(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 1585945 B
(45)授权公告日 2011.05.18

(21)申请号 02822347.0
(22)申请日 2002.09.27
(30)优先权数据
60/326,052 2001.09.28 US
60/378,800 2002.05.07 US
(85)PCT申请进入国家阶段日
2004.05.11
(86)PCT申请的申请数据
PCT/US2002/030783 2002.09.27
(87)PCT申请的公布数据
WO2003/030031 EN 2003.04.10
(73)专利权人 甲骨文国际公司
地址 美国加利福尼亚州
(72)发明人 拉维·默西
穆拉利达尔·克里希纳普拉萨德
西瓦桑卡兰·钱德拉塞克
埃里克·塞德拉
维斯瓦纳坦·克里希纳穆尔蒂
尼普恩·阿加瓦尔
(74)专利代理机构 北京康信知识产权代理有限责任公司 11240
代理人 余刚
(51)Int.Cl.
G06F 17/30(2006.01)
(56)对比文件
WO 0159602 A,2001.08.16, 图6,摘要.
WO 0142881 A2,2001.06.14, 全文.
WO 0161566 A1,2001.08.23, 图1,摘要,说明书第18页1第6行-第19页第4行.
审查员 白雪涛

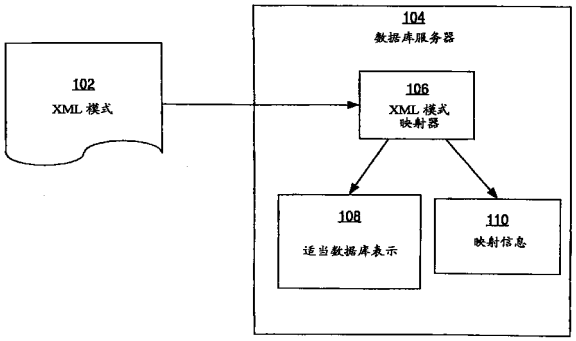
权利要求书 4 页 说明书 77 页 附图 7 页

(54)发明名称

用于将XML模式映射到对象关系数据库系统的机制

(57)摘要

本发明提供了一种方法和系统,允许用户将XML模式注册到数据库系统中。所述数据库系统基于已注册XML模式来确定如何在数据库系统内部保存那些符合所述XML模式的XML文档。这个确定包含了将XML模式中定义的结构映射成数据库系统所支持的结构。这种结构可以包括数据类型、元素之间的层次关系、约束条件、继承等等。一旦确定了映射,则数据库系统对其加以保存并且将其用于确定如何保存后续接收的那些符合已注册XML模式的XML文档。



CN 1585945 B

1. 一种方法，所述方法包括以下步骤：

数据库服务器确定一个适当的数据库表示，以便在所述数据库服务器内保存那些符合 XML 模式的文档；

所述数据库服务器生成表示所述 XML 模式的元素与所述适当数据库表示的元素之间的相关性的映射数据；

其中所述确定适当数据库表示的步骤包括将与所述 XML 模式中的元素关联的数据类型映射成所述数据库服务器支持的数据类型；

所述数据库服务器接收包括 XML 组件操作的数据库指令，所述 XML 组件操作在 XML 架构上执行；

至少部分基于所述映射数据，所述数据库服务器确定所述 XML 组件操作是否能够被转换为在所述适当的数据库表示的所述元素的至少一部分上的关系数据库操作；以及

如果确定所述 XML 组件操作能被转换为关系数据库操作，则所述数据库服务器将所述 XML 组件操作重新写为特定关系数据库操作，以及

至少部分基于所述适当的数据库表示，评估所述特定关系数据库操作。

2. 根据权利要求 1 所述的方法，其中：

所述确定适当数据库表示的步骤包括：基于用户规定的信息来确定将要把所述 XML 的元素映射成一个单独的 CLOB，而不在所述数据库服务器内为所述元素产生其它对象类型；以及

所述生成映射数据的步骤包括生成将所述元素映射成所述单个 CLOB 的数据。

3. 根据权利要求 2 所述的方法，还包括如下步骤：以用户规定的针对所述 XML 模式的注释的形式来接收所述用户规定的信息。

4. 根据权利要求 1 所述的方法，其中：

所述确定适当数据库表示的步骤包括基于用户规定的信息来确定所述 XML 模式的一个元素的第一组子元素将要映射成一个单独的 CLOB；以及

所述生成映射数据的步骤包括生成将所述第一组子元素映射成所述单个 CLOB 的数据，以及生成将所述元素的第二组子元素映射成一个或多个与所述 CLOB 不同的对象的数据。

5. 根据权利要求 1 所述的方法，其中所述确定适当数据库表示的步骤包括定义一个包含了与所述 XML 模式中的元素相对应的属性的 SQL 对象类型。

6. 根据权利要求 1 所述的方法，其中所述映射数据类型的步骤包括如下步骤：

如果与所述 XML 模式中的元素相关联的一个特定数据类型与第一长度相关联，则将所述特定数据类型映射成第一数据库数据类型；以及

如果所述特定数据类型与第二长度相关联，则将所述特定数据类型映射成第二数据库数据类型，其中所述第一数据库数据类型不同于所述第二数据库数据类型。

7. 根据权利要求 1 所述的方法，其中确定适当数据库表示的步骤包括：如果将所述特定元素定义成具有大于 1 的最大出现次数，则将所述 XML 模式的特定元素映射成所述数据库服务器支持的集合类型。

8. 根据权利要求 7 所述的方法，其中所述集合类型是数组类型，其中基于为所述特定数据库元素规定的最大出现次数来选择所述数组类型的基数。

9. 根据权利要求 1 所述的方法，其中所述限制确定适当数据库表示的步骤包括基于所述 XML 模式中为所述 XML 模式的元素规定的约束条件，在所述适当数据库表示中定义一个约束条件。

10. 根据权利要求 9 所述的方法，其中所述定义约束条件的步骤包括从包含以下约束条件的组中定义一个约束条件：唯一性约束、引用约束、以及非空约束。

11. 根据权利要求 1 所述的方法，其中：

第一数据类型与所述 XML 模式的一个元素相关联；

所述 XML 模式规定所述第一数据类型继承第二数据类型；以及

确定适当数据库表示的步骤包括在所述数据库服务器内定义一个对象类型的子类型，其中所述对象类型对应于所述第二数据类型。

12. 根据权利要求 1 所述的方法，其中所述确定适当数据库表示的步骤包括：

将所述 XML 模式中的第一组元素映射成使所述第一组中分离的每个元素保持与所述第一组中的其它元素相分离的数据库结构；以及

将所述 XML 模式中的第二组元素映射成一个数据库结构，在所述数据库结构中所述第二组元素中的所有元素都被组合成作为一个单独的无差别数据库元素。

13. 根据权利要求 12 所述的方法，其中所述数据库服务器基于与所述 XML 模式相关联的指示来确定所述第一组的成员以及所述第二组的成员。

14. 根据权利要求 12 所述的方法，其中：根据选择到所述第一组中的元素与选择到所述第二组元素中的元素相比将会得到更频繁访问的可能性，而将所述第一组元素中的元素选择到所述第一组中。

15. 根据权利要求 1 所述的方法，其中：

确定适当数据库表示和生成映射数据的步骤是作为 XML 模式注册操作的一部分执行的，其中所述操作将会导致在所述数据库服务器内部进行修改；以及

所述方法还包括对在所述 XML 模式注册操作中遭遇到特定差错做出响应而自动删除所述 XML 模式注册操作引起的所有修改的步骤。

16. 根据权利要求 1 所述的方法，其中所述确定适当数据库表示的步骤包括确定如何中断所述 XML 模式中的循环。

17. 根据权利要求 1 所述的方法，其中：

所述 XML 模式包括一个包含了多个组件的循环；以及

所述确定如何中断循环的步骤包括使得所述循环定义的每个分量都保持一个针对其所有子组件的指针。

18. 根据权利要求 1 所述的方法，其中确定如何中断循环的步骤包括为了进行存储而导致将整个循环定义作为数据库服务器内部的单个 CLOB 来进行映射。

19. 根据权利要求 1 所述的方法，其中所述生成映射信息的步骤包括向所述 XML 模式添加注释，以及在所述数据库服务器内部保存所述带有注释的 XML 模式。

20. 根据权利要求 1 所述的方法，还包括如下步骤：

基于所述适当数据库表示而在数据库内部创建结构；以及

将来自那些符合所述 XML 模式的 XML 文档的数据保存在所述结构中。

21. 根据权利要求 20 所述的方法，其中所述保存来自 XML 文档的数据的步骤包括如下

步骤：

在所述数据库服务器处接收一个 XML 文档；

从所述 XML 文档识别与所述 XML 模式的单个元素关联的数据；

基于与所述数据关联的所述元素和所述映射数据，将与单个元素相关联的数据保存在所述结构内部的位置上。

22. 根据权利要求 1 所述的方法，还包括以下步骤：在所述数据库服务器内部验证所述 XML 模式，以便确定所述 XML 模式是否符合一个用于 XML 模式的 XML 模式。

23. 根据权利要求 1 所述的方法，其中所述确定步骤是作为 XML 模式注册操作的一部分来执行的，所述操作是响应于在所述数据库服务器上接收到所述 XML 模式而发起的。

24. 根据权利要求 1 所述的方法，其中所述确定步骤是作为 XML 模式注册操作的一部分来执行的，所述操作是响应于在数据库服务器上接收到一个符合所述 XML 模式的 XML 文档而发起的。

25. 根据权利要求 23 所述的方法，其中：

所述 XML 模式包含了用户规定的注释，所述注释指示的是数据库服务器应该如何映射至少一个 XML 模式的元素；以及

至少一部分映射数据反映了所述用户规定的注释。

26. 根据权利要求 1 所述的方法，所述确定所述 XML 组件操作是否能够被转换为关系数据库操作的步骤还包括确定所述 XML 组件操作中包含的 XPath 表达是否至少为以下之一：

简单的 XPath 表达，仅沿着子轴和属性轴上的 XML 节点进行，其中每个 XML 节点对应于目标型列或标量型列；

集合遍历表达，仅沿着子轴和属性轴上的 XML 节点进行，其中，至少一个 XML 节点对应于集合类型列；

通配符轴表达，用于使 XML 节点能够被全部强制成相同的数据类型；以及

派生轴表达，用于使 XML 节点能够被全部强制成相同的数据类型。

27. 根据权利要求 1 所述的方法，所述确定所述 XML 组件操作是否能够被转换为关系数据库操作的步骤还包括确定所包含的 XPath 表达是否被结构化查询语言 (SQL)/XML 函数使用。

28. 根据权利要求 1 所述的方法，其中所述 SQL/XML 函数是 EXISTNODE、EXTRACT，以及 EXTRACTVALUE 中的至少一个。

29. 根据权利要求 1 所述的方法，所述确定所述 XML 组件操作是否能够被转换为关系数据库操作的步骤还包括确定所包含的 XPath 表达是否被 SELECT 列表、WHERE 语句谓语、GROUPBY 表达、ORDER BY 表达、FROM 语句、以及 HAVING 语句中的至少一个的 SQL 查询中的所述 SQL/XML 函数使用。

30. 根据权利要求 1 所述的方法，所述确定所述 XML 组件操作是否能够被转换为关系数据库操作的步骤还包括确定所包含的 XPath 表达是否被 SQL CREATE INDEX 命令的 INDEX 语句中的所述 SQL/XML 函数使用。

31. 根据权利要求 1 所述的方法，所述重写所述 XML 组件操作的步骤还包括将简单的 XPath 遍历重写成 SQL 目标型存取器和 SQL 标量型存取器中的至少一种。

32. 根据权利要求 1 所述的方法，所述重写所述 XML 组件操作的步骤还包括将所包含的

XPath 表达中的谓语重写成 SQLWHERE 语句中的谓语。

33. 根据权利要求 1 所述的方法，所述重写所述 XML 组件操作的步骤还包括将所述数据库命令中的 XPath 表达的 EXISTSNODE 函数重写为目标类型的 IS NOT NULL 测试，对应于 XPath 表达的目标。

34. 根据权利要求 1 所述的方法，所述重写所述 XML 组件操作的步骤还包括将集合的 XPath 遍历重写为对应于所述集合的集合表上的子查询。

35. 根据权利要求 1 所述的方法，所述确定所述 XML 组件操作是否能够被转换的步骤还包括将所述 XML 组件扩展为 XPath 操作器的树，所述操作器中的每个都表示包含在所述 XML 组件操作中的 XPath 表的中的一个位置步骤。

36. 根据权利要求 1 所述的方法，所述重写所述 XML 组件操作的步骤还包括将所述集合的 XPath 遍历中的谓语增加至所述集合表上的子查询的 WHERE 语句。

用于将 XML 模式映射到对象关系数据库系统的机制

[0001] 相关申请

[0002] 本申请要求享有以下美国临时专利申请的优先权，这些申请的全部内容结合于此，作为参考：

[0003] 2001年9月28日由Eric Sedlar和Viswanathan Krishnamurthy提交的名为File Based Access Provided With a Database System的美国临时专利申请60/326,052；

[0004] 2002年5月7日由Nipun Agarwal、Ravi Murthy、Eric Sedlar、Sivasankaran Chandrasekar、Fei Ge、 Syam Pannala、Neema Jalali以及Muralidhar Krishnaprasad提交的名为SQL Access to Data thatProvides a File System Abstraction的美国临时专利申请60/378,800。

[0005] 本申请还涉及以下美国专利申请，这些申请的全部内容结合于此作为参考：

[0006] 与此同日由Nipun Agarwal、Ravi Murthy、Eric Sedlar、Sivasankaran Chandrasekar以及Fei Ge提交的名为OPERATORSFOR ACCESSING HIERARCHICAL DATA IN A RELATIONALSYSTEM的美国专利申请__ ___(律师案卷号50277-1975)；

[0007] 与此同日由Nipun Agarwal、Eric Sedlar、Ravi Murthy以及NamitJain提交的名为PROVIDING A CONSISTENT HIERARCHICALABSTRACTION OF RELATIONAL DATA的美国专利申请_____(律师案卷号50277-1976)；

[0008] 与此同日由Nipun Agarwal、Eric Sedlar以及Ravi Murthy提交的名为INDEXING TO EFFICIENTLY MANAGE VERSIONEDDATA IN A DATABASE SYSTEM的美国专利申请_____(律师案卷号50277-1978)；

[0009] 与此同日由Ravi Murthy、Eric Sedlar、Nipun Agarwal以及Neema Jalali提交的名为MECHANISMS FOR STORING CONTENTAND PROPERTIES OF HIERARCHICALLY ORGANIZEDRESOURCES的美国专利申请_____(律师案卷号50277-1979)；

[0010] 与此同日由Ravi Murthy、Eric Sedlar、Nipun Agarwal、SamIdicula以及Nicolas Montoya提交的名为MECHANISM FORUNIFORM ACCESS CONTROL IN A DATABASE SYSTEM的美国专利申请__(律师案卷号50277-1980)；

[0011] 与此同日由Syam Pannala、Eric Sedlar、Bhushan Khaladkar、Ravi Murthy、Sivasankaran Chandrasekar以及Nipun Agarwal提交的名为LOADABLE UNITS FOR LAZY MANIFESTATION OF XMLDOCUMENTS的美国专利申请__(律师案卷号50277-1981)；

[0012] 在同一日由Neema Jalali、Eric Sedlar、Nipun Agarwal、and RaviMurthy提交的名为MECHANISM TO EFFICIENTLY INDEXSTRUCTURED DATA THAT PROVIDES HIERARCHICAL ACCESSIN A RELATIONAL DATABASE SYSTEM的美国专利申请__(律师案卷号50277-1982)。

技术领域

[0013] 本发明涉及用于在数据库系统中保存XML数据的技术。

背景技术

[0014] 在关系数据库系统内部，数据保存在各种类型的数据容器中。这种数据容器通常具有一种结构。而所述容器结构则施加于其包含的数据。举例来说，表格将被组织到行和列中。当在表格中保存数据的时候，数据内部的单独数据项保存在特定的行与列中，由此向所述数据施加了一种结构。

[0015] 通常，施加到数据上的结构与数据内部的逻辑关系相对应。举例来说，保存在表格中指定的行的内部的所有值通常彼此具有某种逻辑关系。例如，雇员表格给定行内部的所有值可以对应于同一雇员。

[0016] 在数据库系统外部，结构化电子数据的程度可以基于数据特性而发生很大变化。举例来说，电子表格中保存的数据通常是高度结构化的，而表示可视图像的数据则通常是高度无结构的。

[0017] XML（可扩展标记语言）正作为一种描述和保存所有数据形式的格式而逐渐得到普及。因此，对当前的数据管理系统来说，为保存、搜索和操作 XML 文档提供支持是一个极其重要的问题。

[0018] 在名为“XML 模式”中可以对与特定类型的 XML 文档结构有关的信息加以规定。举例来说，关于特定类型的 XML 文档的 XML 模式可以规定特定类型的 XML 文档中包含的数据项的名称，所述类型的 XML 文档中包含的数据项之间的层次关系，特定类型的 XML 文档中包含的数据项的数据类型等等。

[0019] 然而不幸的是，尽管构造了 XML 文档，但在使用数据库系统来保存 XML 文档的时候，数据库系统在很大程度上忽视了 XML 文档的结构。举例来说，高度结构化的 XML 文档包含了关于多种属性的多个值，这种文档可以像表格中单独的 CLOB 列的原子型无差别数据一样简易的加以保存。而在以这种方式保存 XML 文档的时候则无法彻底利用数据库性能和可扩展性来访问 XML 数据。

附图说明

[0020] 本发明是借助实例来描述的，但这并不作为限制，在附图的图形中，相同的标号代表相同的部件，其中：

[0021] 图 1 是一个包含了依照本发明实施例而将 XML 模式中包含的结构映射成对象关系结构的机制的框图；

[0022] 图 2 是一个描述了实施本发明实施例的计算机系统的框图；

[0023] 图 3 是一个显示了依照本发明实施例来创建一个 XML 类型表格的语法的框图；

[0024] 图 4 是一个对依照本发明实施例配置并为符合特定 XML 模式的文档创建适当数据库表示的数据库对象的数据库系统；

[0025] 图 5 是一个显示了有选择地将 XML 串映射成两个备选数据库所支持的数据类型的框图；

[0026] 图 6 显示了是一个为了进行离线 (out-of-line) 存储而被映射成 SQL 的 complexType（复杂类型）；

[0027] 图 7 显示的是映射成字符型大对象 (CLOBs) 的 complexTypeXML 分段；

[0028] 图 8 显示的是同一 XML 模式中的 complexType 类型之间的交叉引用；

[0029] 图 9 是一个显示 XML 模式内部的 complexType 自引用的框图；以及

[0030] 图 10 是显示 XML 模式之间的循环引用的框图。

具体实施方式

[0031] 在这里描述了一种用于将 XML 模式映射成对象关系数据库系统的方法和系统。在以下描述中，出于说明的目的而对许多细节进行了阐述，从而提供了关于本发明的全面理解。然而很明显，本发明可以在不具备这些特定细节的情况下得到实施。在其他情况下，为了避免不必要地造成本发明不清楚，众所周知的结构和设备是以框图形式显示的。

[0032] 功能概述

[0033] 在这里描述了多种技术，用于以一种提高保持数据的数据库容器施加于数据的结构与产生数据的 XML 文档的结构之间的相关性的方式来对数据库系统内部的 XML 数据进行管理。一方面，在这里提供了一种允许数据库系统的用户将 XML 模式注册到数据库系统的机制。而 XML 模式则既可以显性注册（借助于 API 调用）也可以隐性注册（在将一个符合 XML 模式的实例文件首次插入数据库的时候）。

[0034] 在针对一个给定 XML 模式的登记处理中，数据库系统确定 (1)XML 模式的适当数据库表示以及 (2) 映射信息。“适当数据库表示”的确定是一个关于数据库系统如何映射那些符合 XML 模式的数据的判定。举例来说，为给定的 XML 模式确定适当数据库表示可以包括确定数据库系统用以保存来自那些来源于符合给定的 XML 模式的 XML 文档的数据库对象、集合类型、约束条件乃至索引。

[0035] 映射信息表示的是 XML 模式包含的结构与适当数据库表示包含的结构之间的映射。举例来说，所述映射信息可以表示应该将与 XML 模式的特定元素相关联的数据保存在表格中的特定的行，其中所述表格是作为适当数据库表示的一部分而产生的。为了在 XML 模式中描述的结构与保存 XML 数据的数据库容器施加于数据的结构之间创建高度的相关性，通常会产生适当数据库表示和映射信息。

[0036] 系统概述

[0037] 图 1 是一个系统框图，其中包含了一种用于将 XML 模式映射成对象关系数据库系统的机制。具体地说，数据库服务器 104（在这里也称为“XDB”）包括一个 XML 模式映射器 106。在向数据库服务器 104 注册 XML 模式 102 时，XML 模式映射器 106 为符合 XML 模式 102 的文档确定适当数据库表示，并且产生那些表示 XML 模式的元素与适当数据库表示 108 的元素之间的相关性的映射信息 110。

[0038] 根据一个实施例，数据库服务器 104 被配置成：

[0039] ●注册任何符合 W3C 的 XML 模式

[0040] ●相对一个已注册 XML 模式来执行 XML 文档验证

[0041] ●注册局部和全局模式

[0042] ●从对象类型中产生 XML 模式

[0043] ●支持重新注册一个 XML 模式（作为一种用于手动模式演化的机制）

[0044] 在经由某些 APIs（例如 FTP、HTTP）插入文档的时候支持 XML 模式的隐性注册

[0045] ●允许用户引用另一个用户所拥有的模式

[0046] ●当存在具有同一名称的局部模式时，允许用户显性引用一个全局模式

[0047] ●支持 XML 模式演化

[0048] 根据一个实施例，XML 模式映射器 106 被配置成：

[0049] ●从 XML 模式中产生结构化数据库映射（通常是在模式注册中）

[0050] - 举例来说，这其中可以包括产生 SQL 对象类型、集合类型等等以及借助模式注释来捕获映射信息。

[0051] ●当存在多个合法映射的时候，允许用户规定某个 SQL 类型映射

[0052] ●基于已注册 XML 模式来创建 XMLType 表格和列

[0053] ●为基于模式的 XMLType 表格提供 DML 和查询支持

[0054] XML 模式注册

[0055] 根据一个实施例，当在数据库服务器 104 内部可以使用或引用一个 XML 模式之前，首先必须将所述 XML 模式注册到数据库服务器 104。在完成所述注册处理之后，数据库服务器 104 可以对符合这种模式（以及经由文档内部的模式 URL 来对其加以引用）的 XML 文档进行处理。此外还可以为这种模式所定义的根 XML 元素创建表格和 / 或列，以便保存这个相容文档。

[0056] 根据一个实施例，一个模式是通过规定模式文档及其 URL（也称为模式位置）而使用一个 DBMS_XMLSCHEMA 包来注册的。应该指出的是，这里使用的 URL 只是唯一识别数据库内部已注册模式的名称 - 不必是模式文档所在的物理 URL。此外，所述模式的目标命名空间是规定了一个在内部声明了元素和类型的“抽象”命名空间的另一个 URL（与模式位置 URL 不同）。实例文档则应该规定根元素的命名空间并规定这个元素的模式位置 (URL)。

[0057] 举例来说，对如下 XML 模式加以考虑。该模式声明了一个名为“PurchaseOrderType”的 complexType（复杂类型）和一个所述类型的元素“PurchaseOrder”。

[0058] <schema xmlns =" http://www.w3.org/2001/XMLSchema"

[0059] targetNamespace =" http://www.oracle.com/PO.xsd" >

[0060] <complexType name =" PurchaseOrderType" >

[0061] <attribute name =" PurchaseDate" type =" date" />

[0062] <sequence>

[0063] <element name =" PONum" type =" decimal" />

[0064] <element name = " Company " type = " string " maxLength =" 100" />

[0065] <element name =" Item" maxOccurs =" 1000" >

[0066] <complexType>

[0067] <sequence>

[0068] <element name =" Part" type =" string" maxLength =" 1000" >

[0069] <element name =" Price" type =" float" />

[0070] </sequence>

[0071] </complexType>

[0072] </element>

[0073] </sequence>

[0074] </complexType>

[0075] <element name ＝" PurchaseOrder" type ＝" PurchaseOrderType" />

[0076] </schema>

[0077] 以下语句在URL“http://www.oracle.com/PO.xsd”注册了这个模式(doc是一个保持上述模式文本的变量)。

[0078] dbms_xmlschema.registerSchema(' http://www.oracle.com/PO.xsd' ,doc);

[0079] 如下文中更详细描述的那样,可以使用已注册XML模式来创建基于模式的XMLType表格和列。以下是一个符合上述XML模式的XMLType实例。其中schemaLocation属性规定了模式URL。

[0080] <PurchaseOrder xmlns ＝" http://www.oracle.com/PO.xsd"

[0081] xmlns:xsi ＝" http://www.w3.org/2001/XMLSchema-instance"

[0082] xsi:schemaLocation ＝" http://www.oracle.com/PO.xsd

[0083] http://www.oracle.com/PO.xsd"

[0084] PurchaseDate ＝" 01-JAN-2001" >

[0085] <PONum>1001</PONum>

[0086] <Company>Oracle Corp</Company>

[0087] <Item>

[0088] <Part>9i Doc Set</Part>

[0089] <Price>2550</Price>

[0090] </Item>

[0091] </PurchaseOrder>

[0092] 根据本发明的一个实施例,XML模式注册包含了(1)模式验证,(2)确定适当的数据结构,以及(3)生成映射信息。下文对其中每一个阶段都进行了详细描述。

[0093] XML模式验证

[0094] XML模式描述的是特定类型的XML文档结构。然而,XML模式自身必须是符合XML模式所规定结构的XML文档。具体地说,每个XML模式必须符合与XML模式文档类型相关的XML模式文档中描述的结构。在XML模式注册的模式验证阶段将对正在注册的XML模式进行检查,以便验证XML模式是否符合与XML模式文档类型相关的XML模式中规定的结构。

[0095] 判定适当的数据库表示

[0096] 如上所述,适当数据库表示做出的是数据库系统应该如何管理符合XML模式的判定。根据一个实施例,在这里选择了一个适当数据库表示,以便实现(1)包含数据的XML文档施加于数据的结构与(2)数据库系统施加于数据的结构之间的高度相关性。

[0097] 实现高度相关性的能力至少部分依赖于数据库系统性能。数据库系统的具体性能随厂商和版本而不同。虽然某些能力为大多数数据库系统所共有,但其他性能却并非如此。因此,虽然这里是在具有一组特定能力的数据库系统的环境中描述本发明实施例的,但是本发明并不局限于具有这些特定能力的数据库系统。

[0098] 根据一个实施例,确定适当数据库表示是基于一组用于管理XML模式映射器106的操作的通用规则来进行的,所述规则涉及的是如何将XML模式中可能遇到的各种结构类型映射成目标对象关系数据库系统支持的相应结构。这些规定可以固定编码到XML模式映

