都必须包括如下版权声明:

©（*贵公司的名称*）（*年*）。此部分代码是根据 IBM 公司的样本程序衍生出来的。© Copyright IBM Corp.（*输入年份*）。All rights reserved.

商标

下列各项是国际商业机器公司在美国和 / 或其他国家或地区的商标，这些商标至少在 DB2 UDB 文档库中的一个文档中曾经使用过。

ACF/VTAM
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
BookManager
C Set++
C/370
CICS
Database 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Query Patroller
DB2 通用数据库
Distributed Relational Database Architecture
DRDA
eServer
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
iSeries
LAN Distance
MVS
MVS/ESA
MVS/XA
Net.Data
NetView
OS/390
OS/400
PowerPC
pSeries
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/400
SQL/DS
System/370
System/390
SystemView
Tivoli
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WebSphere
WIN-OS/2
z/OSzSeries

下列各项是其他公司的商标或注册商标，这些商标至少在 DB2 UDB 文档库中的一个文档中曾经使用过:

Microsoft、Windows、Windows NT 和 Windows 徽标是 Microsoft Corporation 在美国和 / 或其他国家或地区的商标。

Intel 和 Pentium 是 Intel Corporation 在美国和 / 或其他国家或地区的商标。

Java 和所有基于 Java 的商标是 Sun Microsystems, Inc. 在美国和 / 或其他国家或地区的商标。

UNIX 是 The Open Group 在美国和其他国家或地区的注册商标。

其他公司、产品或服务名称可能是其他公司的商标或服务标记。

程序号: 5722–DE1

中国印刷

S152-0843-00

Spine information:

IBM DB2 UDB iSeries 版　管理与编程 iSeries 版　版本 8