射器 106 的逻辑电路中，也可以在 XML 模式映射器 106 使用的元数据中加以表示。根据一个实施例，所述通用规则旨在解决以下问题：

[0099] ●如何将 XML 支持的数据类型映射成目标对象关系数据库系统支持的数据类型；

[0100] ●如何将 XML 模式定义的结构映射成具有相似结构的数据库对象；

[0101] ●如何将 XML 支持的约束条件映射成目标对象关系数据库系统支持的约束条件执行机制；

[0102] ●如何在目标对象关系数据库系统中反映从另一个 XML 模式那里继承了 XML 模式；以及

[0103] 如何在目标对象关系数据库系统中反映 XML 支持的其它构造，例如替换群、简单内容、通配符、借助于包括和引入元素等等的外部引用。

[0104] 将 XML 数据类型映射成对象关系数据类型

[0105] XML 模式声明了一个基本类型集合。根据一个实施例，XML 模式映射器 106 使用的规则定义了目标数据库系统支持并与每种 XML 数据类型相对应的数据类型。例如在一个实施例中，XML 数据类型“string”映射成了 VARCHAR 或 CLOB SQL 数据类型之一。举例来说，在这个实例中，XML 模式映射器 106 可以选择是否根据可以在 XML 模式中为串元素声明的任何长度约束条件而将某个串元素映射成一个 VCHAR 或 CLOB。在下文中给出并且在附录 I 中描述了 XML 模式映射器 106 可用的数据类型 - 数据类型的映射规则。

[0106] 将 XML 结构映射成数据库对象

[0107] SQL 模式是根据可能在其内部出现的元素和属性来对元素结构进行描述的。将 XML 结构映射成数据库对象的规则表示的是如何映射一个具有与 XML 模式内部定义的 XML 属性和元素相对应的属性的 SQL 对象类型。例如，包含属性 X 以及元素 Y 与 Z 的 XML 元素 A 将会映射成具有三个属性 X、Y 和 Z 的对象类型。

[0108] 将 XML 约束条件映射到数据库约束条件

[0109] XML 模式可以规定不同形式的约束条件。当 XML 模式映射器 106 遇到这类约束条件时，所述约束条件将会映射成 SQL 中的适当约束条件机制。举例来说，XML 模式中定义的“string”属性的长度约束条件可以是 maxLength =“20”。根据一个实施例，这种约束条件导致将串属性映射成数据类型 VARCHAR2(20)。

[0110] 另一种适用于 XML 元素的约束条件是一个规定元素的最大出现次数的约束条件。当所述最大次数大于 1 时，所述元素可以映射成目标数据库系统（例如 VARRAY）支持的一个数组类型。而为 XML 约束条件规定的出现次数表示的则是 VARRAY 的基数。

[0111] 可以为 XML 模式元素规定并在适当数据库表示的对应约束条件中反映的其它类型的约束条件包括单值性限制、引用完整性限制、非零限制等等。

[0112] 映射继承

[0113] XML 模式模型考虑到了复杂类型的继承。根据一个实施例，当 XML 模式使用继承结构时，所述继承将会映射成目标数据库系统支持的 SQL 目标继承机制。例如在 XML 模式内部，XMLcomplexType “USAddress”可以作为另一个 complexType “Address”的扩展而得到声明。与之对应的是，在适当数据库表示内部，SQL 对象类型“USAddress”是作为一个与“Address”相对应的 SQL 对象类型的子类型而得到声明的。

[0114] 局部和全局模式

[0115] 作为默认情况，XML 模式归属于执行注册的用户。针对 XML 模式的引用则保存在目录 /sys/schemas/<username>/... 内部的 XDB 层次中。例如，如果用户 SCOTT 注册上述模式，则所述模式映射成文件 /sys/schemas/SCOTT/www.oracle.com/PO.xsd。这种模式称为局部模式。通常，它们只能由拥有它们的用户所使用。需要指出，由于出现在实例 XML 文档中的模式位置只是 URLs，因此在这里并没有使用数据库用户名来限制模式 URL 的意图。因此，只有模式拥有者才可以在定义 XMLType 表格、列或视图以及确认文档等等的过程中使用所述模式。

[0116] 与局部模式相反，特许用户可以通过将自变量规定成 dbms_xmlschema 注册函数来将 XML 模式注册成一个全局模式。全局模式可以为所有用户所看到并且保存在 XDB 层次内部的目录 under/sys/schemas/PUBLIC/... 之下。应该指出的是，针对这个目录所进行的访问受控于 ACLs- 作为默认情况，所述目录只能由 DBA 写入。用户需要具有这个目录的写入权限才能够注册全局模式。

[0117] 用户可以将一个具有同一 URL 的局部模式注册成一个现有全局模式。局部模式则始终隐藏了具有同一名称 (URL) 的任何全局模式。

[0118] 用户可以注册一个针对某些其他用户可能拥有的现有模式的链接。所述模式链接由其 URL 标识。然后，无论何时，只要预期得到一个模式 URL，例如创建一个 xmltype 表格，则可以使用所述模式链接 URL。在进行引用的时候，针对模式链接的引用将会转换成基础模式。如果用户具有名称与全局模式相同的局部模式，则存在一种允许用户显性引用全局模式的机制。用户可以注册一个针对（具有不同名称）全局模式的链接。

[0119] 删除 XML 模式

[0120] 依照一个实施例，可以通过使用 dbms_xmlschema.deleteSchema 过程来删除 XML 模式。当用户尝试删除模式时，数据库服务器首先检查其从属对象。如果存在任何从属对象，则数据库服务器产生一个差错并且删除操作失败。在删除模式的时候还提供了一个 FORCE（强制）选项 -- 如果用户规定了 FORCE 选项，那么即使没有完成从属对象检查，所述模式删除也会继续进行。在这种方式下，模式删除会将其从属对象全部标记为无效。

[0121] XML 模式的从属对象模型

[0122] 根据一个实施例，以下对象“依赖”于已注册 XML 模式：

[0123] ●具有一个符合这个模式中的某些元素的 XMLType 列的表格 / 视图。

[0124] ●作为定义的一部分而包含或引入了这个模式的 XML 模式

[0125] ●引用了例如 XMLGEN 运算符内部的模式名称的游标 (cursor)。（注意：这些都是完全暂态的对象）

[0126] 以下操作产生了附加于 XML 模式对象的从属对象：

[0127] ●模式注册：将从属对象添加到全部的包含 / 引入的模式上创建表格 / 视图 / 游标：向所引用的 xml 模式对象添加那些来自表格 / 视图 / 游标的从属对象。

[0128] 事务行为

[0129] 根据一个实施例，与其他 SQL DDL 操作相似，模式注册是非事务性的和自动提交的。如果注册奏效，则自动提交所述操作。然而如果注册失败，那么数据库将会回滚到注册开始之前的状态。由于模式注册处理可能包括创建对象类型和表格，因此错误恢复包括撤消任何那些如此创建的表格和类型。由此确保了整个模式注册是原子类型的，也就是说，所

述注册要么成功，要么将数据库恢复到注册开始之前的状态。

[0130] XML 模式演化

[0131] 用户可以通过重新注册一个已注册 XML 模式和提供新的 XML 模式文档来演化所述 XML 模式。此外可以使用 dbms_xmlschema.registerSchema 函数来重新注册 XML 模式。如果没有依赖于这个模式的 XMLType 表格（XMLType 视图得到认可），则这个操作始终是成功的。根据一个实施例，如果存在任何从属的 XMLType 表格，则数据库服务器 104 要求输入模式文档包含完整的 SQL 映射注释并且它们表示的是一个适合所有这种 XMLType 表格的有效映射。

[0132] 实例 -- 改变元素或属性名称：用户检索已注册模式文档，做出需要的修改并且重新注册所述模式文档。应该指出的是，这种变化不会影响到基础表格。

[0133] 实例 -- 添加一个新元素或属性：由于这种变化影响到基础表格，因此必须在多个步骤中执行所述变更。用户首先使用 ALTERTYPE 和 / 或 ALTER TABLE 命令来演化基础表格。这其中将 XML 模式标记为无效。然后，用户酌情修改 XML 并且对其进行重新注册。

[0134] 根据一个实施例，在这里提供了一种一个步骤的 XML 模式演化，也就是用户只输入一个新的 XML 模式并且所有基础类型和表格的变更都是隐性确定的。

[0135] XML 模式的隐性注册

[0136] 在经由 HTTP 或 FTP 之类的协议将实例文档插入 XDB 时，如果还未注册的话，那么它们所符合（如果规定的话）的模式是隐性注册的。由于模式注册始终是自动提交的，因此隐性注册是在一个自主事务内部执行的。

[0137] XMLTYPE 表格

[0138] 在这里将作为 XML 模式的“适当数据库表示”一部分的表格和列称为“基于模式”的表格和列。根据一个实施例，基于模式的 XMLTYPE 表格和列可以通过引用（已注册模式的）模式 URL 与根元素名称来创建。此外还可以在提供包含了模式位置和元素名称的单个 URL 的过程中使用 XPointer 符号（如下所示）的子集。

[0139] CREATE TABLE po_tab OF xmltype

[0140] XMLSCHEMA" http://www.oracle.com/PO.xsd" ELEMENT

[0141] " PurchaseOrder"

[0142] 一个等价的定义是

[0143] CREATE TABLE po_tab of xmltype

[0144] element" http://www.oracle.com/PO.xsd#PurchaseOrder" ;

[0145] 作为默认情况，基于模式的 XMLType 保存在一个基础（隐藏）对象类型的列中。SQL 对象类型可以在模式注册处理过程中（随意）创建。XML 到 SQL 对象类型和属性的映射本身是作为附加注释保存在 XML 模式文档内部的，也就是 XDB 定义的新属性。

[0146] 基于模式的 XMLType 还可以保存在单个基础 LOB 列中。

[0147] CREATE TABLE po_tab OF xmltype

[0148] STORE AS CLOB

[0149] ELEMENT" http://www.oracle.com/PO.xsd#PurchaseOrder" ;

[0150] 创建 SQL 对象类型

[0151] 依照一个实施例，在注册一个 XML 模式时，数据库服务器 104 创建的是允许结构化

保存符合这个模式的 XML 文档的适当 SQL 对象类型。所有 SQL 对象类型都是在当前用户的模式（作为默认）中创建的。举例来说，在注册 PO.xsd 的时候将会创建以下的 SQL 类型。

[0152] create type Item_t as object

[0153] (

[0154] part varchar2(1000),

[0155] price number

[0156]);

[0157] create type Item_varray_t as varray(1000)of OBJ_T1;

[0158] create type PurchaseOrder_t as object

[0159] (

[0160] purchasedate date,

[0161] ponum number,

[0162] company varchar2(100),

[0163] item Item_varray_t

[0164]);

[0165] 实际上，上述对象类型和属性的名称可以由系统产生。如果所述模式已经包含了所填充的 SQLName 属性，则将这个名称用作对象属性名称。否则，所述名称来源于 XML 名称 -- 除非由于长度或冲突原因而不能使用所述名称。如果填充了 SQLSchema 属性，那么 Oracle 将会尝试在规定模式中创建所述类型。当前用户则必须具有执行这个操作的必要权限。

[0166] 将 XML 模式映射到对象类型 -- 详细实例

[0167] 以下章节提供了如何从 XML 模式信息中产生 SQL 对象类型的细节。如上所述，实际映射规则可能因为多种因素而随实施方式发生变化。其中一个因素是目标数据库系统的能力。在以下的详细实例中，假设目标数据库系统支持那些当前在可以从 Oracle 公司购买的 Oracle 9iR2 中使用的数据类型和对象类型机制。

[0168] 映射简单类型

[0169] 根据一个实施例，XML 基本类型映射成了最接近的 SQL 数据类型。例如，十进制、正整数和浮点数全都映射成 SQL 数字。XML 枚举类型则映射成具有单个 RAW(n) 属性的对象类型 --n 的值由枚举声明中的可能值确定。XML 列表或并集数据类型则映射成 SQL 中的串 (VARCHAR2/CLOB) 数据类型。

[0170] XML 简单类型与 SQL 的缺省映射

[0171]

XML 简单类型	长度/精度	最大长度（MaxLength）/范围	缺省的 Oracle 数据类型	兼容的数据类型	注释
串（string）		n（n<4000）	VARCHAR2（n）	NVARCHAR2 CHAR、CLOB NCHAR CLOB	对 UTF-16 编码来说，n<2000
串	m（m<4000）		CHAR（n）	VARCHAR2 NVARCHAR2 CLOB NCHAR NCLOB	对 UTF-16 编码来说，n<2000
串	m（m>4000）		CLOB	VARCHAR2 NVARCHAR2 CHAR NCHAR NCLOB	对 UTF-16 编码来说，n>2000
串		N（n>4000）	CLOB	VARCHAR2 NVARCHAR2 CHAR NCHAR NCLOB	对 UTF-16 编码来说，n>2000
布尔值			RAW（1）		值必须为 0、1
浮点型			FLOAT	NUMBER DOUBLE	
双精度型			DOUBLE	NUMBER	
小数	精度 m	范围 n	NUMBER（m,n）		如果 m&n==0，则映射成 NUMBER
时间			TIMESTAMP	TIME??	XML 中的时间戳可以映射成 CCYY-MM-DD Thh-mm-ss.sss 的形式
持续时间			INTERVAL	TIMESTAMP??	XML 持续时间支持来源于 ISO 8601 的格式 PnYnMnDTnHnMr
循环持续时间			INTERVAL		
二进制	m（m<4K）	或者 n, n<4K	RAW（m）或 RAW（n）	BLOB	
二进制	m（m>4K）	或者 n, n>4K	BLOB	RAW	最大长度为 4K
uri			UriType (VARCHAR2)	NVARCHAR2 CLOB NCHAR CHAR、 NCLOB	长度或 uri 必须小于 4K，对 UTF-16 来说则小于 2K。

[0172] 缺省的 XML 数据类型映射到 SQL（针对简单类型）(CONTD)

[0173]

XML 简单类型	缺省的 Oracle 类型	兼容类型	注释
语言（串）	VARCHAR2（4000）	NVARCHAR2 CLOB、CHAR NCLOB、NCHAR	对 UTF-16 来说， 值为 2000 （针对所有类型）
NMTOKEN（串）	VARCHAR2（4000）	相同	""
NMTOKENS（串）	VARCHAR2（4000）	相同	""
Name（串）	VARCHAR2（4000）	相同	普通的 XML 名称
NCName（串）	VARCHAR2（4000）	相同	表示一个非移植名称
ID	VARCHAR2（4000）	相同	在整个文档中是唯一的
IDREF	VARCHAR2（4000）	相同	必须匹配文档中的一个 ID
多个 IDREF	VARCHAR2（4000）	相同	
ENTITY	VARCHAR2（4000）	相同	
多个 ENTITY	VARCHAR2（4000）	相同	
NOTATION	VARCHAR2（4000）	相同	
QName	XDB.XDB$QNAME		表示一个合格的 XML 名称。保存的是一个具有两个属性的对象类型——不合格名称串和变成全局命名空间数组的命名空间的索引编号。
整数	INTEGER	INT, NUMBER	
非负整数	INTEGER	INT, NUMBER	
正整数	INTEGER	INT, NUMBER	
非正整数	INTEGER	INT, NUMBER	
负整数	INTEGER	INT, NUMBER	
日期	DATE	TIMESTAMP	
时间	TIMESTAMP	DATE	

[0174] 映射复杂类型

[0175] 根据一个实施例，complexType 映射为对象类型。在 complexType 内部声明的 XML 属性则映射成对象属性－定义 XML 属性的 simpleType 确定的是相应属性的 SQL 数据类型。在 complexType 内部声明的 XML 元素同样映射为对象属性。所述对象属性的数据类型是由定义 XML 元素的 simpleType（简单类型）或 complexType（复杂类型）来确定的。

[0176] 如果声明了 maxOccurs 属性值大于 1 的 XML 元素，则将其映射成 SQL 中的一个集合属性。所述集合可以是 VARRAY（缺省值）或者嵌套表格（如果将 maintainOrder 属性设定为 FALSE）。此外，VARRAY 默认保存在表格 (OCTs)[OCT-FS] 而不是 LOBs 中－用户可以通过将 storeAsLob 属性设定为 TRUE 来选择 LOB 存储。

[0177] 通常，SQL 属性名称是使用以下算法从 XML 元素或属性名称中产生的：

[0178] 1. 使用 XML 元素／属性名称（截取成 30 个字符）

[0179] 2. 如果发现一个非法 SQL 字符，则将其映射成下划线（' _' ）

[0180] 3. 如果这个名称不是唯一的，则追加一个序列号（注意：其中可能需要在追加数字之前截取名称）然而，用户也可以通过提供用于模式内部的 SQLName 属性的值来显性规定 SQL 属性名称。

[0181] DOM 保真度

[0182] XML 模式内部声明的所有元素和属性都会得到映射，以便分离相应 SQL 对象类型内部的属性。然而，在 XML 实例文档中存在某些并非由这种元素／属性直接表示的信息。其实例包括：

[0183] ●注释

[0184] ●命名空间声明

[0185] ●前缀信息

[0186] 出于 DOM 遍历的目的，为了确保返回的 XML 文档与原始文档一致（这称为 DOM 保真度），在这里将一个名为 SYS_XDBPD$ 的二进制属性添加于生成的所有 SQL 对象类型。这个属性保存的是所有那些不能保存在任何其他属性中的信息，由此确保了数据库系统保存的 XML 文档的 DOM 保真度。注意：为了清楚起见，在许多实例中省略了 SYS_XDBPD$ 属性。然而，所述属性可以存在于模式注册处理生成的所有 SQL 对象结构中。

[0187] SQL 离线存储

[0188] 根据一个实施例，作为默认情况，子元素映射成一个嵌入式对象属性。然而也有可能存在这样一种情况，其中离线存储提供了更好的性能。在这种情况下，SQLInline 属性可以设定为“FALSE”并且 XML 模式映射器 106 产生一个具有嵌入式 REF 属性的对象类型。REF 则指向 XMLType 的另一个实例，所述实例与离线保存的 XML 分段相对应。此外还创建了（XMLType 的）缺省表格，以便保存离线分段。

[0189] Example

[0190] <complexType name ＝" Employee" >--OBJ_T2

[0191] <sequence>

[0192] <element name ＝" Name" type ＝" string" maxLength ＝" 1000" />

[0193] <element name ＝" Age" type ＝" decimal" />

[0194] <element name ＝" Addr" SQLInline ＝" false" >

[0195] <complexType>--OBJ_T1

[0196] <sequence>

[0197] <element name ＝" Street" type ＝" string" maxLength ＝" 100" />

[0198] <element name ＝" City" type ＝" string" maxLength ＝" 100" />

[0199] </sequence>

[0200] </complexType>

[0201] </element>

[0202] </sequence>

[0203] </complexType>

[0204] create type OBJ_T1 as object

[0205] (

[0206] Street varchar2(100),

[0207] City varchar2(100)

[0208]);

[0209] create type OBJ_T2 as object

[0210] (

[0211] Name varchar2(100),

[0212] Age number,

[0213] Addr REF XMLType

[0214]);

[0215] 将 XML 分段映射到 LOBS

[0216] 用户可以将用于复杂元素的 SQLType 规定成 LOB(CLOB/BLOB),在这样情况下,整个 XML 分段都会保存在 LOB 属性中。这在很少查询 XML 文档某些部分而这些部分主要作为单条信息恢复和保存的情况下是很有用的。通过将分段保存为 LOB,可以减少分析 / 分解 / 重组的开销。

[0217] Example

[0218] <complexType name =" Employee" >--OBJ_T

[0219] <sequence>

[0220] <element name =" Name" type =" string" maxLength =" 1000" />

[0221] <element name =" Age" type =" decimal" />

[0222] <element name =" Addr" SQLType =" CLOB" >

[0223] <complexType>

[0224] <sequence>

[0225] <element name =" Street" type =" string" maxLength =" 100" />

[0226] <element name =" City" type =" string" maxLength =" 100" />

[0227] </sequence>

[0228] </complexType>

[0229] </element>

[0230] </sequence>

[0231] </complexType>

[0232] create type OBJ_T as object

[0233] (

[0234] Name varchar2(100),

[0235] Age number,

[0236] Addr CLOB

[0237]);

[0238] 映射简单内容

[0239] 基于简单内容声明的 complexType 映射成一个对象类型,所述对象类型具有那些与 XML 属性相对应的属性以及一个与实体值相对应的附加 SYS_XDBBODY$ 属性。主体属性的数据类型则是基于定义主体类型的 simpleType。

[0240] Example

[0241] <complexType>

[0242] <simpleContent>

[0243] <restriction base =" string" maxLength =" 1000" >

[0244] <attribute name =" a1" type =" string" maxLength =" 100" />

[0245] </restriction>

[0246] </simpleContent>

[0247] </complexType

[0248] create type OBJ_T as object

[0249] (

[0250] a1 varchar2(100),

[0251] SYS_XDBBODY$ varchar2(1000)

[0252]);

[0253] 映射 ANY/ANYATTRIBUTE

[0254] any 元素声明以及 anyAttribute 属性声明映射成对象类型中的 LOBs。LOBs 保存的是与任何声明匹配的 XML 分段的文本。命名空间属性可用于限制归属于规定命名空间的内容。任何元素声明内部的 processContents 属性表示的是与任何声明相匹配的内容所需要的验证等级。

[0255] Example

[0256] <complexType name =" Employee" >

[0257] <sequence>

[0258] <element name =" Name" type =" string" maxLength =" 1000" />

[0259] <element name =" Age" type =" decimal" />

[0260] <any namespace =" http://www/w3.org/2001/xhtml" processContents =

[0261] " skip" />

[0262] </sequence>

[0263] </complexType>

[0264] create type OBJ_T as object

[0265] (

[0266] Name varchar2(100),

[0267] Age number,

[0268] SYS_XDBANY$ blob

[0269]);

[0270] 将串映射成 SQLVARCHAR2VS CLOB

[0271] 如果 XML 模式将数据类型规定为“串”并且规定了一个小于 4000 的 maxLength(最大长度)值,则将其映射成一个规定长度的 varchar2 属性。然而,如果在 XML 模式中没有规定 maxLength 的值,则只能将其映射成一个 LOB。在大多数串值实际很小并且其中很少一部分大到足以需要 LOB 的情况下,这是次最佳的。理想的 SQL 数据类型是 varchar2(*),对很小的串来说,它是像 varchars 那样执行的,但是也可以容纳更大的串。此外,这类列应该支持所有 varchar 函数,例如索引、SQL 函数等等。此外也可以进行一个较小的范例,因为需要一个 raw(*) 数据类型,以便在没有丧失性能和/或功能的情况下保持无限制的二进制值。

[0272] 根据一个替换实施例,所有无限制的串都映射成 CLOBs 并且所有无限制的二进制元素/属性都映射成 BLOBs。

[0273] 将串映射成 SQL VARCHAR2RCHAR2

[0274] 作为默认情况，XML 串数据类型映射成了 SQL varchar2。然而，用户也可以用两种方式来替换这种行为：

[0275] 1. 用户可以将 SQLType 规定成用于某个串元素或属性的 NVARCHAR2。这样就确保将 NVARCHAR2 选作某个元素 / 属性的 SQL 类型。

[0276] 2. 用户可以在模式声明顶端将 mapStringToNCHAR 属性设定为“true”。这样一来，除非在元素级别得到明确替换，否则将会确保将所有 XML 串映射成 NVARCHAR2（或 NCLOB）数据类型。

[0277] 创建基于模式的 XML 表格

[0278] 假定已经注册了由 http://www.oracle.com/PO.xsd 标识的 XML 模式。为了保存符合这个模式的 PurchaseOrder 元素的实例，可以创建一个 XMLType 表格 -- 在对象关系格式中如下所示：

[0279] create table MyPOs of xmltype

[0280] element" http://www.oracle.com/PO.xsd#PurchaseOder" ;

[0281] 隐藏列是对应于 PurchaseOrder 元素映射的对象类型而被创建的。此外，为了保存最高级别的实例数据，例如命名空间声明等等，在这里还创建了 XMLExtra 对象列。注意：XMLDATA 是允许直接访问基础对象列的 XMLType 的一个伪属性。

[0282] 规定存储从句

[0283] 基础列可以在存储从句中通过以下注释来加以引用：

[0284] 1. 对象符号 :XMLDATA.<attr1>.<attr2>…

[0285] 2.XML 符号 :ExtractValue(xmltypecol, ' /attr1/attr2')

[0286] create table MyPOs of xmltype

[0287] element " http://www.oracle.com/PO.xsd#PurchaseOrder"

[0288] lob(xmldata.lobattr)store as(tablespace...) ;

[0289] create table MyPOs of xmltype

[0290] element" http://www.oracle.com/PO.xsd#PurchaseOrder"

[0291] lob(ExtractValue(MyPOs, ' /lobattr'))store as(tablespace...) ;

[0292] 创建索引

[0293] 如上所示，处于 XMLType 列下方的那些列可以使用 CREATEINDEX 声明中的对象标志或 XML 标志来加以引用。

[0294] create index ponum_idx on MyPOs(xmldata.ponum) ;

[0295] create index ponum_idx on MyPOs p(ExtractValue(p, ' /ponum') ;

[0296] 约束条件

[0297] 通过使用对象或 XML 符号，可以为基础列规定约束条件。

[0298] create table MyPOs of xmltype

[0299] element" http://www.oracle.com/PO.xsd#PurchaseOrder"

[0300] (unique(xmldata.ponum)) ;

[0301] create table MyPOs p of xmltype

[0302] element

[0303] " http://www.oracle.com/PO.xsd#PurchaseOrder" (unique(ExtractVal

[0304] ue(p, ' /ponum'));

[0305] DMLS

[0306] 新的实例是如下插入 XMLType 表格的：

[0307] insert into MyPOs values

[0308] (xmltype.createxml(' <PurchaseOrder>.....</PurchaseOrder>'));

[0309] XMLType 表格可以使用基于 XPath 的 SQL 运算符来进行查询。

[0310] select value(p)from MyPOs where extractValue(value(p),

[0311] ' /Company) =' Oracle' ;

[0312] 该查询改写机制改写了包含 existsNode 和提取运算符在内的查询，以便直接访问基础属性列 -- 由此避免 XML 结构之后跟随后续的 XPath 估计。举例来说，上述查询改写为：

[0313] select value(p)from MyPOs where p.xmldata.company =' Oracle' ;

[0314] 查询改写

[0315] 操作那些基于模式的 XMLType 列的基于 XPath 的运算符 (Extract，ExistNode，ExtractValue) 将被重写，以便依靠基础 SQL 列来加以执行。这样则允许充分利用 XML 对象关系存储来进行进一步的 SQL 优化。以下类型的 XPath 表达式可以转换为基础 SQL 查询：

[0316] 1. 简单 XPath 表达式 - 包括只在对象类型属性上的遍历，其中所述属性是简单标量或者对象类型自身。而唯一的坐标轴则是子女和属性坐标轴。

[0317] 2. 集合遍历表达式 - 包括集合表达式的遍历。只有子和属性坐标轴才是得到支持的。

[0318] 3. 表达式包括 * 轴 - 如果合成节点的数据类型全都是可强制的 (coercible)，则转换这些包含了通配符坐标轴的表达式。(例如 CUST/*/CUSTNAME 必须指向全部相同或可压缩数据类型的 CUSTNAME)。

[0319] 4. 包括子轴 (//) 的表达式 -- 如果最终得到的节点的数据类型是相同或可压缩的，则转换这些表达式。

[0320] 5. 所有这些表达式必须结合类型缓存器来进行操作，其中包含了与 REFs 到 XMLTypes 等等相似的“隐藏”遍历。(举例来说，xdb$schema_t 保存的是 REF 到 xdb$element_t 的一个 varray，这在 XPath 表达式或是最终得到的 XML 文档并不是直接显现的)。

[0321] 在 ExistsNode、ExtractValue 和 Extract 使用情况中，这些 XPath 表达式的转换是得到支持的。

[0322] XPath 的查询改写的实例

[0323] 初始查询

[0324] select * from MyPOs p

[0325] where ExistsNode(p, ? /PO[PNAME =? PO1 ?]PONO ?) = 1

[0326] 在 ExisitsNode 的重写之后

[0327] select * from MyPOs p

[0328] where(CASE WHEN(p.xmldata.pono IS NOT NULL)

[0329] AND(p.xmldata.PNAME =? PO1 ?))THEN1 ELS∃0) = 1

[0330] 初始语句

[0331] select ExtractValue(p, ? /[PNAME =? PO1']/PONO ?)from MyPOs p

[0332] 在Extract(提取)改写之后

[0333] select(select p.xmldata.pono from dual where

[0334] p.xmldata.pname =? PO1 ?)

[0335] from MyPOs;

[0336] 函数改写规则

[0337] EXTRACT、EXTRACTVALUE 和 EXISTSNODE 可以出现在以下位置

[0338] ●在 SQL 查询的 select 列表、where 从句谓语、group by 和 orderby 表达式中。

[0339] ●在创建 INDEX 声明的索引从句中。

[0340] create index foo_index on foo_tab(extractvalue(xml_col, ' /PO/PONO'));

[0341] 在所有这些情况中,EXISTSNODE 和 EXTRACT 运算符由其定义基础表达式替换。XPath 表达式必须满足与之相关的先前部分中列举的条件,以便得到改写。

[0342] 在索引情况中,如果替换整个运算符树将会导致产生单个的列,则将所述索引转换成一个 BTree 或是列上的一个域索引,而不是一个函数索引。

[0343] 为 OBJECT/SCALAR 属性遍历进行改写

[0344] 简单的 XPath 遍历将被改写到对象类型访问程序中。谓语则是通过置于 where 从句中而得到处理的。一个对象类型上的任何 XPath 子访问都转换成一个针对基础对象类型的对象属性访问。举例来说,A/B 映射到 a.b,其中 A 映射到对象类型 a 并且 XPath 节点映射成以“b”命名的“a”的属性。

[0345] 在任何等级的 XPath 表达式中,也就是说,无论 XPath 遍历出现在谓语内部还是位置路径变量,这个改写都是一致的。

[0346] 举例来说,

[0347] PO/CUSTOMER/CUSTOMERNAME 变成“po”.“cust”.“custname”(假设 PO 映射成“po”等等)

[0348] 谓语 (predicate) 是通过改写基础对象表达式中的谓语表达式而得到处理的。

[0349] 在简单范例中,对于 EXISTSNODE 来说,主位置路径遍历变成一个 IS NOT NULL 谓语,而对 EXTRACT 范例来说,所述遍历变成了正在提取的实际节点。

[0350] EXISTSNODE(po_col, ' PO/CUSTOMER/CUSTOMERNAME') 变成

[0351] CASE(WHEN(" po" ." cust" ." custname" IS NOT NULL)then 1else 0)

[0352] 谓语也是以相似方式处理的。举例来说,在如下给出的表达式中,

[0353] EXISTSNODE(po_col, ' PO/CUSTOMER[CUSTOMERNO = 20]/CUSTOMERNAME')

[0354] 谓语 D = 20 是如用户规定的那样得到处理的 (A/B/D = 20)

[0355] 因此整个表达式变成

[0356] CASE(WHEN(" PO" ." CUST" ." CUSTNAME" IS NOT NULL

[0357] AND(" PO" ." CUST" ." CUSTNO" = 20))THEN 1 ELSE 0)

[0358] 集合遍历

[0359] XPath 表达式还可以跨越集合结构并且查询仍是通过对集合表格使用子查询而得到改写的。举例来说，

[0360] EXISTSNODE(po_col, ' /PO/lineitems[lineitemno = 20]') 是在一个购货订单中检查 lineitem 是否存在,其中 lineitem 数目是 20。这样则变成了

[0361] case(when(exists(select*from TABLE(" po " . " lineitems ") wherelineitemno = 20))then 1 else 0)

[0362] 缺省表格

[0363] 作为模式注册的一部分,也可以创建缺省表格。在借助于不具有任何 FTP、HTTP 之类的表格规范的 API 插入符合这种模式的 XML 实例文档的情况下,缺省表格是有用的。在这种情况下,XML 实例插入缺省表格中。

[0364] 如果用户为 defaultTable 属性给出一个值,则使用所述名称来创建 XMLType 表格。否则使用内部产生的名称来创建所述表格。此外,由 tableStorage 属性规定的任何文本都附加于生成的 CREATETABLE 语句。

[0365] 规定内部存储器的数据类型

[0366] XML 数据保存在 RDBMS 存储器内部的一个 C 结构中。通常,由于未必访问文档的很多部分,因此 XML 数据的内存表示试图避免在加载时间进行数据类型转换并且旨在访问时转换数据。作为模式注册的一部分,内存数据类型是基于 XML 数据类型来选择的 -- 这个信息是使用 memDatatype 属性保存在模式文档内部的。然而也存在某些情况,其中应用可能希望替换缺省存储器类型,以便有利于不同的内存表示。

[0367] 举例来说,串的缺省存储器表示是“char”,它在数据库会话字符集合中保存串数据。然而如果这个数据只由在定宽 UCS-2Unicode 中需要所述数据的 Java 应用所消耗,那么性能更好的有可能是将 memDatatype 数据类型复位成“JavaString”。这样则确保了数据库服务器 104 直接将数据以 Unicode 格式保存在 Java 存储器中 -- 从而避免任何格式转换或拷贝。

[0368]

XML Datatype	许可的存储器数据类型	描述	缺省值
串	Char	在当前为这个会话激活的字符集中改变宽度字符	是
	JavaString	从 JServer 存储器中分配的定宽 UCS-2 Unicode	否
整数	整数	作为缺省值的是有符号的 8 字节本地整数; 如果 XML 模式规定最大和最小值, 则可以使用一个更小或是无符号数据类型	是
	数字	Oracle 数字格式	否
浮点	浮点	本地最大精度浮点;如果在较小类型的范围内部 规定了最大和最小值,则可以使用较小的值。	是
	数字	Oracle 数字格式	否

[0369] 映射信息的产生

[0370] 一旦为特定的 XML 模式确定了适当的数据库表示,则产生映射信息来对适当数据库表示的元素与特定 XML 模式中标识的元素之间的相关性加以识别。举例来说,如果关于类型“person”的 XML 模式的适当数据库表示包括一个用于保存 person XML 文档中包含的数据项的表格,那么映射信息指示的是 person XML 文档与表格 PERSON 之间的相关性。

[0371] 除了 XML 模式与数据库模式对象(例如表格)之间的常规相关性之外,所述映射信息还可以反映更精密的粒度层次相关性。举例来说,映射信息可以表示应该使用 PERSON 表格的哪一列来来保存 person XML 文档内部的特定数据项。

[0372] 根据一个实施例,涉及 SQL 映射的信息自身保存在 XML 模式文档内部。在注册处理过程中,XML 模式映射器 106 产生 SQL 类型(如上所示)。此外,它还将注释添加到 XML 模式文档中,以便保存映射信息。在这里,这些注解采用了新属性的形式。实例 :以下模式显示了借助 SQLType 和 SQLName 属性捕获的 SQL 映射信息。

[0373] <schema xmlns =" http://www.w3.org/2001/XMLSchema"

[0374] targetNamespace =" http://www.oracle.com/PO.xsd" >

[0375] <complexType name =" PurchaseOrder" >

[0376] <attribute name = " PurchaseDate " type = " date " SQLName =" PURCHASEDATE"

[0377] SQLType =" DATE" />

[0378] <sequence>

[0379] <element name =" PONum" type =" decimal" SQLName =" PONUM"

[0380] SQLType =" NUMBER" />

[0381] <element name =" Company" type =" string" maxLength =" 100"

[0382] SQLName =" COMPANY" SQLType =" VARCHAR2" />

[0383] <element name = " Item " maxOccurs = " 1000 " SQLName

=" ITEM" SQLType =" ITEM_T"

[0384] SQLCollType =" ITEM_VARRAY_T" >

[0385] <complexType>

[0386] <sequence>

[0387] <element name =" Part" type =" string" maxLength =" 1000"

[0388] SQLName =" PART" SQLType =" VARCHAR2" />

[0389] <element name =" Price" type =" float" SQLName =" PRICE"

[0390] SQLType =" NUMBER" />

[0391] </sequence>

[0392] </complexType>

[0393] </element>

[0394] </sequence>

[0395] </complexType>

[0396] <element name =" PO" type =" PurchaseOrder" SQLType =" PURCHASEORDER_T" />

[0397] </schema>

[0398] 处于输入模式文档中的用户规定的名称

[0399] 在注册模式之前,用户可以填充 SQLName 和 SQLType 属性来规定 SQL 对象类型及其属性的名称。如果 SQLName 和 SQLType 值由用户规定,那么 XML 模式映射器 106 使用这些名称来创建 SQL 对象类型。如果这些属性不是用户规定的,则使用一个内部名称生成算法来产生所述名称。关于名称生成算法的细节可以参见附录。

[0400] 以下表格列举了模式内部用以捕获 SQL 映射信息的所有注释。应该注意的是,用户不必规定这些属性中任何一个的值。XML 模式映射器会在模式注册处理中填充适当的值。然而,在这里建议用户至少规定顶层 SQL 类型的名称 -- 以便稍后能够对其加以引用。所有注释都使用了可以在属性和元素声明中规定的属性的模式。这些属性归属于 XDB 域名空间 :http://xmlns.oracle.com/xdb/XDBSchema.xsd

[0401] 表 1 :可以在元素和属性声明内部规定的 XDB 属性

[0402]

属性	值	缺省值	描述
SQLName	任何 SQL 标识符	元素名称	这个属性规定的是映射到这个 XML 元素的 SQL 对象内部的属性名称。
SQLType	任何 SQL 类型名称	从元素名称中产生的名称	这个性质规定的是与这个 XML 元素或属性相对应的 SQL 类型的名称。这其中可以引用一个标量或依赖于 XML 模式声明的对象类型。
SQLCollType	任何 SQL 集合类型名称	从元素名称中产生的名称	这个属性规定的是与 maxOccurs>1 的 XML 元素相对应的 SQL 集合类型的名称。
SQLSchema	任何 SQL 用户名称	注册 XML 模式	拥有 SQLType 规定的类型的数据库用户的名称。
SQLCollSchema	任何 SQL 用户名称	用户注册 XML 模式	拥有 SQLCollType 规定的类型的数据库用户的名称。
maintainOrder	true | false	True	如果是 “TRUE”，则将集合映射成一个 VARRAY。否则将集合映射成一个云套表格。
storeVarrayAsLob	true | false	True	如果为 “TRUE”，则将 VARRAY 保存在一个 LOB 中。如果为 “FALSE”，则将 varray 作为表格（OCT）加以保存。
SQLInline	true | false	true	如果为 “TRUE”，则将这个元素作为一个嵌入属性联机保存（如果 maxOccurs>1，则将其作为集合）。如果为 “FALSE”，则保存一个 REF（如果 maxOccurs>1，则保存一个 REF 集合）。在某些情况下，SQL 不支持联机，则将这个属性强制为 “FALSE”。
maintainDOM	true | false	true	如果为 “TRUE”，则保存这个元素的实例，以使它们在输出上保留 DOM 保真度。这意味着除了元素的排序之外，所有注释、处理指令、命名空间声明等等都得到了保持。如果为 “FALSE”，则不必保证输出具有与输出相同的 DOM 行为。
tableStorage	任何接下来有效的存储从句	NULL	这个属性规定的是附加于缺省表格创建语句的存储从句。它主要对映射到表格即顶级元素声明和离线元素声明的元素有意义。
defaultTable	任何表格名称	基于元素名称	这个属性规定的是这个模式的 XML 实例应该存入的表格的名称。这在从没有诸如 FTP、HTTP 之类的规定表格名称的 API 插入 XML 的时候最有用。
defaultACL	任何指向 ACL 文档的 URL	NULL	这个属性规定的是应该作为缺省情况应用于这个元素[Folder-FS]的所有实例的 ACL 的 URL。
isFolder	true | false	false	如果为 TRUE，则可以在 XDB[Folder-FS] 内部将这个元素的实例用作一个文件夹（或容器）。
mapStringToNCHAR	true | false	false	如果为 “TRUE”，则将所有 XML 串映射成 NVARCHAR2（或 NCLOB）数据类型，除非在元素级别得到了显性覆盖。如果为 “FALSE”，则将所有 XML 串/元素/属性映射成 varchar2 列。
memDatatype	内存数据类型	内部	这个属性可用于覆盖（简单）元素和属性的缺省内存映射。以下参见用于给定 XML 数据类型的许可存储器数据类型的表格。

[0403] 混合存储模型

[0404] 依照一个实施例，在这里实施了 XML 模式映射器 106 来支持混合存储模型，其中

XML 内部定义的某些元素的结构保存在适当数据库表示之中，而其他元素的结构则并非如此。例如，可以将 XML 文档类型中最频繁查询 / 更新的部分映射成对象类型属性，同时 XML 文档的剩余部分则一起保存在 CLOB 中。根据一个实施例，针对这种结构的规定部分是通过预先以适当映射命令预注释 XML 模式来加以保持或是未曾保持。

[0405] XML 模式注册的事务特性

[0406] 根据一个实施例，当在模式注册操作中遇到差错时，XML 模式注册是以一种允许执行消除局部影响的补偿活动的方式来使用数据库服务器 104 的事务支持而被执行的。

[0407] 处理 XML 模式中的循环定义

[0408] 对 XML 模式来说，有可能包含了循环。根据一个实施例，XML 模式映射器 106 被配置成检测这种循环，并且在映射到 SQL 对象类型的同时使用 REF 将其中断。在附录 I 中提供了如何将 REF 用于中断循环的详细描述。

[0409] 基于映射信息来保存 XML 文档

[0410] 在向数据库服务器 104 注册了某个文档类型的 XML 模式之后，符合所述模式的 XML 文档可以由数据库服务器 104 智能管理。根据一个实施例，当协议表明必须将资源存入一个受数据库服务器 104 管理的数据库时，数据库服务器 104 将会在文件名扩展部分对诸如 xml、.xsl、.xsd 等等进行检查。如果文档是 XML，则执行一个预分析步骤，其中将会读取足够的资源来确定文档中命名空间与根元素的 XMLschemaLocation。而这个位置则被用于寻找具有 schemaLocation URL 的已注册模式。如果已注册模式确定具有当前文档的根元素的定义，则使用为所述元素规定的缺省表格来保存资源内容。

[0411] 根据一个实施例，在将 XML 文档保存在支持这里描述的 XML 模式注册技术的数据库服务器的时候，数据库服务器可以验证文档，以便核实它们是否符合相应的 XML 模式。所述验证可以包括关于 XML 文档所用结构和数据类型的验证。

[0412] 多个其他优点是通过使用这里描述的技术来实现的。举例来说，模式注册处理允许数据库服务器执行完整性约束条件以及有关 XML 文档与用以保存所述文档的表格的其他形式的约束条件。此外，数据库服务器能够根据 XML 数据来创建索引和分区 XML 表格。

[0413] 由于 XML 文档结构是在如何将来自 XML 文档的数据保存在数据库内部的过程中得到反映的，因此，通常使用标签信息来反映不必将所述结构与数据一起保存。由于 XML 标签通常会形成 XML 文档尺寸的很大一部分，因此避免保存某些或全部 XML 标签的能力有可能显著降低存储开销。

[0414] 其他性能优点也同样是可能的。举例来说，通过改写 XPath 查询来直接访问基础列，可以改善查询性能。此外，通过改写更新来直接更新基础列，可以改善更新性能。因此，对来自所保存的文档的 XML 数据的一部分进行更新不会始终需要改写所保存文档的全部 XML 数据。

[0415] 硬件综述

[0416] 图 2 是描述可以执行本发明一个实施例的计算机系统 200 的框图。计算机系统 200 包括一条总线 202 或是用于传递信息的其他通信结构，并且包括一个与总线 202 耦合并用于处理信息的处理器 204。计算机系统 200 还包含一个主存储器 206，例如随机访问存储器 (RAM) 或是其它动态存储器，这个存储器与总线 202 耦合，用于保存信息以及处理器 204 所要执行的指令。在运行处理器 204 所执行指令的过程中，主存储器 206 还可用于保存临

时变量或是其它中间信息。计算机系统 200 还包括一个只读存储器 (ROM)208 或其它静态存储设备，它与总线 202 耦合，用于保存静态信息和涉及处理器 204 的指令。此外还提供了诸如磁盘或光盘这种存储设备 210，所述设备与总线 202 耦合，以便保存信息和指令。

[0417] 计算机系统 200 可以经由总线 202 而与阴极射线管 (CRT) 之类的显示器 212 相耦合，以便将信息显示给计算机用户。包含字母数字及其他按键的输入设备 214 与总线 202 相连，以便将信息和命令选择传递到处理器 204。另一种用户输入设备是光标控制器 216，例如鼠标、轨迹球或游标方向键，用于将方向信息和命令选择传递给处理器 204 并控制显示器 212 上的游标移动。这种输入设备通常在第一轴 (例如 x) 和第二轴 (例如 y) 这两个轴上具有两个自由度，由此设备能够确定一个平面上的位置。

[0418] 本发明涉及使用计算机系统 200 来执行这里所描述的技术方法。根据本发明的一个实施例，处理器 204 执行主存储器 206 中包含的一个或多个指令的一个或多个序列，计算机系统 200 对此做出响应，由此执行这些技术方法。这些指令可以从诸如存储设备 210 等等的另一种计算机可读介质读入主存储器 206。通过执行主存储器中包含的指令序列，处理器 204 执行这里描述的处理步骤。在替换实施例中，硬布线电路可用于取代软件指令或是与之组合，由此实现本发明。因此，本发明的实施例不限于硬件电路和软件的任何一种特定组合。

[0419] 这里使用的术语“计算机可读介质”是指任何一种参与向处理器 204 提供指令以供执行的介质。这种介质可以采取很多形式，其中包括但不局限于 :非易失介质、易失介质和传输介质。举例来说，非易失介质包括光盘或磁盘，例如存储设备 210。易失介质包括动态存储器，例如主存储器 206。传输介质包括同轴电缆、铜线和光纤，其中包括了构成总线 202 的线路。传输介质还可以采取声波或光波的形式，例如无线电波和红外数据通信中产生的信号。

[0420] 举例来说，计算机可读介质的通用形式包括 :软盘、软磁盘、硬盘、磁带或任何其它磁介质、光盘或任何其它光学介质、穿孔卡、纸带纸条或具有孔洞图案的任何其它物理介质、RAM、PROM 和 EPROM、FLASH-EPROM、其它任何存储芯片或盒式磁盘机、如下所述的载波或是计算机可以读取的其它任何介质。

[0421] 在向处理器 204 传递一个或多个指令的一个或多个序列来加以执行的过程中，可以包含不同形式的计算机可读介质。举例来说，最初将指令传送到远程计算机磁盘上。远程计算机可以将指令加载到它的动态存储器中，并且使用调制解调器而经由电话线来发送指令。计算机系统 200 的本地调制解调器可以在电话线上接收数据并且使用红外发射机来将数据转换成红外信号。红外检测器可以接收红外信号中传送的数据，而适当的电路则可将数据安插到总线 202 上。总线 202 将数据传送到主存储器 206，处理器 204 从主存储器 206 中检索并执行指令。在由处理器 204 执行之前或之后，由主存储器 206 接收的指令可以随意保存在存储设备 210 中。

[0422] 计算机系统 200 还包括一个与总线 202 相连的通信接口 218。通信接口 218 提供了一个与网络链路 220 耦合的双向数据通信，其中网络链路 220 与本地网络 222 相连。举例来说，通信接口 218 可以是一个向相应类型的电话线路提供数据通信连接的综合业务数字网 (ISDN) 的网卡或是调制解调器。作为另一个实例，通信接口 218 可以是一个 LAN 网卡，它向兼容的 LAN 提供数据通信连接。此外还可以实施无线链路。在任何一种这类实施

中，通信接口 218 都会收发电、电磁或光信号，这些信号传送的是那些代表不同类型信息的数字数据流。

[0423] 网络链路 220 通常经由一个或多个网络来向其它数据设备提供数据通信。举例来说，网络链路 220 可以经由本地网络 222 而将一个连接提供给主机 224 或是互联网服务供应商 (ISP)226 运作的数据设备。ISP 226 进而又通过现在通常称为“互联网”的全球分组数据通信网络 228 来提供数据通信业务。本地网络 222 和互联网 228 都使用了传送数字数据流的电、电磁或光信号。经由不同网络的信号以及网络链路 220 上经由通信接口 218 的信号传送的是那些往返于计算机系统 200 的数字数据，而这些信号即为传送信息的载波的示范性形式。

[0424] 计算机系统 200 可以经由一个或多个网络、网络链路 220 以及通信接口 218 来发送消息和接收数据，其中包括了程序代码。在互联网实例中，服务器 230 可以经由互联网 228、ISP 226、本地网络 222 以及通信接口 218 来发送一个用于应用程序的被请求码。

[0425] 接收到的代码可以在接收时由处理器 204 执行和 / 或存入存储设备 210 或其它非易失存储器中，以供随后执行。这样，计算机系统 200 可以通过载波形式来获取应用码。

[0426] 在以上的说明书中，本发明的实施例是参考众多可能随实施方式变化的具体细节来描述的。因此，关于本发明和意图确定为本发明的应用的唯一排它指示即为本说明书中公开的权利要求集合，其中采用的是包含后续任何校正的权利要求发布的形式。关于这里描述的权利要求所包含内容的任何定义应该控制权利要求中使用的术语的意义。因此，无论如何，那些没有在权利要求中明确陈述的限制、元素、性质、特征、优点或属性并没有限制权利要求的范围。因此，所述说明书和附图应该视为是示范性而不是限制意义的。

[0427] 附录 I

[0428] ●介绍 XML 模式

[0429] ●介绍 XML 模式和 Oracle XML DB

[0430] ●使用 Oracle XML DB 和 XML 模式

[0431] ●介绍 DBMS_XMLSCHEMA

[0432] ●在使用 Oracle XML DB 之前注册您的 XML 模式

[0433] ●使用 DBMS_XMLSCHEMA 来删除您的 XML 模式

[0434] ●已注册 XML 模式的使用指南

[0435] ●在 XML 模式注册过程中产生 Java Bean

[0436] ●使用 DBMS_XMLSCHEMA.generateSchema() 来从对象关系类型中产生 XML 模式

[0437] ●涉及 XML 模式的 XMLType 的方法

[0438] ●管理和保存 XML 模式

[0439] ● DOM 保真度

[0440] ● Oracle XML DB 基于 XML 模式来创建 XMLType 表和列

[0441] ●在注册 XML 模式之前使用 SQLName 和 SQLType 属性来规定 SQL 对象类型名称

[0442] ●使用 DBMS_XMLSCHEMA 来映射类型

[0443] XML 模式 :将 SimpleTypes 映射成 SQL

[0444] ● XML 模式 :将 ComplexTypes 映射成 SQL

[0445] ● Oracle XML DB complexType 的扩展和限制

[0446] ●用于创建基于 XML 模式的 XML 表格的进一步准则

[0447] ●使用基于 XML 模式的对象关系存储的查询改写

[0448] ●在 XML 模式注册中创建缺省表格

[0449] ●表格中的有序集合 (OCTs)

[0450] ● XML 模式之间的循环引用

[0451] 介绍 XML 模式

[0452] XML 模式建议由万维网协会 (W3C) 创建,用于以 XML 方式来描述 XML 文档的内容和结构。它包含了了文档类型定义 (DTD) 的全部能力,以便将现有的 DTD 转换成 XML 模式。与 DTD 相比,XML 模式具有附加的能力。

[0453] 介绍 XML 模式和 Oracle XML DB

[0454] XML 模式是一种用 XML 方式便编写的模式定义语言。它可以用于描述相容实例文档的结构和多种其他语义。举例来说,以下的 XML 模式定义 po.xsd 描述了购买订单 XML 文档的结构和其他属性。

[0455] 实例 5-1 XML 模式定义 po.xsd

[0456] -- 以下是 XML 模式定义的一个实例 po.xsd :

[0457] <schema targetNamespace =" http://www.oracle.com/PO.xsd"

[0458] xmlns:po =" http://www.oracle.com/PO.xsd"

[0459] xmlns =" http://www.w3.org/2001/XMLSchema" >

[0460] <complexType name =" PurchaseOrderType" >

[0461] <sequence>

[0462] <element name =" PONum" type =" decimal" />

[0463] <element name =" Company" >

[0464] <simpleType>

[0465] <restriction base =" string" >

[0466] <maxLength value =" 100" />

[0467] </restriction>

[0468] <simpleType>

[0469] </element>

[0470] <element name =" Item" maxOccurs =" 1000" >

[0471] <complexType>

[0472] <sequence>

[0473] <element name =" Part" >

[0474] <simpleType>

[0475] <restriction base =" string" >

[0476] <maxLength value =" 1000" />

[0477] </restriction>

[0478] </simpleType>

[0479] </element>

[0480] <element name =" Price" type =" float" />

[0481] </sequence>

[0482] </complexType>

[0483] </element>

[0484] </sequence>

[0485] </complexType>

[0486] <element name＝" PurchaseOrder" type＝" po:PurchaseOrderType" />

[0487] </schema>

[0488] 实例 5-2 符合 XML 模式 po.xsd 的 XML 文档 po.xml

[0489] -- 以下是符合 XML 模式 po.xsd 的 XML 文档的一个实例 :

[0490] <PurchaseOrder xmlns ＝" http://www.oracle.com/PO.xsd"

[0491] xmlns:xsi ＝" http://www.w3.org/2001/XMLSchema-instance"

[0492] xsi:schemaLocation ＝" http://www.oracle.com/PO.xsd

[0493] http://www.oracle.com/PO.xsd" >

[0494] <PONum>1001</PONum>

[0495] <Company>Oracle Corp</Company>

[0496] <Item>

[0497] <Part>9i Doc Set</Part>

[0498] <Price>2550</Price>

[0499] </Item>

[0500] </PurchaseOrder>

[0501] 注意 :

[0502] 这里使用的 URL http://www.oracle.com/PO.xsd 只是一个唯一标识数据库内部已注册 XML 模式的名称并且不需要是 XML 模式文档所在的物理 URL。同样,XML 模式的目标命名空间是不同于 XML 模式位置 URL 的另一个 URL,它规定的是一个在内部声明了元素和类型的抽象命名空间。

[0503] XML 模式可选地规定了目标命名空间 URL。如果省略这个属性,则 XML 模式没有目标命名空间。注意 :目标命名空间通常与 XML 模式的 URL 是相同的。

[0504] XML 实例文档必须规定根元素的命名空间 (与 XML 的目标命名空间相同) 以及定义了这个根元素的 XML 模式的位置 (URL)。所述位置是结合属性 xsi:schemaLocation 来规定的。当 XML 模式没有目标命名空间时,使用属性 xsi:noNamespaceSchemaLocation 来规定 XML 模式的 URL。

[0505] 使用 Oracle XML DB 和 XML 模式

[0506] Oracle XML DB 将带有注释的 XML 模式即带有若干定义了 Oracle XML DB 的属性的标准 XML 模式定义用作元数据。这些属性处于不同的命名空间并对实例文档如何映射成数据库加以控制。由于这些属性处于与 XML 模式命名空间不同的命名空间,因此这种带有注释的 XML 模式仍旧是合法的 XML 模式文档。

[0507] 当使用 Oracle XML DB 时,您必须首先注册您的 XML 模式。然后您可以在创建 XMLType 表格、列和视图的同时使用 XML 模式 URL。

[0508] Oracle XML DB 为以下任务提供了 XML 模式支持 :

[0509] ●注册任何符合 W3C 的 XML 模式。

[0510] ●依靠一个已注册 XML 模式来对您的 XML 文档进行验证。

[0511] ●注册局部和全局 XML 模式。

[0512] ●从对象类型中生成 XML 模式。

[0513] ●引用另一个用户拥有的 XML 模式。

[0514] ●当存在具有同一名称的局部 XML 模式时，允许用户显性引用一个全局 XML 模式。

[0515] ●在 XML 模式注册过程中从您的 XML 模式中产生结构化的数据库映射。这其中包括产生 SQL 对象类型、集合类型以及缺省表格，并且使用 XML 模式属性来获取映射信息。

[0516] ●在存在多个合法映射的时候规定某个 SQL 类型映射。基于已注册 XML 模式来创建 XMLType 表格、视图和列。

[0517] ●对基于 XML 模式的 XMLType 表格执行操作 (DML) 和查询

[0518] ●在使用 FTP、HTTP/WebDav 协议及其他语言而将基于模式的 XML 实例插入 Oracle XML DB 的时候，将数据自动插入缺省表格。

[0519] 为什么我们需要 XML 模式？

[0520] XMLType 是一种数据类型，它简化了 XML 在数据库的表格和列中的存储。XML 模式还简化了 XML 列和表格在数据库中的存储，并且它们为您提供了针对 XML 数据的更多存储和访问选择以及节约空间 - 性能的选择。

[0521] 举例来说，您可以使用 XML 模式来声明可以使用哪些元素和属性以及允许在 XML 文档中保存或处理何种嵌套元素以及数据类型。

[0522] XML 模式提供了灵活的 XML-to-SQL 映射建立

[0523] 将 Oracle XML DB 与 XML 模式结合使用为 XML 存储映射提供了一个灵活的建立方式。举例来说：

[0524] ●如果您的数据是高度结构化的（最接近 XML），那么 XML 文档中的每个元素都可以作为一个列而被保存在一个表格中。

[0525] ●如果您的数据是无结构的（所有或大多数非 XML 数据），则可以将数据保存在一个字符大型对象 (CLOB) 中。

[0526] 您选择哪一种存储方法依赖于您的数据如何使用，此外还依赖于查询能力以及您对查询和更新您的数据的需要。换句话说，使用 XML 模式为您提供了保存高度结构化或无结构数据的更多灵活性。

[0527] XML 模式允许 XML 实例验证

[0528] 使用具有 Oracle XML DB 的另一个优点是您可以根据 XML 模式并且相对于最优性能的 Oracle XML 存储需要来执行 XML 实例验证。举例来说，XML 模式可以检查所述输入 XML 文档是否遵守 XML 模式中声明的定义，例如许可的结构、类型、许可项目出现的次数或是许可的项目长度。

[0529] 同样，通过在 Oracle XML DB 中注册 XML 模式，当使用 FTP 或 HTTP 之类的协议插入和保存 XML 实例时，XML 模式信息可以影响到如何有效插入 XML 实例。

[0530] 当必须在没有任何关于 XML 实例的在先信息的情况下处理 XML 实例时，XML 可以在预测最优存储、保真度和访问方面得到应用。

[0531] 介绍 DBMS_XMLSCHEMA

[0532] Oracle XML DB 的 XML 模式功能可以通过一个提供 PL/SQL 的包 DBMS_XMLSCHEMA 来得到，所述包是一个对 Oracle XMLDB 应用所用的 XML 模式定义注册进行处理的服务器端的组件。

[0533] 主 DBMS_XMLSCHEMA 函数中的两个函数是：

[0534] ● registerSchema()。这个函数注册的是一个给出以下信息的 XML 模式：

[0535] XML 模式源，它可以采用多种格式，其中包括串、LOB、XMLType 和 URIType

[0536] ●它的模式 URL 或 XMLSchema 名称

[0537] ● deleteSchema()。这个函数删除一个先前注册的 XML 模式，该模式是通过 URL 或 XMLSchema 名称而得到标识的。

[0538] 在使用 Oracle XML DB 之前注册您的 XML 模式

[0539] 在 Oracle XML DB 在任何环境中可以使用或引用 XML 模式，首先必须对 XML 模式进行注册。XML 模式是通过使用 DBMS_XMLSCHEMA.registerSchema() 以及规定以下信息来注册的。

[0540] ●作为 VARCHAR、CLOB、XMLType 或 URIType 的 XML 模式源文档

[0541] ● XML 模式 URL。这是在 XML 实例文档内部用以规定与之符合的 XML 模式的位置的 XML 模式的名称。

[0542] 在结束注册之后：

[0543] ●符合这个 XML 模式并且使用 XML 文档内部的 XML 模式的 URL 来对其进行引用的 XML 文档可以由 Oracle XMLDB 加以处理。

[0544] ●可以为这个 XML 模式定义的根 XML 元素创建表格和列，以便保存相容的 XML 文档。

[0545] 使用 DBMS_XMLSCHEMA 来注册您的 XML 模式

[0546] 使用 DBMS_XMLSCHEMA 来注册您的 XML 模式。这其中包含了规定 XML 模式文档及其 URL，其中所述 URL 又称为 XML 模式位置。

[0547] 实例 5-3 使用 DBMS_XMLSCHEMA 来注册一个声明了 complexType 的 XML 模式

[0548] -- 设想以下显示的 XML 模式。它声明了一个名为 PurchaseOrderType 的

[0549] --complexType 以及一个这种类型的元素 PurchaseOrder。所述模式

[0550] -- 保存在 PL/SQL 变量 doc 中。

[0551] -- 下文中在 URL :http://www.oracle.com/PO.xsd 注册了 XML 模式。

[0552] declare

[0553] doc varchar2(1000): =' <schema

[0554] targetNamespace =" http://www.oracle.com/PO.xsd"

[0555] xmlns:po =" http://www.oracle.com/PO.xsd"

[0556] xmlns =" http://www.w3.org/2001/XMLSchema" >

[0557] <complexType name =" PurchaseOrderType" >

[0558] <sequence>

[0559] <element name =" PONum" type =" decimal" />

[0560] <element name =" Company" >

[0561] <simpleType>

[0562] <restriction base =" string" >
[0563] <maxLength value =" 100" />
[0564] </restriction>
[0565] </simpleType>
[0566] </element>
[0567] <element name =" Item" maxOccurs =" 1000" >
[0568] <complexType>
[0569] <sequence>
[0570] <element name =" Part" >
[0571] <simple Type>
[0572] <restriction base =" string" >
[0573] <maxLength value =" 1000" />
[0574] </restriction>
[0575] </simpleType>
[0576] </element>
[0577] <element name =" Price" type =" float" />
[0578] </sequence>
[0579] </element>
[0580] </complexType>
[0581] <complexType>
[0582] </sequence>
[0583] <element name =" PurchaseOrder " type =" po:PurchaseOrderType" />
[0584] </schema>' ;
[0585] begin
[0586] dbms_xmlschema.registerSchema(' http://www.oracle.com/PO.xsd' ,
[0587] doc) ;end ;
[0588] -- 已注册模式可用于已创建的基于 XMLSchema 的表格或是基于 XMLSchema 的列。或者
[0589] -- 举例来说,以下语句创建了一个具有基于 XMLSchema 的列的表格。
[0590] create table po_tab(
[0591] id number,
[0592] po sys.XMLType
[0593])
[0594] xmltype column po
[0595] XMLSCHEMA" http://www.oracle.com/PO.xsd"
[0596] element" PurchaseOrder" ;
[0597] -- 下文显示了一个与正在插入以上表格的前述 XML 模式相符合的 XMLType 实例。

[0598] -- 所述 schemaLocation 属性则规定了模式 URL：

[0599] insert into po_tab values(1,

[0600] xmltype(' <PurchaseOrder xmlns =" http://www.oracle.com/PO.xsd"

[0601] xmlns:xsi =" http://www.w3.org/2001/XMLSchema-instance"

[0602] xsi:schemaLocation =" http://www.oracle.com/PO.xsd"

[0603] http://www.oracle.com/PO.xs d" >

[0604] <PONum>1001</PONum>

[0605] <Company>Oracle Corp</Company>

[0606] <Item>

[0607] <Part>9i Doc Set</Part>

[0608] <Price>2550</Price>

[0609] </Item>

[0610] <Item>

[0611] <Part>8i Doc Set</Part>

[0612] <Price>350</Price>

[0613] </Item>

[0614] </PurchaseOrder>'))；

[0615] 局部和全局 XML 模式

[0616] XML 模式可以注册为局部或全局类型：

[0617] ●局部 XML 模式：作为局部模式注册的 XML 模式缺省是只可以由拥有者看到。

[0618] ●全局 XML 模式：作为全局模式注册的 XML 模式缺省是可以由所有数据库用户看到和使用的。

[0619] 当您注册一个 XML 模式时，DBMS_XMLSCHEMA 将一个与 XML 模式相对应的 Oracle XML DB源议案加到Oracle XML DB储存库中。所述XML模式URL根据以下规则来确定Oracle XML DB 中的源的路径名称。

[0620] 局部 XML 模式

[0621] 在 Oracle XML DB 中，局部 XML 模式源是在 /sys/schemas/< 用户名 > 这个目录下注册的。剩余路径名称来源于模式 URL。

[0622] 实例 5-4 局部 XML 模式

[0623] 举例来说，具有模式 URL 的局部 XML 模式：

[0624] http://www.myco.com/PO.xsd

[0625] 由 SCOTT 注册的，给出了以下路径名：

[0626] /sys/schemas/SCOTT/www.myco.com/PO.xsd。

[0627] 数据库用户需要适当的权限 (ACLs)，以便创建一个带有这个路径名的源，从而将 XML 模式注册为本地 XML 模式。

[0628] 作为默认情况，在向 Oracle XML DB 注册了 XML 模式之后，XML 模式才归属于您。针对 XML 模式的引用保存在 Oracle XML

[0629] DB 储存库中的以下目录：

[0630] /sys/schemas/< 用户名 >/…

[0631] 举例来说，如果你即 SCOTT 注册了前述的 XML 模式，则将其映射到文件：

[0632] /sys/schemas/SCOTT/www.oracle.com/PO.xsd

[0633] 这种 XML 称为本地的。通常，它们只能由其随归属的您才能看到。

[0634] 注意：通常，只有 XML 模式拥有者才能使用它来定义 XML 表格、列或视图，验证文档等等。然而，Oracle 完全支持那些可以如下规定的合格的 XML 模式 URL：

[0635] http://xmlns.oracle.com/xdb/schemas/SCOTT/www.oracle.com/PO.xsd

[0636] 这个扩展的 URL 可以由特许用户使用，以便规定归属于其他用户的 XML 模式。

[0637] 全局 XML 模式

[0638] 与本地模式相反，特许用户可以通过规定 DBMS_XMLSCHEMA 注册函数中的自变量来将 XML 模式注册为全局 XML 模式。

[0639] 全局模式可以为所有用户看到并且保存在 Oracle XML DB 储存库的目录 /sys/schemas/PUBLIC/ 之下。

[0640] 注意：针对这个目录所进行的访问受控于访问控制列表 (ACLs)，并且作为默认情况，所述目录只能由 DBA 写入。您需要关于这个目的的写入权限来注册全局模式。

[0641] 假设这个目录受到缺省的“受保护的”ACL 的保护，那么 XDBAdmin 角色还提供了针对这个目的的写入访问。

[0642] 您可以向同一 URL 注册一个局部模式，作为现有的全局模式。局部模式始终隐藏了具有相同名称 (URL) 的全局模式。

[0643] 实例 5-5 全局 XML 模式

[0644] 举例来说，SCOTT 向 URL：www.myco.com/PO.xsd 注册的全局模式是在 /sys/schemas/PUBLIC/www.myco.com/PO.xsd 注册到 Oracle XML DB 储存库的。

[0645] 数据库用户需要适当的权限 (ACLs) 来创建这个资源，以便将 XML 模式注册为全局类型。

[0646] 注册您的 XML 模式：Oracle XML DB 建立存储和访问架构

[0647] 作为注册 XML 模式的一部分，Oracle XML DB 还执行了若干个其他步骤来简化存储、访问和运算符合 XML 模式的 XML 实例。这些步骤包括：

[0648] ●创建类型：在注册 XML 模式的时候，Oracle 创建了允许无结构保存符合这个 XML 模式的 XML 文档的适当 SQL 对象类型。您可以使用 XML 模式文档中定义了 OracleXML DB 的属性来控制如何生成这些对象类型。

[0649] ●创建缺省表格：作为 XML 模式注册的一部分，Oracle XMLDB 为所有根元素产生了缺省的 XMLType 表格。您可以规定任何的列和表格等级约束条件，以便在表格创建过程中使用。

[0650] ●创建 Java beans：在 XML 模式注册过程中可以可选地产生 Java beans。这些 Java 类为模式中声明的元素和属性提供了取值和赋值方法。当 XML 模式众所周知的时候，使用 Java beans 所进行的访问为操作 XML 提供了更好的性能。这有助于避免运行时间的名称变换。

[0651] 使用 DBMS_XMLSCHEMA 来删除您的 XML 模式

[0652] 您可以通过使用 DBMS_XMLSCHEMA.deleteSchema 过程来删除您的已注册 XML 模式。当您尝试删除一个 XML 模式时，DBMS_XMLSCHEMA 将会检查：

[0653] ●当前用户是否具有适当权限 (ACLs) 来删除与 Oracle XMLDB 储存库内部的 XML 模式相对应的资源。由此您可以控制哪些用户可以通过对 XML 模式资源设定适当的 ACL 来删除哪些 XML 模式。

[0654] ●从属对象。如果存在任何从属对象,则产生一个差错并且删除操作失败。这称为删除 XML 模式的 RESTRICT 方式。

[0655] FORCE(强制)方式

[0656] FORCE 方式选项是在删除 XML 模式的时候提供的。如果您指定了 FORCE 方式选项,那么即使 XML 模式删除无法完成从属对象检查,所述删除也会继续进行。在这种情况下,XML 模式删除将其所有从属对象标记为无效。

[0657] CASCADE(级联)方式

[0658] CASCADE 方式选项丢弃所有生成的类型、缺省表格、以及作为先前调用注册模式一部分的 Java beans。

[0659] 还可以参见:Oracle9i XML API Reference-XDK and XDBchapter on DBMS_XMLSCHEMA

[0660] 实例 5-6 使用 DBMS_XMLSCHEMA 来删除 XML 模式

[0661] -- 以下实例删除了 XML 模式 PO.xsd。首先丢弃从属表格。

[0662] -- 然后结合 DBMS_XMLSCHEMA.DELETESCHEMA 而使用 FORCE 和 CASCADE 方式

[0663] -- 来阐述所述模式。

[0664] drop table po_tab ;

[0665] EXEC dbms_xmlschema.deleteSchema(' http://www.oracle.com/PO.xsd' ,

[0666] dbms_xmlschema.DELETE_CASCADE_FORCE) ;

[0667] 已注册 XML 模式的使用指南

[0668] 以下章节描述的是用于向 Oracle XML DB 注册 XML 模式的指南。

[0669] 依赖于已注册 XML 模式的对象

[0670] 以下对象依赖于一个已注册 XML 模式 :

[0671] ●具有一个符合 XML 模式中的某个元素的 XMLType 列的表格或视图。

[0672] ●包含或引入了这个模式以作为其定义一部分的 XML 模式

[0673] ●在例如 DBMS_XMLGEN 运算符中引用了 XML 模式名称的游标。应该注意的是,这些仅仅是暂态对象。

[0674] 创建 XMLType 表格、视图或列

[0675] 在注册了一个 XML 模式之后,可以通过引用以下信息来使用所述模式创建基于 XML 模式的 XMLType 表格、视图和列 :

[0676] ●已注册 XML 模式的 XML 模式 URL

[0677] 根元素的名称

[0678] 实例 5-7 注册之后创建一个 XML Type 表格

[0679] -- 举例来说,您可以如下创建一个基于 XML 模式的 XMLType 表格 :

[0680] CREATE TABLE po_tab OF XMLTYPE

[0681] XMLSCHEMA" http://www.oracle.com/PO.xsd" ELEMENT

[0682] " PurchaseOrder" ;

[0683] -- 以下语句插入了符合模式的数据

[0684] insert into po_tab values(

[0685] xmltype(' <PurchaseOrderxmlns =" http://www.oracle.com/PO.xsd"

[0686] xmlns:xsi =”http://www.w3.org/2001/XMLSchema-instance"

[0687] xsi:schemaLocation =" http://www.oracle.com/PO.xsd

[0688] http://www.oracle.com/PO.xsd" >

[0689] <PONum>1001</PONum>

[0690] <Company>Oracle Corp</Company>

[0691] <Item>

[0692] <Part>9i Doc Set</Part>

[0693] <Price>2550</Price>

[0694] </Item>

[0695] <Item>

[0696] <Part>8i Doc Set</Part>

[0697] <Price>350</Price>

[0698] </Item>

[0699] </PurchaseOrder>')) ;

[0700] 依靠 XML 模式来验证 XML 实例 :schemaValidate()

[0701] 您可以通过使用验证方法中的一种并依靠已注册的 XML 模式来验证一个 XMLType 实例。

[0702] 实例 5-8 使用 schemeValidate() 来验证 XML

[0703] -- 以下的 PL/SQL 实例依靠 XML 模式 PO.xsd 来验证一个 XML 实例 :

[0704] declare

[0705] xmldoc xmltype ;

[0706] begin

[0707] --(通过从表格获取) 来填充 xmldoc

[0708] select value(p)into xmldoc from po_tab p ;

[0709] -- 依靠 XML 模式来进行验证

[0710] xmldoc.schemavalidate() ;

[0711] if xmldoc.isschemavalidated() = 1then

[0712] dbms_output.put_line(' Data is valid') ;

[0713] else

[0714] dbms_output.put_line(' Data is invalid') ;

[0715] end if ;

[0716] end ;

[0717] 完全合格的 XML 模式 URLs

[0718] 作为默认情况,XML 模式 URL 名称总是在当前用户的范围内得到引用。换句话说,当数据库用户规定 XML 模式 URL 时,它们首先解析为当前用户拥有的局部 XML 模式的名称。

[0719] ●如果没有这种 XML 模式,则将它们解析成全局 XML 模式的名称。

[0720] ●如果没有全局 XML 模式,则 Oracle XML DB 产生一个错误。

[0721] 用户不能引用的 XML 模式

[0722] 作为默认情况,这些规则意味着用户不能引用以下类型的 XML 模式 :

[0723] ●不同数据库用户拥有的 XML 模式

[0724] ●与局部 XML 模式具有相同名称的全局 XML 模式

[0725] 完全合格的 XML 模式 URLs 允许显性引用 XL 模式 URLs

[0726] 为了允许在这些情况中显性引用 XML 模式,Oracle XML DB 支持一个完全合格的 XML 模式 URL 的概念。在这种形式中,拥有 XML 模式的数据库用户的名称同样规定为 XML 模式 URL 的一部分,只不过这种 XML 模式 URL 归属于 Oracle XML DB 命名空间,如下所示 :

[0727] http://xmlns.oracle.com/xdb/schemas/<database-user-name>/<schemaURL-minus-protocol>

[0728] 实例 5-9 使用完全合格的 XML 模式 URL

[0729] -- 例如设想具有以下 URL 的全局 XML 模式 :

[0730] http://www.example.com/po.xsd

[0731] -- 假设数据库用户 SCOTT 具有一个带有相同 URL 的局部 XML 模式 :

[0732] http://www.example.com/po.xsd

[0733] -- 用户 JOE 可以引用 SCOTT 所拥有的如下所示的局部 XML 模式 :

[0734] http://xmlns.oracle.com/xdb/schemas/SCOTT/www.example.com/po.xsd

[0735] -- 简单的说,用于全局 XML 模式的完全合格的 URL 是 :

[0736] http://xmlns.oracle.com/xdb/schemas/PUBLIC/www.example.com/po.xsd

[0737] XML 模式注册中的事务行为

[0738] XML 模式的注册是非事务性的并且是与其他 SQL DDL 操作一起自动提交的,如下所示 :

[0739] ●如果注册成功,则自动提交操作,

[0740] ●如果注册失败,则数据库回滚到注册开始之前的状态。

[0741] 由于 XML 模式注册有可能包含创建对象类型和表格,因此差错恢复包含了丢弃任何那些如此创建的类型和表格。由此确保了整个 XML 模式注册都是原子的。也就是说,所述注册要么成功,要么数据库恢复到注册开始之前的状态。

[0742] 在 XML 模式注册过程中产生 Java Bean

[0743] 在 XML 模式注册过程中可选地生成了 Java beans,并且所述 Java beans 为 XML 模式中声明的元素和属性提供了取值和赋值方法。当 XML 众所周知并且大多数固定的时候,通过避免运行时间的名称变换,使用 Java beans 而对数据库中保存的 XML 数据所进行的访问为操作 XML 数据提供了更好的性能。

[0744] 实例 5-10 在 XML 模式注册过程中产生 Java Bean 类

[0745] 举例来说,与 XML 模式 PO.xsd 相对应的 Java bean 类具有以下取值和赋值方法 :

[0746] public class PurchaseOrder extends XMLTypeBean

[0747] {

[0748] public BigDecimal getPONum()

[0749] {

[0750] }

[0751] public void setPONum(BigDecimal val)

[0752] {

[0753] }

[0754] public String getCompany()

[0755] {

[0756] }

[0757] public void setCompany(String val)

[0758] {

[0759] }

[0760] }

[0761] 注意：在Oracle XML DB中，Java Bean的支持仅仅针对基于XML模式的XML文档。不基于模式的XML文档可以使用OracleXML DB DOM API来进行操作。

[0762] 使用DBMS_XMLSCHEMA.generateSchema()来从对象关系类型中产生XML模式

[0763] XML模式可以从一个自动使用缺省映射的对象关系类型中产生。DBMS_XMLSCHEMA包内的generateSchema()和generateSchemas()函数接受了一个具有对象名称的串和另一个具有Oracle XML DB XMLschema的串。

[0764] ● generateSchema()返回一个包含了XML模式的XMLType。它可选地为给定对象类型引用或是只受限于顶级类型的所有类型产生XML模式。

[0765] ● generateSchemas()与之相似，只不过它返回的是XML模式的一个XMLSequenceType，其中每一个XML模式对应于一个不同的命名空间。它还可以获取一个附加的可选自变量，从而规定优选XML模式位置的根URL：

[0766] http://xmlns.oracle.com/xdb/schemas/<schema>.xsd

[0767] 它们可以可选地产生带有注释的XML模式，所述XML模式可用于向Oracle XML DB注册XML模式。

[0768] 实例5-11产生XML模式：使用generateSchema()

[0769] --举例来说，在给定对象类型的情况下：

[0770] connect t1/t1

[0771] CREATE TYPE employee_t AS OBJECT

[0772] (

[0773] empno NUMBER(10)；

[0774] ename VARCHAR2(200)；

[0775] salary NUMBER(10,2)

[0776])；

[0777] --如下所示，我们可以为这个类型产生模式：

[0778] select dbms_xmlschema.generateschema('T1', 'EMPLOYEE_T')from dual；

[0779] --这个操作返回的是一个与类型EMPLOYEE T相对应的模式。所述模式

[0780] --声明了一个名为EMPLOYEE_T的元素以及一个名为EMPLOYEE_TType的

COMPLEXTYPE。

[0781] --所述模式包括来自 http://xmlns.oracle.com/xdb 的其他注释。

[0782] DBMS_XMLSCHEMA.GENERATESCHEMA(' T1' , ' EMPLOYEE_T')

[0783] ----------

[0784] <xsd:schema targetNamBSpace = " http://ns.oracle.com/xdb/T1 " xmlns =" http:

[0785] //ns.oracle.com/xdb/T1 " xmlns:xsd = " http://www.w3.org/2001/XMLSchema"

[0786] xmlns:xdb =" http://xmlns.oracle.com/xdb" xmlns:xsi =" http://www.w3.org/

[0787] 2001/XMLSchema-instance" xsi:schemaLocation =" http://xmlns.oracle.com/

[0788] xdb http://xmlns.oracle.com/xdb/XDBSchema.xs d" >

[0789] <xsd:element name =" EMPLOYEE_T" type =" EMPLOYEE_TType" xdb:

[0790] SQLType =”EMPLOYEE_T" xdb:SQLSchema =" T1" />

[0791] <xsd:complexType name =" EMPLOYEE_TType" >

[0792] <xsd:sequence>

[0793] <xsd:element name =" EMPNO" type =" xsd:double" xdb:SQLName

[0794] =”EMPNO" xdb:S QLType =" NUMBER" />

[0795] <xsd:element name =" ENAME”type =" xsd:string" xdb:SQLName

[0796] =" ENAME”xdb:SQL1VPe =" VARCHAR2" />

[0797] <xsd:element name =" SALARY" type =”xsd:double" xdb:SQLName

[0798] =" SALARY" xdb:SQLType =" NUMBER" />

[0799] </xsd:sequence>

[0800] </xsd:complexType>

[0801] </xsd:schema>

[0802] 涉及 XML 模式的 XMLType 的方法

[0803] 表 5-1 列举了与 XMLType API 的 XML 模式有关的方法。

[0804] 表 5-1 涉及 XMLTypeAPIXML 模式的方法

[0805]

XMLType API 方法	描述
isSchemaBased ( )	如果 XMLType 实例基于一个 XML 模式，则返回 TRUE，否则返回 FALSE。
getSchemaURL ( ) getRootElement ( ) getNamespace ( )	返回 XML 模式 URL、根元素的名称以及基于 XML 模式的 XMLType 实例的命名空间。
schemaValidate ( ) isSchemaValid ( ) isSchemaValidated ( ) setSchemaValidated ( )	依靠一个已注册XML模式，通过使用所述验证方法，可以对一个 XMLType 实例进行验证。

[0806] 管理和保存 XML 模式

[0807] XML 模式模式文档自身作为 XMLType 实例保存在 Oracle

[0808] XMLDB 中。基于 XML 模式的 XMLType 类型和表格是作为 Oracle

[0809] XML DB 安装脚本 catxdbs.sql 的一部分而被创建的。

[0810] 根 XML 模式，XDBSchema.xsd

[0811] 用于多个 XML 模式的 XML 模式称为根 XML 模式 XDBSchema.xsd。XDBSchema.xsd 描述了可以由 Oracle XML DB 注册的任何有效的 XML 模式文档。您可以通过位于 /sys/schema/PUBLIC/xmlns.oracle.com/xdb/XDBSchema.xsd 的 Oracle XML DB 储存库来访问 XDBSchema.xsd。

[0812] 如何保存基于 XML 模式的 XMLType 结构？

[0813] 基于 XML 模式的 XMLType 结构是使用以下方法中的一种而得到保存的：

[0814] ●在基础对象类型列中，这是默认的保存机制。

[0815] -可选地，SQL 对象类型可以在 XML 模式注册处理中创建。

[0816] -从 XML 到 SQL 对象类型和属性的映射作为附加注释即 Oracle XML DB 与 http://xmlns.oracle.com/xdb 中定义的属性保存在 XML 模式文档中。

[0817] ●在一个单独的基础LOB列中。在这里，存储选择是在CREATE TABLE语句的STORE AS 从句中规定的：

[0818] CREATE TABLE po_tab OF xmltype

[0819] STORE AS CLOB

[0820] ELEMENT" http://www.oracle,com/po.xsd#purchaseOrder" ;

[0821] 规定存储机制

[0822] 您可以规定根据一个基于特定 XML 模式的映射来保存表格和列，而不是使用 STORE AS 从句。您可以为用于映射的 XML 模式指定 URL。

[0823] 不基于 XML 模式的 XML 数据可以通过使用 CLOB 而被保存在表格中。然而这样的话，您并未得到诸如索引、查询改写等等的益处。

[0824] DOM 保真度

[0825] 对 DOM 遍历来说，文档对象模型 (DOM) 保真度是相对于原始 XML 文档而对所检索

的 XML 文档结构加以保持的概念。为了确保 Oracle XML DB 中保存的 XML 文档的精度和完整性，需要所述 DOM 保真度。

[0826] Oracle XML DB 如何就 XML 模式来确保 DOM 保真度

[0827] 在 XML 模式中声明的所有元素和属性都被映射，以便分离出相应 SQL 对象类型中的属性。然而，XML 实例文档中的某些信息并不是由这些元素或属性直接表示的，例如：

[0828] ●注释

[0829] ●命名空间声明

[0830] ●前缀信息

[0831] 为了确保数据的完整和精确，举例来说，当重新产生数据库中保存的 XML 文档的时候，Oracle XML DB 使用了一个名为 DOM 保真度的数据完整性机制。

[0832] DOM 保真度涉及的是返回的 XML 文档与原始 XML 文档相比是否一致，尤其是在用于 DOM 遍历目的的情况下。

[0833] DOM 保真度和 SYS_XDBPD$

[0834] 为了确保 DOM 保真度得到保持以及返回的 XML 文档与用于 DOM 遍历的原始 XML 文档一致，Oracle XML DB 向每个所创建的对象类型添加了一个系统二进制属性 SYS_XDBPD$。

[0835] 这个属性保存的是不能保存在任何其他属性中的所有信息，由此确保 Oracle XML DB 中保存的所有 XML 文档的 DOM 保真度。这种信息的实例包括：排序信息、注释、处理指令、命名空间前缀等等。

[0836] 而这个属性则映射到一个位置描述符 (PD) 列中。

[0837] 注意：通常，设置这个信息并不是一个很好的主意，因为如果没有 PD 列，那么诸如注释、处理指令等等的附加信息有可能会丢失。

[0838] 如何排除 SYS_XDBPD$

[0839] 如果不需要 DOM 保真度，那么您可以通过设定属性 maintainDOM = FALSE 来排除 XML 模式定义中的 SYS_XDBPD$。

[0840] 注意：为了清楚起见，在这里的很多实例中省略了属性 SYS_XDBPD$。然而，所述属性在 XML 模式注册处理生成的所有 SQL 对象类型中始终是作为位置描述符 (PD) 列而给出的。

[0841] Oracle XML DB 创建基于 XML 模式的 XMLType 表格和列

[0842] Oracle XML DB 通过引用以下信息来创建基于 XML 模式的 XMLType 表格和列：

[0843] ●已注册 XML 模式的 XML 模式 URL

[0844] ●根元素的名称

[0845] 图 3 显示了用于创建一个 XMLType 表格的语法：

[0846] CREATE TABLE[schema.]table OF XMLTYPE

[0847] [XMLTY PE XMLType_storage][XMLSchema_spec]；

[0848] 在以下实例中显示的 XPointer 标志的一个子集也可用于提供一个包含了 XML 模式位置和元素名称的单个 URL。

[0849] 实例 5-12 创建基于 XML 模式的 XMLType 表格

[0850] 这个实例使用了处于给定的 URL 的 XML 模式来创建

[0851] XMLType 表格 po_tab：

[0852] CREATE TABLE po_tab OF XMLTYPE

[0853] XMLSCHEMA" http://www.oracle.com/PO.xsd" ELEMENT

[0854] " PurchaseOrder" ;

[0855] 一个等价的定义是：

[0856] CREATE TABLE po_tab OF XMLTYPE

[0857] ELEMENT" http://www.oracle.com/PO.xsd#PurchaseOrder" ;

[0858] SQL 对象关系类型保存基于 XML 模式的 XMLType 表格

[0859] 在注册一个 XML 模式的时候，Oracle XML DB 创建了能够结构化保存符合这个 XML 模式的 XML 文档的适当的 SQL 对象类型。作为默认情况，所有 SQL 对象类型都是基于当前注册的 XML 模式来创建的。

[0860] 实例 5-13 创建 SQL 对象类型来保存 XMLType 表格

[0861] --举例来说，在向 Oracle XML DB 注册 PO.xsd 的时候，以下的 SQL 类型将被创建。

[0862] --应该注意的是，所述类型的名称是所生成的名称并且没有必要

[0863] --匹配 Itemxxx t、Itemxxx COLL 以及 PurchaseOrderTypexxx_T，

[0864] --其中 xxx 是一个三位整数。

[0865] CREATE TYPE" Itemxxx_T" as obj ect

[0866] (

[0867] part varchar2(1000),

[0868] price number

[0869]) ;

[0870] CREATE TYPE" Itemxxx_COLL" AS varray(1000)OF" Item_T"

[0871] CREATE TYPE" PurchaseOrderTypexxx_T" AS OBJECT

[0872] (

[0873] ponum number,

[0874] company varchar2(100),

[0875] item Item_varray_COLL

[0876]) ;

[0877] 注意：先前实例中的对象类型和属性的名称可以是由系统产生的。

[0878] ●如果 XML 模式已经包含了在所填充的 SQLName、SQLType 或 SQLColType，则将这个名称用作对象属性名称。

[0879] ●如果 XML 模式不包含 SQLName 属性，则所述名称来源于 XML 名称，除非因为长度或冲突原因而不能使用所述名称。

[0880] 如果使用了 SQLSchema 属性，则 Oracle XML DB 尝试使用规定的数据库模式来创建对象类型。当前用户则必须具有必要的权限来执行这个操作。

[0881] 在注册 XML 模式之前使用 SQLName 和 SQLType 属性来指定 SQL 对象类型名称

[0882] 在注册 XML 模式之前，为了规定所生成的 SQL 对象的特定名称，在 XML 模式定义中包含了属性 SQLName 和 SQLType。

[0883] ●如果您规定了 SQLName 和 SQLType 的值，那么 OracleXML DB 将会创建使用了这些名称的 SQL 对象类型。

[0884] ●如果您没有规定这些属性，那么 Oracle XML DB 使用的是系统产生的名称。

[0885] 注意：您不必为这些属性中的任何一个规定数值。Oracle XMLDB 在 XML 模式注册处理中填充了适当的值。然而，在这里建议您至少对顶级的 SQL 类型的名称加以规定，以使您能够稍后对其进行引用。

[0886] 所有注释都采用了可以在属性和元素声明内部规定的属性的形式。这些属性归属于 Oracle XML DB 命名空间：http://xmlns.oracle.com/xdb

[0887] 表 5-2 列举了您可以在元素和属性声明中规定的 Oracle XMLDB 属性。

[0888] 表 5-2 您可以在元素中规定的属性

[0889]

属性	值	缺省值	描述
SQLName	任何 SQL 标识符	元素名称	规定映射到这个 XML 元素的 SQL 对象内部的属性名称
SQLType	任何 SQL 类型名称	从元素名称中产生的名称	规定对应于这个 XML 元素声明的 SQL 类型的名称
SQLCollType	任何 SQL 集合类型名称	从元素名称中产生的名称	规定与这个 maxOccurs>1 的 XML 元素相对应的 SQL 集合类型的名称
SQLSchema	任何 SQL	注册 XML	拥有 SQLType 所规定的类型的

[0890]

	用户名	模式的用户	数据库用户的名称
SQLCollSchema	任何 SQL 用户名	注册 XML 模式的用户	拥有 SQLCollType 所规定的类型的数据库用户的名称
maintainOrder	true|false	true	如果为真，则将集合映射成 VARRAY。如果为 false，则将集合映射成一个 NESTED TABLE（嵌套表格）
SQLInline	true|false	true	如果为 true，则将这个元素作为一个嵌入属性联机保存（如果 maxOccurs > 1，则将其作为集合）。如果为 false，则保存一个 REF（如果 maxOccurs>1，则保存一个 REF 集合）。在 SQL 不支持联机的某些情况下（类似循环引用），将这个属性强制设定为 false。
maintainDOM	true|false	true	如果为 true，则保存这个元素的实例，以使它们在输出上保留 DOM 保真度。这意味着除了元素的排序之外，所有注释、处理指令、命名空间声明等等都得到保留。如果为 false，则不必保证输出具有与输出相同的 DOM 行为。
columnProps	任何有效的列存储从句	NULL	规定的是插入缺省的 CREATE TABLE 语句的列存储从句。它主要用于那些映射到表格的元素，也就是顶级元素声明和离线元素声明。
tableProps	任何有效的表格存储从句	NULL	规定的是附加于缺省的 CREATE TABLE 语句的 TABLE 存储从句。它主要对全局和离线元素有意义。
defaultTable	任何表格名称	基于元素名称	规定的是这个模式的 XML 实例所要存入的表格的名称。它在将 XML 从例如 FTP、HTTP 之类的没有规定表格名称的 APIs 插入的时候最有用。
beanClassname	任何 Java 类名	从元素名称中产生	可以在元素声明内部使用。如果元素基于一个全局 complexType，那么这个名称必须与 complexType 声明内部的 beanClassname 值一致，如果

[0891]

			用户规定了一个名称，那么 bean 生成将会产生一个带有这个名称的 bean 类，而不是从元素名称中产生一个名称。
JavaClassname	任何 Java 类名	无	用于指定从相应 bean 类中导出的 Java 类的名称，以确保在 bean 访问过程中例示这个类的一个对象。如果没有规定 JavaClassname，则 Oracle XML DB 将会直接例示所述 bean 类的一个对象。

[0892] 表 5-3 您可以在声明全局 complexType 的元素中规定的属性

[0893]

属性	值	缺省值	描述
SQLType	任何 SQL 类型名称	从元素名称中产生的名称	规定对应于这个 XML 元素声明的 SQL 类型的名称
SQLSchema	任何 SQL 用户名	注册 XML 模式的用户	拥有 SQLType 所规定的类型的数据库用户的名称
beanClassname	任何 Java 类名	从元素名称中产生	可以在元素声明内部使用。如果元素基于一个全局 complexType，那么这个名称必须与 complexType 声明内部的 beanClassname 值一致，如果用户规定了一个名称，那么 bean 生成将会产生一个带有这个名称的 bean 类，而不是从元素名称中产生一个名称。
maintainDOM	true|false	true	如果为 true，则保存这个元素的实例，以使它们在输出上保留 DOM 保真度。这意味着除了元素排序之外，所有注释、处理指令、命名空间声明等等都得到保留。如果为 false，则不必保证输出具有与输出相同的 DOM 行为。

[0894] 表 5-4 您可以在 XML 模式声明中规定的属性

[0895]

属性	值	缺省值	描述
mapUnboundedStringToLob	true|false	false	如果为 true，则默认将无限制的串映射成 CLOB。同样，无限制的二进制数据默认映射成 BLOB。如果为 false，则将无限制的串映射成 VARCHAR2（4000）并且将无限制的二进制分量映射成 RAW（2000）。
storeVarrayAsTable	true|false	false	如果为 true，则将 VARRAY 作为表格（OCT）加以保存。如果为 false，则将 VARRAY 保存在一个 LOB 中。

[0896] SQL 映射是在注册过程中在 XML 模式中规定的

[0897] 涉及 SQL 映射的信息保存在 XML 模式文档中。注册处理产生 SQL 类型，并且向 XML 模式文档添加注释，以便保存映射信息。在这里，这些映射采用了新属性的形式。

[0898] 实例 5-14 使用 SQLType 和 SQLName 属性来获取 SQL 映射

[0899] -- 以下的 XML 模式定义显示了如何使用 SQLType 和 SQLName 属性

[0900] -- 来获取 SQL 映射信息：

[0901] declare

[0902] doc varchar2(3000):='<schema

[0903] targetNamespace="http://www.oracle.com/PO.xsd"

[0904] xmlns:po="http://www.oracle.com/PO.xsd" xmlns:xdb="http://xmlns.oracle.

[0905] com/xdb"

[0906] xmlns="http://www.w3.org/2001/XMLSchema">

[0907] <complexType name="PurchaseOrderType">

[0908] <sequence>

[0909] <element name="PONum" type="decimal" xdb:SQLName=

[0910] "PONUM" xdb:SQLType="NUMBER"/>

[0911] <element name="Company" xdb:SQLName="COMPANY" xdb:

[0912] SQLType="VARCHAR2">

[0913] <simpleType>

[0914] <restriction base="string">

[0915] <maxLength value="100"/>

[0916] </restriction>

[0917] </simpleType>

[0918] </element>

[0919] <element name="Item" xdb:SQLName="ITEM" xdb:SQLType=

[0920] "ITEM_T" maxOccurs="1000">

[0921] <complexType>

[0922] <sequence>
[0923] <element name ＝" Part" xdb:SQLName ＝" PART" xdb:SQLTyPe
[0924] ＝" VARCHAR2" >
[0925] </simpleTyPe>
[0926] <restriction base ＝" string" >
[0927] <maxLength value ＝" 1000" />
[0928] </restriction>
[0929] </simpleType>
[0930] </element>
[0931] <element name ＝ " Price " type ＝ " float " xdb:SQLName
＝" PRICE"
[0932] xdb:SQLType ＝" NUMBER" />
[0933] </sequence>
[0934] </complexType>
[0935] </element>
[0936] </sequence>
[0937] </complexType>
[0938] <element name＝" PurchaseOrder" Type＝" po:PurchaseOrderType/" >
[0939] </schema>' ;
[0940] begin
[0941] dbms_xmlschema.registerSchema(' http://www.oracle.com/PO.xsd ' ,
doc) ;
[0942] end ;

[0943] 图 4 显示了 Oracle XML DB 如何使用一个 XML 文档和 XML 模式中规定的映射来创建基于 XML 文档的 XMLType 表格。首先创建的是一个 XMLType 表格，并且根据 XML 模式中如何规定存储，XML 文档进行映射并且作为一个 CLOB 保存在一个 XMLType 列中，或是以对象关系形式加以保存并在表格中展开到若干个列。

[0944] 首先创建的是一个 XMLType 表格，并且根据 XML 模式中如何规定存储，XML 文档进行映射并且作为一个 CLOB 保存在一个 XMLType 列中，或是以对象关系形式加以保存并在表格中展开到若干个列。

[0945] 使用 DBMS_XMLSCHEMA 来映射类型

[0946] 使用 DBMS_XMLSCHEMA 来为属性和元素设定类型信息的映射。

[0947] 设定属性映射类型信息

[0948] 属性声明可以具有根据以下信息中的一个所规定的类型：

[0949] ●原始类型

[0950] ●在这个 XML 模式或外部 XML 模式内部声明的全局 simpleType

[0951] ●针对在这个 XML 模式或外部 XML 模式内部声明的全局属性（ref ＝" .." ）

[0952] 局部 simpleType

[0953] 在所有情况中，SQL 类型和相关信息（长度和精度）以及存储映射信息都是从属

性所基于的 simpleType 中导出的。

[0954] 覆盖 SQL 类型

[0955] 您可以在输入 XML 模式文档中显性规定一个 SQLType 值。在这种情况下，您规定的类型将被验证。这样就顾及了以下特定形式的覆盖：

[0956] ●如果缺省类型是一个 STRING，那么您可以用 CHAR、VARCHAR 或 CLOB 中的任何一个将其覆盖。

[0957] ●如果缺省类型是 RAW，那么您可以使用 RAW 或 BLOB 来将其覆盖。

[0958] 设定元素映射类型信息

[0959] 元素声明可以根据以下信息之一来规定它的类型：

[0960] ●用于为属性声明规定类型的任何方式

[0961] ●在这个 XML 模式文档内部或是外部 XML 模式中规定的全局 complexType。

[0962] ●对于一个全局元素的引用（ref ＝" ..." ），所述引用自身可以处于这个 XML 模式文档内部或是外部 XML 模式之中。

[0963] ●局部 complexType。

[0964] 覆盖 SQL 类型

[0965] 基于 complexType 的元素默认映射成一个包含了与每个子元素和属性相对应的属性的对象类型。然而，您可以通过为输入 XML 模式中的 SQLType 属性显性规定一个值来覆盖这个映射。在这个实例中允许下列这些关于 SQLType 的值：

[0966] ● VARCHAR2

[0967] ● RAW

[0968] ● CLOB

[0969] ● BLOB

[0970] 这些值表示的是以文本或非快速增长的形式而将 XML 保存在数据库中。在这里对下列特殊情况进行了处理：

[0971] ●如果检测到一个循环，作为处理用于声明元素的 complexType 以及在 complexType 内部声明的元素的一部分，SQLInline 属性强制为" false"，并且正确的 SQL 映射设定为 REF XMLTYPE。

[0972] ●如果 maxOccurs ＞ 1，则需要创建一个 VARRAY 类型。

[0973] -如果 SQLInline ＝" true"，则创建一个 varray 类型，所述类型的元素类型是先前确定的 SQL 类型。

[0974] *VARRAY 的基数是基于 maxOccurs 属性的值来确定的。

[0975] *VARRAY 类型的名称是由使用 SQLCollType 属性的用户显性规定的，或者是通过毁损元素名称来获取的。

[0976] -如果 SQLInline ＝" false"，则将 SQL 类型设定为 XDB.XDB$XMLTYPE_REF_LIST_T，它是一个表示 REF 数组到 XMLType 的预定类型。

[0977] ●如果所述元素是一个全局元素，或者如果 SQLInline ＝" false"，则需要创建一个缺省表格。它被添加到表格创建上下文中。缺省表格的名称要么由用户规定，要么通过毁损元素名称来加以规定。

[0978] XML 模式：将 SimpleType 映射成 SQL

[0979] 这一节描述的是如何使用 XML 模式定义来将 XML 模式 simpleType 映射成 SQL 对象类型。

[0980] 表 5-5 到表 5-8 列举了 XML 模式定义中规定的 XML 模式 simpleType 与 SQL 的缺省映射，例如

[0981] ● XML 原是类型映射成最接近的 SQL 数据类型。例如，DECIMAL（小数）、POSITIVEINTEGER（正整数）和 FLOAT（浮点）都映射成了 SQL 数字。

[0982] ● XML 枚举类型映射成带有单个 RAW(n) 属性的对象类型。n 的值是由枚举声明中可能的值的数目来确定的。

[0983] ● XML 列表或并集数据类型映射成 SQL 中的一个串 (VARCHAR2/CIOB) 数据类型。

[0984] 表 5-5 将 XML 串映射成 SQL

[0985]

XML 原始类型	长度或 MaxLength 平面	缺省映射	兼容的数据类型
串	n	如果 n<4000，则是 VARCHAR2（n），否则是 VARCHAR2（4000）	CHAR VARCHAR2 CLOB
串	--	如果 mapUnboundedStringToLob="true"，则为 VARCHAR2（4000），CLOB	CHAR VARCHAR2 CLOB

[0986] 表 5-6 将 XML 二进制数据类型 (hexBinary/base64Binary) 映射成 SQL

[0987]

XML 原始类型	长度或 MaxLength 平面	缺省映射	兼容的数据类型
hexBinary base64Binary	n	如果 n<2000，则是 RAW（n），否则是 RAW（2000）	RAW,BLOB
hexBinary base64Binary	-	如果 mapUnboundedStringToLob ="true",则为 RAW（2000），BLOB	RAW,BLOB

[0988] 表 5-7 将数字型 XML 原始类型缺省映射成 SQL

[0989]

XML 简单类型	缺省的 Oracle 数据类型	totalDigits（m）、规定的 fractionDigits（n）	兼容的数据类型
float（浮点）	NUMBER	NUMBER（m,n）	NUMBER,FLOAT,DOUBLE
double（双精度）	NUMBER	NUMBER（m,n）	NUMBER,FLOAT,DOUBLE
decimal（小数）	NUMBER	NUMBER（m,n）	NUMBER,FLOAT,DOUBLE
integer（整数）	NUMBER	NUMBER（m,n）	NUMBER,FLOAT,DOUBLE
nonNegativeInteger（非负整数）	NUMBER	NUMBER（m,n）	NUMBER,FLOAT,DOUBLE
positiveInteger（正整数）	NUMBER	NUMBER（m,n）	NUMBER,FLOAT,DOUBLE
nonPositiveInteger（非正整数）	NUMBER	NUMBER（m,n）	NUMBER,FLOAT,DOUBLE
negativeInteger（负整数）	NUMBER	NUMBER（m,n）	NUMBER,FLOAT,DOUBLE
long（长整型）	NUMBER（20）	NUMBER（m,n）	NUMBER,FLOAT,DOUBLE

[0990]

unsignedLong（无符号长整型）	NUMBER（20）	NUMBER（m,n）	NUMBER,FLOAT,DOUBLE
int（整数）	NUMBER（10）	NUMBER（m,n）	NUMBER,FLOAT,DOUBLE
unsignedInt（无符号整数）	NUMBER（10）	NUMBER（m,n）	NUMBER,FLOAT,DOUBLE
short（短整型）	NUMBER（5）	NUMBER（m,n）	NUMBER,FLOAT,DOUBLE
unsignedshort（无符号短整型）	NUMBER（5）	NUMBER（m,n）	NUMBER,FLOAT,DOUBLE
byte（字节）	NUMBER（3）	NUMBER（m,n）	NUMBER,FLOAT,DOUBLE
unsignedbyte（无符号字节）	NUMBER（3）	NUMBER（m,n）	NUMBER,FLOAT,DOUBLE

[0991] 表 5-8 将 XML 日期数据类型映射成 SQL

[0992]

XML 原始类型	缺省映射	兼容的数据类型
datetime	TIMESTAMP	DATE
time	TIMESTAMP	DATE
date	DATE	DATE
gDay	DATE	DATE
gMonth	DATE	DATE
gYear	DATE	DATE
gYearMonth	DATE	DATE
gMonthDay	DATE	DATE
duration	VARCHAR2（4000）	none

[0993] 表 5-9 将其他 XML 原始数据类型默认映射成 SQL

[0994]

XML 简单类型	缺省的 Oracle 数据类型	兼容的数据类型
boolean	RAW（1）	VARCHAR2
Language（string）	VARCHAR2（4000）	CLOB,CHAR
NMTOKEN（string）	VARCHAR2（4000）	CLOB,CHAR
NMTOKENS（string）	VARCHAR2（4000）	CLOB,CHAR
Name（string）	VARCHAR2（4000）	CLOB,CHAR
NCName（string）	VARCHAR2（4000）	CLOB,CHAR
ID	VARCHAR2（4000）	CLOB,CHAR
IDREF	VARCHAR2（4000）	CLOB,CHAR
IDREFS	VARCHAR2（4000）	CLOB,CHAR
ENTITY	VARCHAR2（4000）	CLOB,CHAR
ENTITIES	VARCHAR2（4000）	CLOB,CHAR
NOTATION	VARCHAR2（4000）	CLOB,CHAR

[0995]

anyURI	VARCHAR2（4000）	CLOB,CHAR
anyType	VARCHAR2（4000）	CLOB,CHAR
anySimpleType	VARCHAR2（4000）	CLOB,CHAR
QName	XDB.XDB$QNAME	--

[0996] simpleType：将 XML 串映射成 SQL VARCHAR2 或 CLOB

[0997] 如果 XML 模式将数据类型规定为 maxLength 的值小于 4000 的串，则将其映射成一个具有规定长度的 VARCHAR2 属性。然而，如果在 XML 模式中并未规定 maxLength，那么只能将其映射成 LOB。在大多数串值很小并且其中只有很少一部分大到足以需要一个 LOB 时，这种处理是次最佳的。参见图 5。

[0998] 图 5-3 Oracle XML DB：将 XML 串映射成 SQL VARCHAR2 或 CLOB。

[0999] XML 模式 :将 ComplexTypes 映射成 SQL

[1000] 通过使用 XML 模式,可以将一个 complexType 映射成一个 SQL 对象类型,如下所示 :

[1001] ● 在 complexType 内部声明的 XML 属性映射成对象属性。定义 XML 属性的 simpleType 确定了相应属性的 SQL 数据类型。

[1002] ●在 complexType 内部声明的 XML 元素还映射成对象属性。所述对象属性的数据类型是由定义 XML 元素的 simpleType 或 complexType 来确定的。

[1003] 如果 XML 元素是结合了 maxOccurs ＞ 1 而得到声明的,那么在 SQL 中将其映射成集合属性。所述集合可以是一个 VARRAY(缺省情况),如果将 maintainOrder 属性设定为 false,那么所述集合也可以是一个嵌套表格。此外,VARRAY 默认保存在表格中的有序集合 (OCTs) 而不是 LOBs 中。您可以通过将 storeAsLob 属性设定为 true 来选择 LOB 存储。

[1004] 也可以参见 :“第 5-70 页上的“表格中的有序集合 (OCT)”

[1005] 将 complexType 映射成 SQL :对离线存储来说,将 SQLInLine 属性设定为 FALSE

[1006] 作为默认情况,子元素映射成一个嵌入式对象属性。然而也存在这样的情况,其中离线存储提供了更好的性能。在这类情况下,可以将 SQLInline 属性设定为 false,并且 Oracle XML DB 产生一个带有嵌入式 REF 属性的对象类型。REF 指向与离线存储的 XML 分段相对应的 XMLType 的另一个实例。此外还创建了缺省的 XMLType 表格,以便保存离线分段。

[1007] 图 6 描述了为了离线存储而将一个 complexType 映射成 SQL。

[1008] 实例 5-15 Oracle XML DB XML 模式 :complexType 映射 -- 为了进行离线存储,将 SQLInLine 属性设定为 FALSE

[1009] -- 为了进行离线存储,将属性设定为 false

[1010] -- 在这个实例中,元素 Addr 的属性 xdb:SQLInLine 设定为 false。

[1011] -- 最终得到的对象类型 OBJ_T2 具有一个带有嵌入的 REF 属性的 XMLType 类型的列。

[1012] --REF 属性指向所创建的表格 Addr_tab 中的 OBJ_T1 的另一个 XMLType。

[1013] --Addr_tab 具有列 Street 和 City。后一个 XMLType 实例是离线保存的。

[1014] declare

[1015] doc varchar2(3000) : =′ <schema xmlns =″ http://www.w3.org/2001/XML

[1016] Schema″ targetNamespace =″ http://www.oracle.com/emp.xsd″

[1017] xmlns:emp =″ http://www.oracle.com/emp.xsd″

[1018] xmlns:xdb =″ http://xmlns.oracle.com/xdb″ >

[1019] <complexType name =″ Employee″ xdb:SQLType =″ OBJ_T2″ >

[1020] <sequence>

[1021] <element name =″ Name″ type =”string″ />

[1022] <element name =″ Age″ type =″ decimal″ />

[1023] <element name =″ Addr″ xdb:SQLInline =″ false″ >

[1024] <complexType xdb:SQLType =″ OBJ_T1″ />

[1025] <sequence>
[1026] <element name =" Street" type =" string" />
[1027] <element name =" City" type =" string" />
[1028] </sequence>
[1029] </complexType>
[1030] </element>
[1031] </sequence>
[1032] </complexType>
[1033] </schema>' ;
[1034] begin
[1035] dhms_xmlschema.registerSchema(' http://www.oracle.com/PO.xsd' , doc):
[1036] end ;
[1037] -- 在注册这个 XML 模式时,Oracle XML DB 产生以下类型和 XMLType 表格 :
[1038] CREATE TYPE OBJ_T1AS OBJECT
[1039] (
[1040] SYS_XDBPD$XDB.XDB$RAW_LIST_T,
[1041] Street VARCHAR2(4000),
[1042] City VARCHAR2(4000)
[1043]) ;
[1044] CREATE TYPE OBJ_T2AS OBJECT
[1045] (
[1046] SYS_XDBPD$XDB.XDB$RAW_LIST_T,
[1047] Name VARCHAR2(4000),
[1048] Age NUMBER,
[1049] Addr REF XMLType
[1050]) ;
[1051] 将 complexType 映射成 SQL :将 XML 分段映射成大型对象 (LOB)
[1052] 您可以将用于一个 complex 的 SQLType 规定成一个字符大型对象 (CLOB) 或二进制大型对象 (BLOB)。在这里,整个 XML 分段都保存在 LOB 属性中。这在 XML 文档的某些部分很少得到查询但是大多数被作为单条信息检索或保存的时候非常有用。通过将 XML 分段保存为 LOB,您可以节省分析 / 分解 / 重组方面的开销。
[1053] 实例 5-16Oracle XML DB XML 模式 :XML 分段到 LOB 的 complex Type 映射
[1054] -- 在以下实例中,XML 模式规定了 XML 分段的元素 Addr
[1055] -- 正在使用属性 SQLType =" CLOB"
[1056] declare
[1057] doc varchar2(3000): =' <schema xmlns =" http://www.w3.org/2001/XML
[1058] Schema" targetNamespace =" http://www.oracle.com/emp.xsd"

[1059] xmlns:emp ＝" http://www.oracle.com/emp.xsd"

[1060] xmlns:xdb ＝" http://xmlns.oracle.com/xdb" >

[1061] <complexType name ＝" Employee" xdb:SQLType ＝" OBJ_T2" >

[1062] <sequence>

[1063] <element name ＝" Name" type ＝" string" />

[1064] <element name ＝" Age" type ＝" decimal" />

[1065] <element name ＝" Addr" xdb:SQLType ＝" CLOB" >

[1066] <complexType>

[1067] <sequence>

[1068] <element name ＝" Street" type ＝" string" />

[1069] <element name ＝" City" type ＝" string" />

[1070] </sequence>

[1071] </complexType>

[1072] </element>

[1073] </sequence>

[1074] </complexType>

[1075] </schema>' ;

[1076] begin

[1077] dbms xmlschema.registerSchema(' http://www.oracle.com/PO.xsd ' , doc) ;

[1078] end ;

[1079] -在注册这个 XML 模式时,Oracle XML DB 产生以下类型和 XMLType 表格 :

[1080] CREATE TYPE OBJ_T AS OBJECT

[1081] (

[1082] SYS_XDBPD$XDB.XDB$RAW_LI ST_T,

[1083] Name VARCHAR2(4000),

[1084] Age NUMBER,

[1085] Addr CLOB

[1086]) ;

[1087] 图 7 将 complexType 分段映射成字符大型对象 (CLOBs)

[1088] Oracle XML DB complexType 的扩展和限制

[1089] 在 XML 模式中,complexType 是基于 complexContent(复杂内容) 和 simpleContent(简单内容) 来声明的。

[1090] ● simpleContent 是作为 simpleType 的扩展来声明的。

[1091] ● complexContent 是作为以下类型中的一种来声明的 :

[1092] -基础类型

[1093] -complexType 扩展

[1094] -complexType 限制

[1095] XML 模式中的 complexType 声明 :处理继承

[1096] 对 complexType 来说，Oracle XML DB 是如下所述来对 XML 模式中的继承进行处理的：

[1097] ●对声明成扩展其他 complexTypes 的 complexTypes 来说，与基础类型相对应的 SQL 类型是作为当前 SQL 类型的父型来规定的。只有子型 complexTypes 中声明的附加属性和元素才可以作为属性添加给子对象类型。

[1098] ●对声明成限制其他 complexTypes 的 complexTypes 来说，用于子复杂类型的 SQL 类型设定成与用于其基础类型的 SQL 类型相同。这是因为 SQL 并不支持通过继承机制来限制对象类型。任何约束都是由 XML 模式中的限制所施加的。

[1099] 实例 5-17XML 模式中的继承：作为 complexTypes 的扩展的 complexContent

[1100] -- 设想一个定义了基础 complexType “Address”和

[1101] -- 两个扩展“USAddress”与“IntlAddress”的 XML 模式

[1102] declare

[1103] doc varchar2(3000) := ' <xs:schema

[1104] xmlns:xs =" http://www.w3.org/2001/XMLSchema"

[1105] xmlns:xdb =" http://xmlns.oracle.com/xdb" >

[1106] <xs:complexType name =" Address" xdb:SQLType =" ADDR_T" >

[1107] <xs:sequence>

[1108] <xs:element name =" street" type =" xs:string" />

[1109] <xs:element name =" city" type =" xs:string" />

[1110] </xs:sequence>

[1111] </xs:complexType>

[1112] <xs:complexType name =" USAddress" xdb:SQLType =" USADDR_T" >

[1113] <xs:complexContent>

[1114] <xs:extension base =" Address" >

[1115] <xs:sequence>

[1116] <xs:element name =" zip" type =" xs:string" />

[1117] </xs:sequence>

[1118] </xs:extension>

[1119] </xs:complexContent>

[1120] </xs:complexType>

[1121] <xs:complexType name =" IntlAddress" final =" #all" xdb:SQLType

[1122] =" INTLADDR_T" >

[1123] <xs:complexContent>

[1124] <xs:extension base =" Address" >

[1125] <xs:sequence>

[1126] <xs:element name =" country" type =" xs:string" />

[1127] </xs:sequence>

[1128] </xs:extension>

[1129] </xs:complexContent>

[1130] </xs:complexType>

[1131] </xs:schema>' ;

[1132] Begin

[1133] dbms_xmlschema.registerSchema(' http://www.oracle.com/PO.xsd ' , doc) ;

[1134] end ;

[1135] --注意 :类型 INTLADDR_T 是作为最终类型创建的,因为对象的 complexType 规定了“最终”属性。

[1136] --作为默认情况,所有 complexType 都可以由其他类型扩展和限制,

[1137] --因此,所有对象类型都不是作为最终类型创建的。

[1138] create type ADDR_T as object(

[1139] SYS_XDBPD$XDB.XDB$RAW_LIST_T,

[1140] " street" varchar2(4000),

[1141] " city" varchar2(4000)

[1142])not final ;

[1143] create type USADDR_T under ADDR_T(

[1144] " zip" varchar2(4000)

[1145])not final ;

[1146] create type INTLADDR_T under ADDR_T(

[1147] " country" varchar2(4000)

[1148])final ;

[1149] 实例 5-18XML 模式中的继承 :complexType 中的限制

[1150] --设想一个定义了基础 complexType Address 和继承了 country 属性规定

[1151] --的受限类型 LocalAddress 的 XML 模式

[1152] declare

[1153] doc varchar2(3000) : =' <xs:schema

[1154] xmlns:xs =”http://www.w3.org/2001/XMLSchema"

[1155] xmlns:xdb =" http://xmlns.oracle.com/xdb" >

[1156] <xs:complexType name =" Address" xdb:SQLType =" ADDR_T" >

[1157] <xs:sequence>

[1158] <xs:element name =" street" type =" xs:string" />

[1159] <xs:element name =" city" type =" xs:string" />

[1160] <xs:element name =" zip" type =" xs:string" />

[1161] <xs:element name =" country" type =" xs:string" minOccurs =" 0" max

[1162] Occurs =" 1" />

[1163] </xs:sequence>

[1164] </xs:complexType>

[1165] <xs:complexType name =" LocalAddress" xdb:SQLType =" USADDR_T" >

[1166] <xs:complexContent>
[1167] <xs:restriction base ＝″ Address″ >
[1168] <xs:sequence>
[1169] <xs:element name ＝″ street″ type ＝″ xs:string″ />
[1170] <xs:element name ＝″ city″ types″ xs:string″ />
[1171] <xs:element name ＝″ zip″ type ＝″ xs:string″ />
[1172] <xs:element name ＝″ country″ type ＝″ xs:string″
[1173] minOccurs ＝″ 0″ maxOccurs ＝″ 0″ />
[1174] </xs:sequence>
[1175] </xs:restriction>
[1176] </xs:complexContent>
[1177] </xs:complexType>
[1178] </xs:schsma>′ ;
[1179] begin
[1180] dbms_xmlschema.registerSchema(′ http://www.oracle.com/PO.xsd′ , doc) ;
[1181] end ;
[1182] -- 由于 SQL 中的继承支持不支持限制标志,因此与受限 complexType 相对应的 SQL
[1183] -- 类型是双亲对象类型的一个空的子类型。对以上的 XML 模式来说,产生了以下的 SQL 类型 :
[1184] create type ADDR_T as object(
[1185] SYS_XDBPD$XDB.XDB$RAW_LIST_T,
[1186] ″ street″ varchar2(4000),
[1187] ″ city″ varchar2(4000),
[1188] ″ zip″ varchar2(4000),
[1189] ″ country″ varchar2(4000)
[1190])not final ;
[1191] Create type USADDR_T under ADDR_T ;
[1192] 映射 complexType :simpleContent 到对象类型
[1193] 基于一个 simpleContent 声明的 complexType 映射成一个具有与 XML 属性相对应的属性以及一个与实体值相对应的附加 SYS_XDBBODY 属性的对象类型。实体属性的数据类型则基于定义了实体类型的 simpleType。
[1194] 实例 5-19XML 模式 complexType :将 complexType 映射成 simpleContent
[1195] declare
[1196] doc varchar2(3000): ＝ ′ <schema xmlns ＝ ″ http://www.w3.org/2001
[1197] /XMLSchema″ targetNamespace ＝”http://www.oracle.com/emp.xsd″
[1198] xmlns:emp ＝″ http://www.oracle.com/emp.xsd″

[1199] xmlns:xdb =" http://xmlns.oracle.conv/xdb" >

[1200] <complexType name =" name" xdb:SQLType =" OBJ_T" >

[1201] <simpleContent>

[1202] <restriction base =" string" >

[1203] </restriction>

[1204] </simpleContent>

[1205] </complexType>

[1206] </schema>' ;

[1207] begin

[1208] dbms_xmlschema.registerSchema(' http://www.oracle.con/emp.xsd ' , doc) ;

[1209] end ;

[1210] --在注册这个 XML 模式时,Oracle XML DB 产生以下类型和 XMLType 表格 :

[1211] create type OBJ_T as object

[1212] (

[1213] SYS_XDBPD$xdb.xdb$raw_list_t,

[1214] SYS_XDBB ODY$VARCHAR2(4000)

[1215]) ;

[1216] 映射 complexType :Any 和 AnyAttributes

[1217] Oracle XML DB 在创建的对象类型中将元素声明 any 和属性声明 anyAttribute 映射成 VARCHAR2 属性 (也可以映射成大型对象 (LOB))。所述对象属性保存的是与 any 声明相匹配的 XML 分段的文本。

[1218] namespace 属性可用于限制内容,以使它们归属于一个指定的命名空间。

[1219] ● any 元素声明内部的 processContents 属性表示的是匹配 any 声明的内容所需要的验证等级。

[1220] 实例 5-20 Oracle XML DB XML 模式 :将 complexType 映射成 Any/AnyAttribute

[1221] --这个 XML 模式实例声明了一个 any 元素并且在对象类型 OBJ_T 中将其映射成列 SYS_XDBANY$。

[1222] --这个元素还声明了属性 processContents 省略了对匹配 any 声明的内容进行验证。

[1223] declare

[1224] doc varchar2(3000): =' <schema xmlns =" http://www.w3.org/2001

[1225] /XMLSchema"

[1226] targetNamespace =" http://www.oracle.com/any.xsd"

[1227] xmlns:emp =" http://www.oracle.com/any.xsd"

[1228] xmlns:xdb =" http://xmlns.oracle.com/xdb" >

[1229] <complexType name =" Employee" xdb:SQLType =" OBJ_T" >

[1230] <sequence>

[1231] <element name =" Name" type =" string" />

[1232] <element name =" Age" type =" decimal" />

[1233] <any namespace =" http://www/w3.org/2001/xhtml" processContents =

[1234] " skip" />

[1235] </sequence>

[1236] </complexType>

[1237] </schema>' ;

[1238] begin

[1239] dbms_xmlschema.registerSchema(' http://www.oracle.com/emp.xsd' doc) ;

[1240] end ;

[1241] --它导致产生以下语句 :

[1242] CREATE TYPE OBJ_TAS OBJECT

[1243] (

[1244] SYS_XDBPD$xdb.xdb$raw_list_t,

[1245] Name VARCHAR2(4000),

[1246] Age NUMBER,

[1247] SYS_XDBANY$VARCHAR2(4000)

[1248]) ;

[1249] 对 XML 模式中的 complexTypes 之间的循环进行处理

[1250] 由于对象类型不允许循环,因此在产生对象类型的时候,通过在循环结束的点上引入一个 REF 属性来中断 XML 模式中的循环。这样一来,一部分数据是离线保存的,当对其进行检索的时候,这些数据仍旧归属于双亲 XML 文档。

[1251] 实例 5-21XML 模式 :complexTypes 之间的循环

[1252] XML 模式允许在 comoplexTypes 定义之间进行循环。图 5 ～ 6 显示了这个实例,其中 complexType CT1 的定义可以引用另一个 complexType CT2,而 CT2 的定义引用了第一类型 CT1。

[1253] --XML 模式允许在 complexTypes 定义之间进行循环,这是一个长度为 2 的循环的实例 :

[1254] declare

[1255] doc varchar2(3000): = ' <xs:schema xmlns:xs = " http://www.w3.org/2001/

[1256] XMLSchema" xmlns:xdb =" http://xmlns.oracle.com/xdb" >

[1257] <xs:complexType name =" CT1" xdb:SQLType =" CT1" >

[1258] <xs:sequence>

[1259] <xs:element name =”e1" type =" xs:string" />

[1260] <xs:element name =" e2" type =" CT2" />

[1261] </xs:sequence>

[1262] </xs:complexType>

[1263] <xs:complex Type name =" CT2" xdb:SQLType =" CT2" >

[1264] <xs:sequence>

[1265] <xs:element name =" e1" type =" xs:string" />

[1266] <xs:element name =" e2" type =" CT1" />

[1267] </xs:sequence>

[1268] </xs:complexType>

[1269] </xs:schema>' ;

[1270] begin

[1271] dbms_xmlschema.registerSchema(' http://www.oracle.com/emp.xsd ' , doc) ;

[1272] end ;

[1273] SQL 类型不允许在类型定义中进行循环。然而,它们支持弱循环,也就是包含了 REF(引用)属性的循环。因此,循环 XML 模式定义映射成 SQL 对象类型,以便通过在适当的点强制使 SQLInline =" false" 来避免任何循环。这样就创建了一个弱循环。

[1274] -- 对先前 XML 模式来说,产生了以下 SQL 类型 :

[1275] create type CT1 as object

[1276] (

[1277] SYS_XDBPD$xdb.xdb$raw_list_t,

[1278] " e1" varchar2(4000),

[1279] " e2" ref xmltype ;

[1280])not final ;

[1281] create type CT2 as object

[1282] (

[1283] SYS_XDBPD$xdb.xdb$raw_list_t,

[1284] " e1" varchar2(4000),

[1285] " e2" CT1

[1286])not final ;

[1287] 图 8 同一 XML 模式中的不同 complexTypes 之间的交叉引用

[1288] 实例 5-22XML 模式 :complexTypes 之间的循环,自引用

[1289] -- 循环 complexType 的另一个实例包括声明具有一个对其自身的引用的 complexType。

[1290] -- 下文是一个对自身进行引用的类型 <SectionT> 的实例。

[1291] declare

[1292] doc varchar2(3000) : =' <xs:schema

[1293] xmlns:xs =" http://www.w3.org/2001/XMLSchema"

[1294] xmlns:xdb =" http://xmlns.oracle.com/xdb" >

[1295] <xs:complexType name =" SectionT" xdb:SQLType =" SECTION_T" >

[1296] <xs:sequence>

[1297] <xs:element name =" title" type =" xs:string" />

[1298] <xs:choice maxOccurs ＝″ unbounded″ >

[1299] <xs:element name ＝″ body″ type ＝″ xs:string″ xdb:SQLCollType ＝

[1300] ″ BODY_COLL″ />

[1301] <xs:element name ＝″ section″ type ＝″ SectionT″ />

[1302] </xs:choice>

[1303] </xs:sequence>

[1304] </xs:complexType>

[1305] </xs:schema>′ ;

[1306] begin

[1307] dbms_xmlschema.registerSchema(′ http://www.oracle.com/section.xsd′ ,doc) ;

[1308] end ;

[1309] -- 在这里产生了以下的 SQL 类型。

[1310] -- 注意 :section 属性是作为一个 REFs 的 varray 声明给 XMLType 实例的。

[1311] -- 由于可能不止一次的出现嵌入部分,因此所述属性是 VARRAY。

[1312] -- 并且它是一个针对 XMLType 的 REF 的 VARRAY,以免形成 SQL 对象的一个循环。

[1313] create type BODY_COLL as varray(32767)of VARCHAR2(4000) ;

[1314] create type SECTION_T as object

[1315] (

[1316] SYS_XDBPD$xdb.xdb$raw_list_t,

[1317] ″ title″ varchar2(4000),

[1318] ″ body”BODY_COLL,

[1319] ″ section″ XDB.XDB$REF_LIST_T

[1320])not final ;

[1321] 关于创建基于 XML 模式的 XML 表格的进一步准则

[1322] 假设由“http://www.oracle.com/PO.xsd”所标识的您的 XML 模式已经进行了注册。那么可以创建一个 XMLType 表格 myPOs,以便以一种如下所示的对象关系格式来保存符合这个 XML 模式的元素 PurchaseOrder 的实例 :

[1323] CREATE TABLE MyPOs OF XMLTYPE

[1324] ELEMENT″ http://www.oracle.com/PO.xsd#PurchaseOrder″ ;

[1325] 图 9 示意性描述了 complexType 如何引用或循环其自身。

[1326] 图 9XML 模式内部的 complexType 自引用

[1327] 用于创建基于 XML 模式的 XML 表格的进一步准则

[1328] 假设由“http://www.oracle.com/PO.xsd”所标识的您的 XML 模式已经进行了注册。那么可以创建一个 XMLType 表格 myPOs,以便以一种如下所示的对象关系格式来保存符合这个 XML 模式的元素 PurchaseOrder 的实例 :

[1329] CREATE TABLE MyPOs OF XMLTYPE

[1330] ELEMENT″ http://www.oracle.com/PO.xsd#PurchaseOrder″ ;

[1331] 在这里将会创建隐藏的列。这些列对应于 PurchaseOrder 元素所映射的对象类型。此外还创建了一个 XMLExtra 对象列,以便保存顶级实例,例如命名空间声明。

[1332] 注意 :XMLDATA 是 XMLType 允许的一个伪属性。

[1333] 规定 XMLType CREATE TABLE 语句中的存储从句

[1334] 为了指定存储,可以使用 Object 或 XML 标志而在 XMLType 存储从句中引用基础列。

[1335] ● Object 标志 :XMLDATA.<attr1>.<attr2>...

[1336] 举例来说 :

[1337] CREATE TABLE MyPOs OF XMLTYPE

[1338] ELEMENT" http://www.oracle.com/PO.xsd#PurchaseOrder"

[1339] lob(xmldata.lobattr)STORE AS(tablespace...) ;

[1340] ● XML 标志 :extractValue(xmltypecol, ' /attr1/attr2')

[1341] 举例来说 :

[1342] CREATE TABLE MyPOs OF XMLTYPE

[1343] ELEMENT" http://www.oracle.com/PO.xsd#PurchaseOrder"

[1344] lob(ExtractValue(MyPOs, ' /lobattr'))STORE AS(tablespace...) ;

[1345] 使用 CREATE INDEX 来引用 XMLType 列

[1346] 如先前实例中所示,可以使用 Object 或 CREATE TABLE 语句中的 XML 标志来对处于 XMLType 列下方的那些列进行引用。同样的情况在 CREATE INDEX 语句中也是成立的。

[1347] CREATE INDEX ponum_idx ON MyPOs(xmldata.ponum) ;

[1348] CREATE INDEX ponum_idx ON MyPOs p(ExtractValue(p, ' /ponum')) ;

[1349] 规定 XMLType 列的约束条件

[1350] 在这里可以使用 Object 或 XML 标志来为基础 XMLType 列规定约束条件。

[1351] ● Object 标志

[1352] CREATE TABLE MyPOs OF XMLTYPE

[1353] ELEMENT" http://www.oracle.com/PO.xsd#PurchaseOrder"

[1354] (unique(xmldata.ponum)) ;

[1355] ● XML 标志

[1356] CREATE TABLE MyPOs P OF XMLTYPE

[1357] ELEMENT

[1358] " http://www.oracle.com/PO.xsd#PurchaseOrder" (unique(ExtractValue

[1359] (p, ' /ponum')

[1360]) ;

[1361] 将新的实例插入 XMLType 列

[1362] 新实例可以如下所示插入一个 XMLType 列中 :

[1363] INSERT INTO MyPOs VALUES

[1364] (xmltype.createxml(' <PurchaseOrder>....</PurchaseOrder>')) :

[1365] 使用基于模式的对象关系存储的查询改写

[1366] 什么是查询改写?

[1367] 当在使用 XML 模式和查询的结构化存储（对象关系类型）中保存 XMLType 以及使用那些利用了 XPath 的查询时，它们都被写入，以便直接转到基础对象相关列中。由此允许在列中存在 BTree 或其他索引的情况下使用所述 B*Tree 或其他索引，以便在优化程序所进行的查询评估中使用所述 B*Tree 或其他索引。这个查许改写机制被用于 SQL 函数中的 XPath，例如 existsNode()、extract()、extractValue() 以及 updateXML()。这样则允许依靠 XML 文档来评估 XPath，但却不必在存储器中构造 XML 文档。

[1368] 实例 5-23 查询改写

[1369] 举例来说，诸如

[1370] SELECT VALUE(p)FROM MyPOs p

[1371] WHERE extractValue(value(p), ' /PurchaseOrder/company ') =' Oracle' ;

[1372] 这样的查询尝试获取 Company 元素的值并且将其与文字‘Oracle’进行比较。由于结合基于 XML 的对象关系存储创建了 MyPOs 表格，因此 extractValue 运算符重写那些保存了 purchaseOrder 的 company 信息的基础相关列。

[1373] 因此，先前查询改写成：

[1374] SELECT VALUE(p)FROM MyPOs p

[1375] WHERE p.xmldata.company =' Oracle' ;

[1376] 如果存在一个在 Company 列上创建的正规索引，例如：

[1377] CREATE INDEX company_index ON MyPos e

[1378] (extractvalue(valuer(e), ' /PurchaseOrder/Company'));

[1379] 那么先前查询会将索引用于其评估。

[1380] 何时需要出现查询改写？

[1381] 查询改写是为以下 SQL 而进行的：

[1382] ● extract()

[1383] ● existsNode()

[1384] ● extractValue()

[1385] ● updateXML

[1386] 所述改写是为这些 SQL 函数进行的，其中这些函数可以在查询、DML 或 DDL 语句中的任何表达式中给出。举例来说，您可以使用 extractValue() 而在基础相关列上创建索引。

[1387] 实例 5-24SELECT 语句和查询改写

[1388] -- 这个实例获取的现有的购物订单

[1389] SELECT EXTRACTVALUE(value(x), ' /PurchaseOrder/Company')

[1390] FROM MYPOs x

[1391] WHERE EXISTSNODE(value(x), ' /PurchaseOrder/Item[1]/Part') = 1 ;

[1392] 这里给出了通过改写来使用基础列的语句的某些实例。

[1393] 实例 5-25DML 语句和查询改写

[1394] -- 这个实例删除了 Company 并非 Oracle 的所有 purchaseorder

[1395] DELETE FROM MYPOs x

[1396] WHERE EXTRACTVALUE(value(x), ' /PurchaseOrder/Company') =' Oracle Corp' ;

[1397] 实例 5-26CREA TE INDEX 语句和查询改写

[1398] -- 这个实例在 Company 列上创建了一个索引 -- 由于这个是以对象关系的形式创建的并且进行了查询改写,因此将会在基础相关列上创建一个规则的索引 :

[1399] CREATE INDEX company_index ON MyPos e

[1400] (EXTRACTVALUE(value(e), ' /PurchaseOrder/Company'))

[1401] 在这种情况下,如果 SQL 函数的改写导致产生一个简单的相关列,则将索引转换成一个 BTree 或是列上的一个域索引,而不是转换成一个基于函数的索引。

[1402] 将 XPath 表达式写成什么?

[1403] XPath 包含了没有通配符或递减坐标轴的简单表达式,所述 XPath 将会得到改写。XPath 可以选择一个元素或属性节点。此外在这个版本中还支持谓语并且将其转换成了基础的 SQL 查询。

[1404] 表 5-10 支持转换成基础 SQL 查询的 XPath 表达式

[1405]

用于转换的 XPath 表达式	描述
简单 XPath 表达式: /PurchaseOrder/@PurchaseDate/PurchaseOrder/Company	只包含了对象类型属性上的遍历, 其中所述属性是简单标量或对象类型自身。 唯一支持的坐标轴是子代和属性坐标轴
集合遍历表达式: /PurchaseOrder/Item/Part	包含了集合表达式的遍历。唯一支持的坐标轴是子代和属性坐标轴。如果在 CREATE INDEX 或 updateXML ( ) 过程中使用了 SQL 运算符,则不支持 SQL 集合遍历。
谓语: [Company="Oracle"]	XPath 中的谓语改写成 SQL 谓语, 这些谓语并不是为 updateXML ( ) 改写的。
列表索引: lineitem[1]	索引将被改写, 以便访问集合中的第 n 项, 这些索引并不是为 updateXML ( ) 改写的。

[1406] 不支持的 XPath 结构以下 XPath 结构没有被改写 :

[1407] ● XPath 函数

[1408] ● XPath 变量引用

[1409] ●除子代和属性坐标轴之外的所有坐标轴

[1410] ●通配符和递减表达式

[1411] ● UNION(并集) 操作

[1412] 不支持的 XMLSchema 结构下列 XML 模式结构并未得到支持。这意味着如果 XPath 表达式包含具有以下 XML 模式结构的节点,那么整个表达式都不会得到改写 :

[1413] ●对包含开放内容即任何内容的元素的子元素进行访问的 XPath 表达式。当节点包含内容时,所述表达式不能改写,只有当 any 针对的是一个不同于 XPath 中规定的命名空

间的时候，才可以使用相似方式来对 any 属性进行处理。

[1414] ● CLOB 存储。如果 XML 模式将部分元素定义映射成一个 SQL CLOB，则不支持遍历这类元素的 XPath 表达式。

[1415] ●枚举类型。

[1416] ●可替换元素。

[1417] 标量类型的非默认映射。举例来说，数字类型映射成了诸如本地整数等等的本地存储。

[1418] ● 用于继承的 complexTypes 的子代访问，其中所述子代不是所声明的 complexType 的一个成员。

[1419] 例如，设想这样一种情况，其中我们具有一个 addresscomplexType，它具有一个 street 元素。我们还可能具有一个名为 shipAddr 的导出类型，所述类型包含了 shipmentNumber 元素。如果 PurchaseOrder 具有一个类型为 address 的 address 元素，那么类似“/PurchaseOrder/address/street”的 XPath 将会得到改写，而“/PurchaseOrder/address/shipmentNumber”则不会得到改写。

[1420] ●非压缩数据类型的操作，诸如布尔型加上数字型。

[1421] 如何改写 XPath ？

[1422] 以下这节使用了在较早章节中描述的同一 purchaseOrder 模式来说明如何改写函数。

[1423] 实例 5-27 在对象类型生成过程中改写 XPath

[1424] -- 设想以下 purchaseorder 模式 ：

[1425] declare

[1426] doc varchar2(1000): =' <schema

[1427] targetNamespace =" http://www.oracle.com/PO.xsd"

[1428] xmlns:po = " http://www.oracle.com/PO.xsd " xmlns = " http:/www.w3.org/2001/

[1429] XMLSchema"

[1430] elementFormDefault =" qualified" >

[1431] <complexType name =" PurchaseOrderType" >

[1432] <sequence>

[1433] <element name =" PONum" type =" decimal" />

[1434] <element name =" Conpany" >

[1435] <simpleType>

[1436] <restriction base =" string" >

[1437] <maxLength value =" 100" />

[1438] </restriction>

[1439] </simpleType>

[1440] </element>

[1441] <element name =" Item" maxOccurs =" 1000" >

[1442] <complexType>

```
[1443]            <sequence>
[1444]              <element name =" Part" >
[1445]                <simpleType>
[1446]                  <restriction base =" string" >
[1447]                    <maxLength value =" 1000" />
[1448]                  </restriction>
[1449]                </simpleType>
[1450]                </element>
[1451]                <element name =" Price" type =" float"/>
[1452]              </sequence>
[1453]            </complexType>
[1454]          </element>
[1455]        </sequence>
[1456]      </complexType>
[1457]    <element name =" PurchaseOrder" type =" po:PurchaseOrderType" />
[1458]  </schema>' ;
[1459]  begin
[1460]     dbms_xmlschema.registerSchema( '  http://www.oracle.com/PO.xsd  ' ,
doc) ;
[1461]  end ;
[1462]  -- 创建一个符合这个模式的表格
[1463]  CREATE TABLE MyPOs OF XMLTYPE
[1464]  ELEMENT" http://www.oracle.com/PO.xsd#PurchaseOrder" ;
[1465]  -- 插入的 XML 文档在插入之前是依靠所述模式部分更新的
[1466]  insert into MyPos values(xmltype(' <PurchaseOrder
[1467]  xmlns =" http://www.oracle.com/PO.xsd"
[1468]    xmlns:xsi =" http://www.w3.org/2001/XMLSchema-instance"
[1469]    xsi:schemaLocation =" http://www.oracle.com/PO.xsd"
[1470]  http://www.oracle.com/PO.xsd" >
[1471]      <PONum>1001</PONum>
[1472]      <Company>Oracle Corp</Company>
[1473]      <Item>
[1474]        <Part>9i Doc Set</Pary>
[1475]        <Price>2550</Price>
[1476]      </Item>
[1477]      <Item>
[1478]        <Part>8i Doc Set</Part>
[1479]        <Price>350</Price>
[1480]      </Item>
```

[1481] </PurchaseOrder>'));

[1482] 由于 XML 模式没有对保持排序做出任何规定,因此缺省情况是保持排序和 DOM 保真度。这样一来,所述类型具有 SYS_XDBPD$ 属性,以便保存保持节点排序以及获取注释、处理指令等附加项所需要的附加信息。

[1483] SYS_XDBPD$ 属性还保持了关于元素的存在信息(即是否所述元素存在于输入文档之中)。对具有标量内容的元素来说,这个信息是必要的,因为所述元素会映射成简单的相关列。在这种情况下,空的和丢失的标量元素将会映射成列中的 NULL 值,并旦只有 SYS_XDBPD$ 可以帮助区分这两种情况。查询改写机制顾及了 SYS_XDBPD$ 的存在与否并且适当的改写了查询。

[1484] 假设将这个 XML 模式注册到模式 URL :

[1485] http://www.oracle.com/PO.xsd

[1486] 您可以结合该模式来 po_tab 表格 :

[1487] CREATE TABLE po_tab OF XMLTYPE

[1488] XMLSCHEMA " http://www.oracle.com/PO.xsd" ELEMENT" PurchaseOrder" ;

[1489] 现在,这个表格具有一个隐藏的 XMLData 列,其类型是“PurchaseOrder_T”,保存的则是实际数据。

[1490] 改写 XPath 表达式 :映射类型和问题

[1491] XPath 表达式映射类型和主题是在以下的章节中描述的 :

[1492] ●“关于简单 XPath 的映射”

[1493] ●“关于标量节点的映射”

[1494] “谓语的映射”

[1495] ●“集合谓语的映射”

[1496] ●“结合集合遍历的文档排序”

[1497] ●“集合索引”

[1498] ●“无法满足的 XPath 表达式”

[1499] ●“命名空间处理”

[1500] ●“日期格式转换”

[1501] 关于简单 XPath 的映射

[1502] 对于简单 XPath 的改写包含了访问于 XPath 表达式相对应的属性。表 5-11 列举了 XPath 映射 :

[1503] 表 5-11 用于 purchaseOrder XML 模式的简单 XPath 映射

[1504]

XPath 表达式	映射成
/PurchaseOrder	列 XMLData
/PurchaseOrder/@PurchaseDate	列 XMLData."PurchaseDate"
/PurchaseOrder/PONum	列 XMLData."PONum"
/PurchaseOrder/Item	集合的 XMLData.Item"的元素
/PurchaseOrder/Item/Part	集合 XMLData."Item"中的属性"Part"

[1505] 关于标量节点的映射

[1506] XPath 表达式可以包含一个 text() 运算符，它映射成 XML 文档中的标量内容。在改写的时候，它直接映射成基础的相关列。

[1507] 举例来说，XPath 表达式" /PurchaseOrder/PONum/text()" 直接映射成 SQL 列 XMLData." PONum" 。

[1508] PONum 列中的 NULL 值意味着文本值不可用，这要么是因为文本节点并未在输入文档中给出，要么是因为元素自身丢失。但与访问标量元素相比，这种操作更为有效，因为我们不需要检查 SYS_XBDPD$ 属性中是否存在所述元素。

[1509] 举例来说，XPath " /PurchaseOrder/PONum " 还映射成 SQL 属性 XMLData." PONum" 。

[1510] 然而在这种情况下，查询请求 (reqrite) 还必须使用 XMLData 列中的 SYS_XDBPD$ 来检查所述元素是否存在。

[1511] 谓语的映射谓语映射成了 SQL 谓语表达式。

[1512] 实例 5-28 映射谓语

[1513] -- 举例来说，XPath 表达式

[1514] /PurchaseOrder[PONum = 1001and Company =" Oracle Corp")

[1515] -- 中的谓语映射成 SQL 谓语 :

[1516] (XMLData." PONum" = 20and XMLData." Company" =" Oracle Corp")

[1517] -- 举例来说，以下查询将会改写成对象关系等价物，并且不会需要 XPath 的函数评估。

[1518] select extract(value(p), ' /PurchaseOrder/Item').getClobval()

[1519] frommypos p

[1520] where existsNbde(value(p),' PurchaseOrder[PONum = 1001 and Company

[1521] =" Oracle Corp"]') = 1 ;

[1522] 集合谓语的映射 XPath 表达式可以包括带有集合表达式的关系运算符。在 XPath 1.0 中，包含集合的条件是存在的。换句话说，即使集合中的一个成员满足条件，但是表达式仍然是成立的。

[1523] 实例 5-29 映射集合谓语

[1524] -- 举例来说，XPath 中的集合谓语 :

[1525] /PurchaseOrder[Items/Price > 200]

[1526] -- 映射成 SQL 集合表达式 :

[1527] EXISTS(SELECT null

[1528] FROM TABLE(XMLDATA.″ Item″)x

[1529] WHERE x.″ Price″ ＞ 200)

[1530] --举例来说,以下查询将会映射成对象关系等价物

[1531] select extract(value(p), ′ /PurchaseOrder/Item′).getClobval()

[1532] from mypos p

[1533] where existsNode(value(p), ′ /PurchaseOrder[Item/Price ＞ 400]′) ＝ 1 ;

[1534] 更复杂的改写出现在您具有一个集合<条件>集合的时候。在这种情况下,如果至少有一个来自这些两个集合的自变量的节点组合满足条件,则将谓语视为是得到满足的。

[1535] 实例 5-30 使用 existsNode() 来映射集合谓语

[1536] --举例来说,设想一个虚构的 XPath,所述 XPath 执行检查,以便了

[1537] --解是否 PurchaseOrder 具有 Items,以使 item 的 price 与某个 part 的数字相等 :

[1538] /PurchaseOrder[Items/Price ＝ Items/Part]

[1539] --映射成一个 SQL 集合表达式 :

[1540] EXISTS(SELECT null

[1541] FROM TABLE(XMLDATA.″ Item″)x

[1542] WHERE EXISTS(SELECT null

[1543] FROM TABLE(XMLDATA.″ Item″)y

[1544] WHERE y.″ Part″ ＝ x.″ Price″))

[1545] --举例来说,以下查询将会改写成对象关系等价物

[1546] select extract(value(p), ′ /PurchaseOrder/Item′).getClobval()

[1547] from mypos p

[1548] where existsNode(value(p), ′ /PurchaseOrder[Item/Price ＝ Item/Part]′) ＝ 1 ;

[1549] 带有集合遍历的文档排序大多数改写保存了原始文档排序。然而,由于 SQL 并不保证子查询结果的排序。因此当使用 extract() 函数从集合中选择元素时,合成的节点可以没有按照文档顺序排列。

[1550] 实例 5-31 带有集合遍历的文档排序

[1551] --举例来说 :

[1552] SELECT extract(value(p), ′ /PurchaseOrder/Item[Price ＞ 2100]/Part′)

[1553] FROM mypos p ;

[1554] --改写成使用如下所示的子查询 :

[1555] SELECT(SELECT XMLAgg(XMLForest(x.″ Part″ AS″ Part″))

[1556] FROM TABLE(XMLData.″ Item″)x

[1557] WHERE x.″ Price″ ＞ 2100)

[1558] FROM po_tab p ;

[1559] 尽管在大多数情况下聚合结果不会与集合元素具有相同顺序,但是这并不是得到

保证的，因此所述结果可以可以不采用文档结构。而这则有可能在未来的版本中得到调整的限制。

[1560] 集合索引 XPath 表达式还可以访问集合的某个索引。例如，“/PurchaseOrder/Item[1]/Part”将会改写，以便提取集合的第一项，然后则访问该项中的 Part 属性。

[1561] 如果集合是作为一个 VARRAY 保存的，那么这个操作将会以原始文档中给出的顺序相同的顺序来检索节点。如果将集合映射成一个嵌套表格，那么所述顺序是不确定的。如果 VARRAY 是作为一个有序集合表格 (OCT) 来保存的（对模式编译器创建的表格来说，如果设定 storeVarrayAsTable =“true”，则所述情况是默认的），那么这个集合索引访问将会优化，以便使用 VARRAY 中给出的 IOT 索引。

[1562] 不能满足的 XPath 表达式 XPath 表达式可以包含针对那些不能在输入文档中给出的节点的引用。在改写过程中，这部分表达式映射成了 SQL NULL。举例来说，XPath 表达式 :“/PurchaseOrder/ShipAddress”无法由符合 PO.xsd 这个 XML 模式的任何实例文档所满足，因为 XML 模式并未顾及 PurchaseOrder 以下的 ShipAddress 元素。因此，这个表达式将会映射成一个 SQL NULL 常量。

[1563] 命名空间的处理命名空间是使用与基于函数的评估相同的方式来处的里。对基于模式的文档来说，如果函数（类似 EXISTSNODE/EXTRACT) 没有规定任何命名空间参数，那么所述模式的目标命名空间将用作 XPath 表达式的缺省命名空间。

[1564] 实例 5-32 处理命名空间

[1565] -- 举例来说，如果 SQL 函数没有显性规定命名空间前缀和映射，

[1566] -- 那么 XPath 表达式 /PurchaseOrder/PONum 视为

[1567] -- 带有 xmlns:a =″ http://www.oracle.com/PO.xsd″ 的 /a:PurchaseOrder/a:PONum.

[1568] -- 换句话说 :

[1569] SELECT*FROM po_tab p

[1570] WHERE EXISTSNODE(value(p),’/PurchaseOrder/PONum’) = 1 ;

[1571] -- 等价于查询 :

[1572] SELECT*FROM po_tab p

[1573] WHERE EXISTSNODE(value(p),’/PurchaseOrder/PONum’,

[1574] ‘xmlns =″ http://www.oracle.com/PO.xsd’) = 1 ;

[1575] 在执行查询改写的时候，某个元素的命名空间匹配于 XML 模式定义的命名空间。如果 XML 模式包含 elementFormDefault =”qualified”，那么 XPath 表达式中的每个节点必须针对一个命名空间（这个操作可以使用缺省命名空间规定或是通过在每个节点上添加命名空间前缀来完成）。

[1576] 如果 elementFormDefault 是不合格（默认情况），那么只有定义了命名空间的节点才包含一个前缀。举例来说，如果 PO.xsd 具有将会是不合格的元素形式，那么 existsNode() 函数应该改写为 :

[1577] EXISTSNODE(value(p),’/a:PurchaseOrder/PONum’,

[1578] ‘xmlns:a =″ http://www.oracle.com/PO.xsd″) = 1

[1579] 注意 :对 elementFormDefault 是不合格的情况来说，其中在先前实例中省略了

SQL 函数 existsNode() 中的命名空间参数，这将导致每个节点默认为目标命名空间。而这也不会与 XMLSchema 定义相匹配，因此不会返回任何结果。无论所述函数是否改写，这种情况都是成立的。

[1580] 日期格式转换对 XML 模式和 SQL 来说，缺省的日期格式是不同的。因此，当改写 XPath 表达式包含了与日期的比较的时候，您需要使用 XML 格式。

[1581] 实例 50-33 日期格式转换

[1582] -- 例如，表达式

[1583] [@PurchaseDate =" 2002-02-01"]

[1584] -- 不能简单的改写成：

[1585] XMLData." PurchaseDate" =" 2002-02-01"

[1586] -- 这是因为 SQL 的缺省日期格式不是 YYYY-MM-DD。

[1587] -- 因此，在改写过程中将会添加 XML 格式串，以便将文本值适当转换成日期数据类型。

[1588] -- 因此，先前谓语将会改写成：

[1589] XMLData." PurchaseDate" = TO_DATE(" 2002-02-01" , " SYYYY-MM-DD")；

[1590] 同样，在将这些列转换成文本值（需要 extract() 等等）的时候将会添加 XML 格式串，以便将其转换成与 XML 相同的日期格式。

[1591] 用于 existsNode() 的 XPath 表达式改写

[1592] existsNode() 返回的是一个数值 0 或 1，它指示的是 XPath 是否返回任何节点 (text() 或元素节点)。基于较早章节中论述的映射，existsNode 仅仅检查标量元素是否在 XPath 针对一个 text() 机诶但或非标量节点的情况下为非空，否则以别的方式使用 SYS_XDBPD$ 来检查是否存在所述元素。如果 SYS_XDBPD$ 属性不存在，那么标量节点是否存在是通过用于标量列的空信息来确定的。

[1593] 结合了所保持的文档顺序的 existsNode 映射当保存了文档排序的时候，也就是当在模式文档中的 SYS_XDBPD$ 存在并且 maintainDOM =" true" 的时候，表格 5-12 显示了 existsNode() 的情况下的多个 XPath。

[1594] 表 5-12 用于 existsNode() 并保存了文档排序的 XPath 映射

[1595]

XPath 表达式	映射成
/PurchaseOrder	CASE WHEN XMLData IS NOT NULL THEN 1 ELSE 0 END
/PurchaseOrder/ @PurchaseDate	CASE WHEN Check_Node_Exists（XMLData,SYS_XDBPD$, 'PurchaseDate'）=1 THEN 1 ELSE 0 END
/PurchaseOrder/P ONum	CASE WHEN Check_Node_Exists（XMLData.SYS_XDBPD$,'PONum'）=1 THEN 1 ELSE 0 END
/PurchaseOrder	CASE WHEN XMLData. "PONum"=2100 THEN 1 ELSE 0

[1596]

[PONum=2100]	
/PurchaseOrder [PONum=2100]/ @PurchaseDate	CASE WHEN XMLData."PONum"=2100 AND Check_Node_Exists (XMLData.SYS_XDBPD$,'PurchaseDate') =1 THEN 1 ELSE 0 END
/PurchaseOrder/ PONum/text ( )	CASE WHEN XMLData."PONum" IS NOT NULL THEN 1 ELSE 0
/PurchaseOrder /Item	CASE WHEN EXISTS (SELECT NULL FROM TABLE (XMLData."Item") x WHERE value (x) IS NOT NULL) THEN 1 ELSE 0 END
/PurchaseOrder /Item/Part	CASE WHEN EXISTS (SELECT NULL FROM TABLE (XMLData.'Item") x WHERE Check_Node_Exists (x.SYS_XDBPD$,'Part') =1) THEN 1 ELSE 0 END
/PurchaseOrder /Item/Part/text ( )	CASE WHEN EXISTS (SELECT NULL FROM TABLE (XMLData. "Item") x WHERE x. "Part " IS NOT NULL) THEN 1 ELSE 0 END

[1597] 实例 5-34 保持了文档顺序的 existsNode 映射

[1598] --通过使用先前映射，一个对数字为 2100 的 purchaseOrder 进行检查的查询包含了一个 price 大于 2000 的部分：

```
[1599] SELECT count(*)
[1600] FROM mypos p
[1601] WHERE EXISTSNODE(value(p), ' /PurchaseOrder[PONum = 1001 and Item
[1602] /Price > 2000]' ) = 1 ;
[1603] -- 将会变成：
[1604] SELECT count(*)
[1605] FROM mypos p
[1606] WHERE CASE WHEN
[1607]          p.XMLData."PONum" = 1001AND
[1608]          EXISTS(SELECT NULL
[1609]                 FROM TABLE(XMLData." Item" )p
[1610]                 WHERE p." Price" > 2000))THEN 1 ELSE 0END = 1 ;
```

[1611] --CASE 表达式由于恒定的关系对等表达式而得到进一步优化，并且这个查询将会变成：

```
[1612] SELECT count(*)
[1613] FROM mypos p
[1614] WHERE p.XMLData." PONum" = 1001AND
[1615]       EXISTS(SELECT NULL
[1616]              FROM TABLE(p.XMLData." Item" )x
[1617]              WHERE x." Price" > 2000) ;
```

[1618] -- 如果存在 Part 和 PONum 列之上，则将会使用用于其评估的关系索引。

[1619] 没有保持文档顺序的 existsNode 映射如果 SYS_XDBPD$ 不存在（也就是说，如果 XML 模式规定 maintainDOM =" false")，那么 NULL 标量列映射成不存在的标量元

素。因此，您不必使用 SYS_XDBPD$ 属性来检查节点是否存在。表 5-13 显示了在缺少 SYS_XDBPD$ 属性时的 existsNode() 的映射。

[1620] 表 5-13 用于没有保持文档排序的 existNode 的 XPath 映射

[1621]

XPath 表达式	映射成
/PurchaseOrder	CASE WHEN XMLData IS NOT NULL THEN 1 ELSE 0 END
/PurchaseOrder/ @PurchaseDate	CASE WHEN XMLData.'PurchaseDate' IS NOT NULL THEN 1 ELSE 0 END
/PurchaseOrder /PONum	CASE WHEN XMLData."PONum" IS NOT NULL THEN 1 ELSE 0 END
/PurchaseOrder [PONum=2100]	CASE WHEN XMLData. "PONum" = 2100 THEN 1 ELSE 0 END
/PurchaseOrder [PONum=2100]/ @PurchaseOrderDate	CASE WHEN XMLData, "PONum" = 2100 AND XMLData."PurchaseDate" NOT NULL THEN 1 ELSE 0 END
/PurchaseOrder /PONum/text ( )	CASE WHEN XMLData."PONum" IS NOT NULL THEN 1 ELSE 0 END
/PurchaseOrder /Item	CASE WHEN EXISTS (SELECT NULL FROM TABLE (XMLData. "Item") x WHERE value (x) IS NOT NULL) THEN 1 ELSE 0 END
/PurchaseOrder /Item/Part	CASE WHEN EXISTS (SELECT NULL FROM TABLE (XMLData. "Item") x WHERE x."Part" IS NOT NULL) THEN 1 ELSE 0 END
/PurchaseOrder/Item /Part/text ( )	CASE WHEN EXISTS (SELECT NULL FROM TABLE (XMLData. "Item") x WHERE x."Part" IS NOT NULL) THEN 1 ELSE 0 END

[1622] 用于 ExtractValue() 的改写

[1623] extractValue() 是一个快捷方式，它通过使用 extract() 以及随后使用一个 getStringVal() 或 getNumberVal() 来获取标量内容，由此提取文本节点和属性。extractValue 返回的是标量元素的文本节点或是属性节点的值。extractValue() 不能对赶回多个值或非标量元素进行处理。

[1624] 表 5-14 显示了 extractValue() 情况下的多种 XPath 表达式的映射。如果 XPath 表达式针对的是一个元素，那么 extractValue 检索所述元素的文本节点的子节点。这样一来，这两个 XPath 表达式 /PurchaseOrder/PONum 和 /PurchaseOrder/PONum/text() 是通过 extractValue 而被同等处理的，并且它们两个都会检索 PONum 的标量内容。

[1625] 表 5-14 用于 extractValue() 的 XPath 映射

[1626]

XPath 表达式	映射成
/PurchaseOrder	不支持，ExtractValue 只能检索用于标量元素和属性的值。
/PurchaseOrder/ @PurchaseDate	XMLData. "PurchaseDate"
/PurchaseOrder/PONum	XMLData. "PONum"
/PurchaseOrder [PONum=2100]	（SELECT TO_XML（x.XMLData）FROM Dual WHERE x. "PONum"=2100）
/PurchaseOrder [PONum=2100]/ @PurchaseDate	（SELECT x.XMLData."PurchaseDate"） FROM Dual WHERE x. "PONum"=2100
/PurchaseOrder /PONum/text（）	XMLData."PONum"
/PurchaseOrder /Item	不支持，ExtractValue 只能检索用于标量元素和属性的值。
/PurchaseOrder /Item/Part	不支持，ExtractValue 不能检索多个标量值
/PurchaseOrder /Item/Part/text（）	不支持，ExtractValue 不能检索多个标量值

[1627] 实例 5-35 改写 extractValue()

[1628] --举例来说，诸如

[1629] SELECT ExtractValue(value(p), ' /PurchaseOrder/PONum')

[1630] FROM mypos p

[1631] WHERE ExtractValue(value(p), ' /PurchaseOrder/PONum') = 1001 ;

[1632] --这样的 SQL 查询将会变成：

[1633] SELECT p.XMLData." PONum"

[1634] FROM mypos p

[1635] WHERE p.XMLData."PONum" = 1001 ;

[1636] 由于将查询改写成了简单的标量列，因此如果存在关于 PONum 属性的索引的话，那么所述索引可以用于满足所述查询。

[1637] 创建索引 ExtractValue 可以在索引表达式中使用。如果将所述表达式改写成标量列，则将所述索引转换成一个 BTree 索引而不是基于函数的索引。

[1638] 实例 5-36 创建索引

[1639] --举例来说：

[1640] create index my_po_index on mypos x

[1641] (Extract(value(x), ' /PurchaseOrder/PONum/text() '). getnumberval()) ;

[1642] --将会得到：

[1643] create index my_po_index on mypos x(x.XMLData." PONum") ;

[1644] --由此将会变成一个正规的 BTree 索引。这种操作非常有用，因为与函数索引所不同，相同的索引现在可以满足针对列的查询，例如：

[1645] EXISTSNODE(value(x), ' /PurchaseOrder[PONum = 1001]') = 1 ;

[1646] --由此将会变成一个正规的 B*Tree 索引。这种操作非常有用，因为与函数索引所不同，相同的索引现在可以满足针对列的查询，例如：

[1647] EXISTSNODE(value(x), ' /PurchaseOrder[PONum = 1001]') = 1 ;

[1648] 用于 extract() 的改写

[1649] extract() 检索的是作为 XML 的 XPath 的结构。用于 extract() 的改写与用于那些包含了文本节点的 XPath 表达式的 extractValue() 的改写相似。

[1650] 保持了文档排序的提取映射表 5-15 显示了在保留文档顺序时（也就是当模式文档中存在 SYS_XDBPD$ 并且 maintainDOM =" true" 的时候）进行的 extract() 情况下的多种 XPath 的映射

[1651] 注意 :这个实例显示了带有空别名串" " 的 XMLElement 和 XMLForest,以便指示您可以创建一个只带有文本值的 XML 实例。这个实例仅仅是作为示范而被显示的。

[1652] 表 5-15 保留了文档排序的 extract() 的 XPath 映射

[1653]

XPath	**映射成**
/PurchaseOrder	XMLForest（XMLData as "PurchaseOrder"）
/PurchaseOrder/ @PurchaseDate	CASE WHEN Check_Node_Exists（XMLData.SYS.XDBPD$, 'PurchaseDate'）=1 THEN XML Element（"",XMLData. "PurchaseDate"）ELSE NULL END
/PurchaseOrder /PONum	CASE WHEN Check_Node_Exists（XMLData.SYS_XDBPD$,'PONum'）=1 THEN XML Element（"PONum",XMLData. "PONum"）ELSE NULL END
/PurchaseOrder [PONum=2100]	（SELECT XMLForest（XMLData as "PurchaseOrder"） FROM Dual WHERE x. "PONum"=2100）
/PurchaseOrder [PONum=2100]/ @PurchaseDate	（SELECT CASE WHEN Check_Node_Exists（x.XMLData.SYS.XDBPD$;"PurchaseDate"）=1 THEN XMLElement（“”,XMLData."PurchaseDate”） ELSE NULL END FROM Dual WHERE x."PONum"=2100）
/PurchaseOrder/ PONum/text（）	XMLElement（"",XMLData.PONum）
/PurchaseOrder /Item	（SELECT XMLAgg（XMLForest（value（p）as "Item"））FROM TABLE（x.XMLData."Item"）p WHERE value（p）IS NOT NULL）
/PurchaseOrder	（SELECT XMLAgg（

[1654]

/Item/Part	CASE WHEN Check_Node_Exists (p.SYS_XDBPD$;"Part") =1 THEN XMLForest (p. "Part" as "Part") ELSE NULL END) FROM TABLE (x.XMLData."Item") p)
/PurchaseOrder /Item/Part/text ( )	(SELECT XMLAgg (XMLElement ("",p. "Part")) FROM TABLE (x.XMLData. "Item") x)

[1655] 实例 5-37 保留文档排序的用于 extract() 的 XPath 映射

[1656] --通过使用表格 5-15 中的映射,一个提取那些purchaseorder包含了 price 大于 2000 的 Part 的 PONum 元素的查询 :

```
[1657] SELECT Extract(value(p), ' /PurchaseOrder[Item/Part > 2000]/PONum' )
[1658] FROM po_tab p ;
[1659] -- 将会变成 :
[1660] SELECT(SELECT CASE WHEN Check_Node_Exists(p.XMLData.SYS_XDBPD$,
[1661] ' PONum' ) = 1
[1662]                     THEN XMLElement(" PONum" ,p.XMLData." PONum" )
[1663]                     ELSENULLEND)
[1664]       FROM DUAL
[1665]       WHERE EXI STS(S ELECT NULL
[1666]                       FROM TABLE(XMLData." Item" )p
[1667]                       WHERE p." Part" > 2000)
[1668]       )
[1669] FROM po_tab p ;
[1670] --Check_Node_Exists 是一个仅仅用于说明目的的内部函数。
```

[1671] 在不保持文档排序的情况下提取映射如果 SYS_XDBPD$ 不存在,也就是说,如果 XML 模式规定 maintainDOM ="false",那么 NULL 标量列映射成不存在的标量元素。因此,您不必使用 SYS_XDBPD$ 属性来检查节点是否存在。表 5-16 显示了在缺少 SYS_XDBPD$ 属性时的 existsNode() 的映射。

[1672] 表 5-16 在没有保持文档排序的情况下的用于 extract() 的 XPath 映射

[1673]

XPath	等价于
/PurchaseOrder	XMLForest (XMLData AS "PurchaseOrder")
/PurchaseOrder/ @PurchaseDate	XMLForest (XMLData. "PurchaseDate" AS "")
/PurchaseOrder /PONum	XMLForest (XMLData. "PONum"AS"PONum")
/PurchaseOrder (PONum=2100]	(SELECT XMLForest (XMLData AS "PurchaseOrder") FROM Dual WHERE x. "PONum"=2100)
/PurchaseOrder [PONum=2100]/ @PurchaseDate	(SELECT XMLForest (XMLData. "PurchaseDate" AS"") FROM Dual WHERE x."PONum"=2100)
/PurchaseOrder/ PONum/text ( )	XMLForest (XMLData.PONumAS"")
/PurchaseOrder /Item	(SELECT XMLAgg (XMLForest (value (p) as "Item") FROM TABLE (x,XMLData. "Item") p WHERE value (p) IS NOT NULL)
/PurchaseOrder /Item/Part	(SELECT XMLAgg (XMLForest (p, "Part" AS "Part") FROM TABLE (x.XMLData. "Item") p)
/PurchaseOrder /Item/Part/text ( )	(SELECT XMLAgg (XMLForest (p."Part" AS "Part")) FROM TABLE (x.XMLData."Item") p)

[1674] 使用了 updateXML() 的优化更新

[1675] 使用 updateXML() 所进行的常规更新包括更新 XML 文档的值，然后使用新更新的文档来替换整个文档。

[1676] 当以对象关系形式保存 XMLType 时，通过使用 XML 模式映射，将会对所述更新进行优化，以便直接更新所述文档。举例来说，可以对更新 PONum 元素值可以进行改写，以便直接更新 XMLData.PONum 列而不是在存储器种实现整个文档以及随后执行更新。

[1677] 在将 updateXML() 用于更新的时候，它必须满足以下条件：

[1678] ●提供给 updateXML() 的 XMLType 列必须与 SET 从句中更新的是相同的列，例如：

[1679] UPDATE po_tab p SET value(p) = updatexml(value(p),...) ;

[1680] ●必须以对象关系方式并使用 Oracle XML DB 的 XML 模式映射来保存 XMLType

[1681] ● XPath 表达式不能包含任何谓语或集合遍历

[1682] ●不能有重复的标量表达式

[1683] ● updateXML() 函数中的所有 XPath 自变量都必须只针对标量内容，也就是文本节点或属性 -- 例如：

[1684] UPDATE po_tab p SET value(p) =

[1685] updatexml(value(p),'/PurchaseOrder/@PurchaseDate','2001-01-02',

[1686] /PurchaseOrder/PONum/text()',2200) ;

[1687] 如果满足全部处理条件，那么更新过的 XML 被重新写进简单关系更新。例如：

[1688] UPDATE po_tab p SET value(p) =

[1689] updatexml(value(p),'/PurchaseOrder/@PurchaseDate','2001-01-02',

[1690] '/PurchaseOrder/PONum/text()',2200) ;

[1691] 将会变成

[1692] UPDATE po_tab p

[1693] SET p.XMLData."PurchaseDate"= TO_DATE('2002-01-02','SYYYY-MM-DD')

[1694] p.XMData."PONum" = 2100 ;

[1695] 日期转换诸如 Date、gMonth、gDate 等等的日期数据类型在 XMLSchema 与 SQL 系统中具有不同的格式。在这种情况下,如果 updateXML 具有一个用于这些列的串值,那么改写将会自动放入 XML 格式串,以便正确转换所述串值。这样一来,为日期列规定的串值必须与 XML 日期格式相匹配而不是与 SQL 日期格式匹配。

[1696] 在 XML 模式注册中创建缺省表格

[1697] 作为模式注册的一部分,您也可以创建缺省表格。在借助于不具有任何 FTP、HTTP 之类的表格规范的 API 插入符合这种模式的 XML 实例文档的时候,缺省表格是非常有用的。在这种情况下,XML 实例插入缺省表格中。

[1698] 如果您为 defaultTable 属性给出一个值,则使用所述名称来创建 XMLType 表格。否则使用内部产生的名称来创建所述表格。

[1699] 此外,使用 tableProps 和 columnProps 属性规定的任何文本都附加于所生成的 CREATE TABLE 语句。

[1700] 表格中的有序集合 (OCTs)

[1701] XML 模式中的数组 (maxOccurs > 1 的元素) 通常保存在 VARRAY 中,它可以保存在一个大型对象 (LOB) 中,也可以保存在一个类似嵌套表格的独立存储表中。

[1702] 注意 :在将 VARRAY 的元素保存在独立的表格中的时候,所述 VARRAY 指的是表格中的有序集合 (OCT)。在以下段落中,对于 OCT 的引用也假设您正将索引组织表格 (IOT) 存储用于"存储"表。

[1703] 这样则允许 VARRAY 的元素驻留在基于 IOT 的单独表格中。所述表格的主键码是 (NESTED_TABLE_ID、ARRAY_INDEX)。NESTED_TABLE_ID 被用于将元素与其包含双亲相结合,而 ARRAY_INDEX 列则记录集合中元素的位置。

[1704] 将 OCT 用于 VARRAY 存储

[1705] 目前存在两种规定一个 OCT 存储的方式 :

[1706] ●借助于模式属性"storeVarrayAsTable"。作为默认情况,这个属性为"false",并且 VARRAYs 保存在一个 LOB 中。

[1707] 如果将其设定为"true",则所有 VARRAYs 即 maxOccurs > 1 的所有元素都是作为 OCTs 而被保存的。

[1708] ●通过使用"tableProps"属性来显性规定所述存储。创建 OCT 所需要的朱雀的 SQL 可以用作 tableProps 属性的一部分 :

[1709] " VARRAYxmldata.<array>STORE AS TABLE<myTable>((PRIMARY KEY

[1710] (NESTED_TABLE_ID,ARRAY_INDEX))ORGANIZATION INDEX)"

[1711] 将 OCTs 用于 VARRAY 存储的优点包括更快的访问元素以及更好的查询能力。索引可以基于元素属性创建并且这些有助于更好的执行查询改写。

[1712] XML 模式之间的循环引用

[1713] XML 模式文档可以具有循环相关性,它可以防止文档以常规方式相继注册。这种 XML 模式的实例如下所示 :

[1714] 实例 5-38 循环相关性

[1715] -- 包含了另一个模式的模式在所包含模式不存在的情况下是不能创建的。

[1716] begin dbms_xmlschema.registerSchema(' xm40.xsd' ,

[1717] ' <schema xmlns = " http://www.w3.org/2001/XMLSchema " xmlns:my =" xm40"

[1718] targetNamespace =" xm40" >

[1719] <include schemaLocation =" xm40a.xsd" />

[1720] <! -- 在这里定义一个全局复杂类型 -->

[1721] <complexType name =" Company" >

[1722] <sequence>

[1723] <element name =" Name" type =" string" />

[1724] <element name =" Address" type =" string" />

[1725] </sequence>

[1726] </complexType>

[1727] <! - 定义一个依赖于所包含的模式的全局元素 -->

[1728] <element name =" Emp" type =" my:Employee" />

[1729] </schema>',

[1730] true,true,false,true) ;end ;

[1731] /

[1732] -- 然而可以使用 FORCE 选项来创建所述模式

[1733] begin dbms_xmlschema.registerSchema(' xm40.xsd' ,

[1734] ' <schema xmlns = " http://www.w3.org/2001/XMLSchema " xmlns:my =" xm40"

[1735] targetNamespace =" xm40" >

[1736] <include schemaLocation =" xm40a.xsd" />

[1737] <! - 在这里定义一个全局复杂类型 -->

[1738] <complexType name =" Company" >

[1739] <sequence>

[1740] <element name =" Name" type =" string" />

[1741] <element name =" Address" type =" string" />

[1742] </sequence>

[1743] </complexType>

[1744] <! - 定义一个依赖于所包含的模式的全局元素 -->

[1745] <element name =" Emp" type =" my:Employee" />

[1746] </schema>' ,

[1747] true,true,false,true,true) ;end ;

[1748] /

[1749] -- 尝试使用这个模式将会试图重新贬义并且将会失败

[1750] create table foo of sys.xmltype xmlschema" xm40.xsd" element" Emp" ;

[1751] -- 现在使用 FORCE 选项来创建第二个模式,这还应该会使得第一模式有效。

[1752] begin dbms_xmlschema.registerSchema(' xm40a.xsd' ,

[1753] ' <schema xmlns = " http://www.w3.org/2001/XMLSchema " xmlns:my =" xm40"

[1754] targetNamespace =" xm40" >

[1755] <include schemaLocation =" xm40.xsd" />

[1756] <! -- 在这里定义一个全局复杂类型 -->

[1757] <complexType name =" Employee" >

[1758] <sequence>

[1759] <element name =" Name" type =" string" />

[1760] <element name =" Age" type =" positiveInteger" />

[1761] <element name =" Phone" type =”string" />

[1762] </sequence>

[1763] </complexType>

[1764] <! -定义一个依赖于所包含的模式的全局元素 -->

[1765] <element name =" Comp”type =" my:Company" />

[1766] </schema>' ,

[1767] true,true,false,true,true) ;end ;

[1768] /

[1769] -- 二者都可用于创建表格等等

[1770] create table foo of sys.xml type xmlschema" xm40.xsd" element" Emp" ;

[1771] create table foo2 of sys.xml type xmlschema " xm40a.xsd" element" Comp" ;

[1772] 为了注册这些彼此具有循环相关性的 XML 模式,您必须如下所示将 FORCE 参数用在 DBMS_XMLSCHEMA.registerSchema 中 :

[1773] 1. 步骤 1 :以“FORCE”方式来注册“s1/xsd”:

[1774] dbms_xmlschema.registerSchema(" s1.xs d" , " <schema..." ....,force = >true)

[1775] 这时,s1.xsd 是无效的并且不能使用

[1776] 2. 步骤 2 :以“FORCE”方式来注册“s2.xsd”

[1777] dbms_xmlschema.registerSchema(" s2.xsd" , " <schema..." ,...,force = >true)

[1778] 第二个操作自动编译 s1.xsd 并且使这两个模式有效。

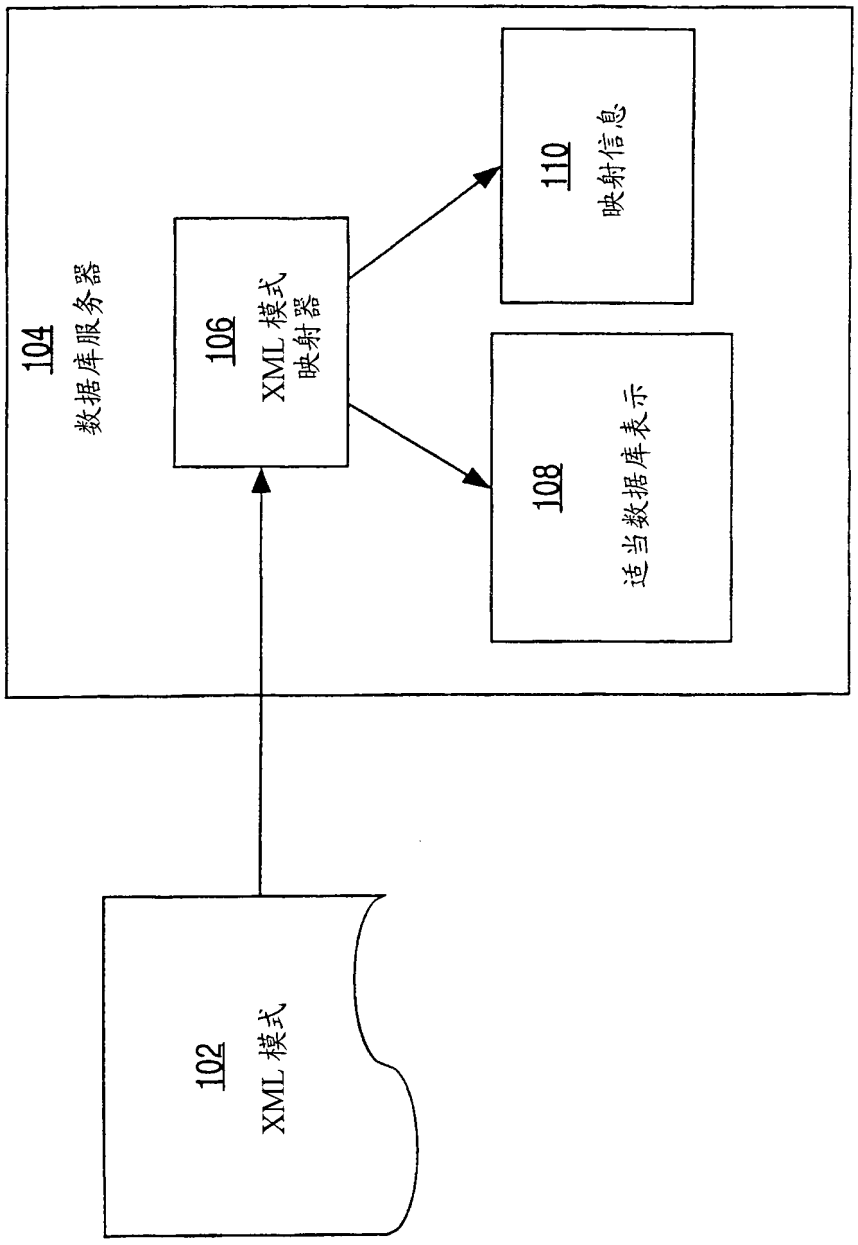


图 1

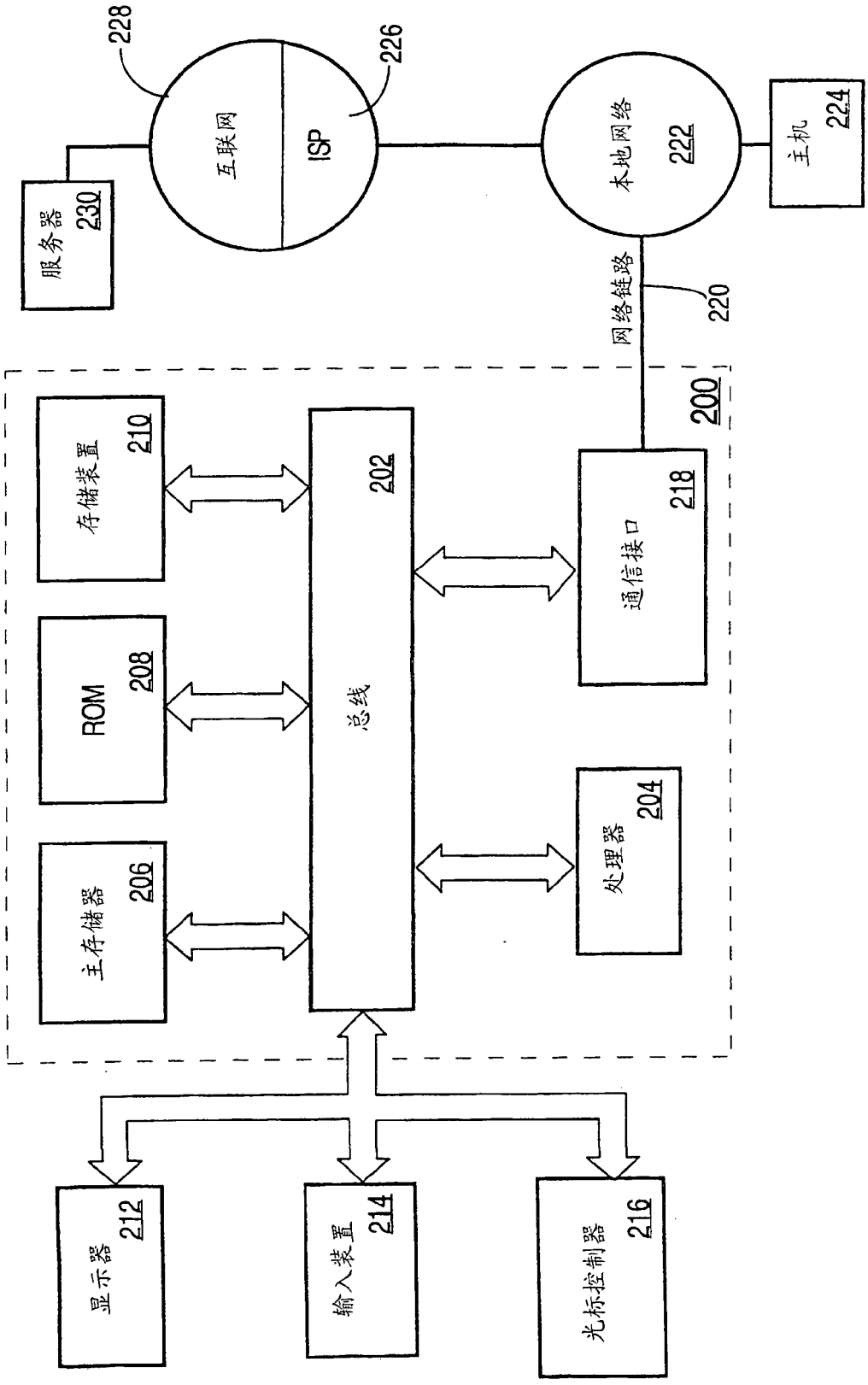

228
互联网
ISP
226
本地网络
222
主机
224
服务器
230
网络链路
220
200
存储装置
210
202
通信接口
218
ROM
208
总线
主存储器
206
处理器
204
显示器
212
输入装置
214
光标控制器
216

图 2

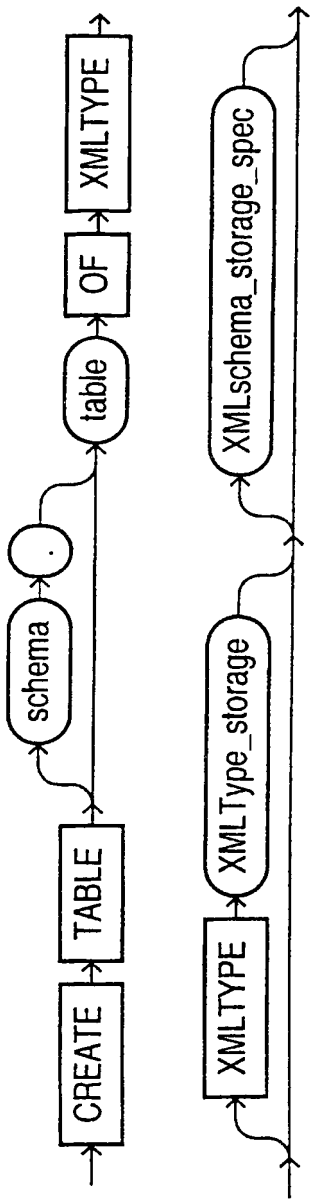

CREATE
TABLE
schema
.
table
OF
XMLTYPE
XMLTYPE
XMLType_storage
XMLschema_storage_spec

图 3

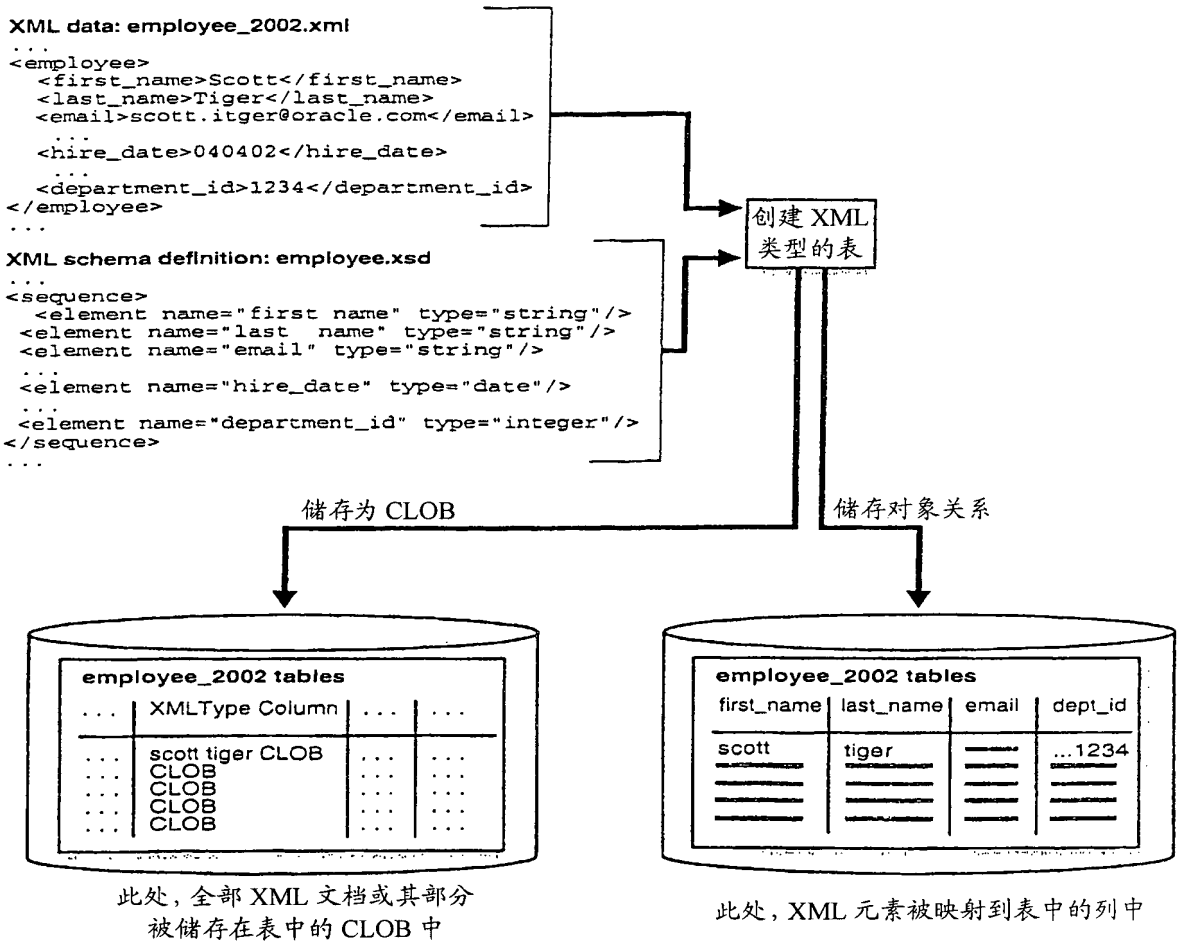

XML data: employee_2002.xml
...
<employee>
<first_name>Scott</first_name>
<last_name>Tiger</last_name>
<email>scott.itger@oracle.com</email>
...
<hire_date>040402</hire_date>
...
<department_id>1234</department_id>
</employee>
...
XML schema definition: employee.xsd
...
<sequence>
<element name="first name" type="string"/>
<element name="last name" type="string"/>
<element name="email" type="string"/>
...
<element name="hire_date" type="date"/>
...
<element name="department_id" type="integer"/>
</sequence>
...
创建 XML 类型的表
储存为 CLOB
储存对象关系
employee_2002 tables
... XMLType Column ... ...
... scott tiger CLOB ... ...
... CLOB ... ...
... CLOB ... ...
... CLOB ... ...
... CLOB ... ...
employee_2002 tables
first_name last_name email dept_id
scott tiger ...1234
此处，全部 XML 文档或其部分被储存在表中的 CLOB 中
此处，XML 元素被映射到表中的列中

图 4

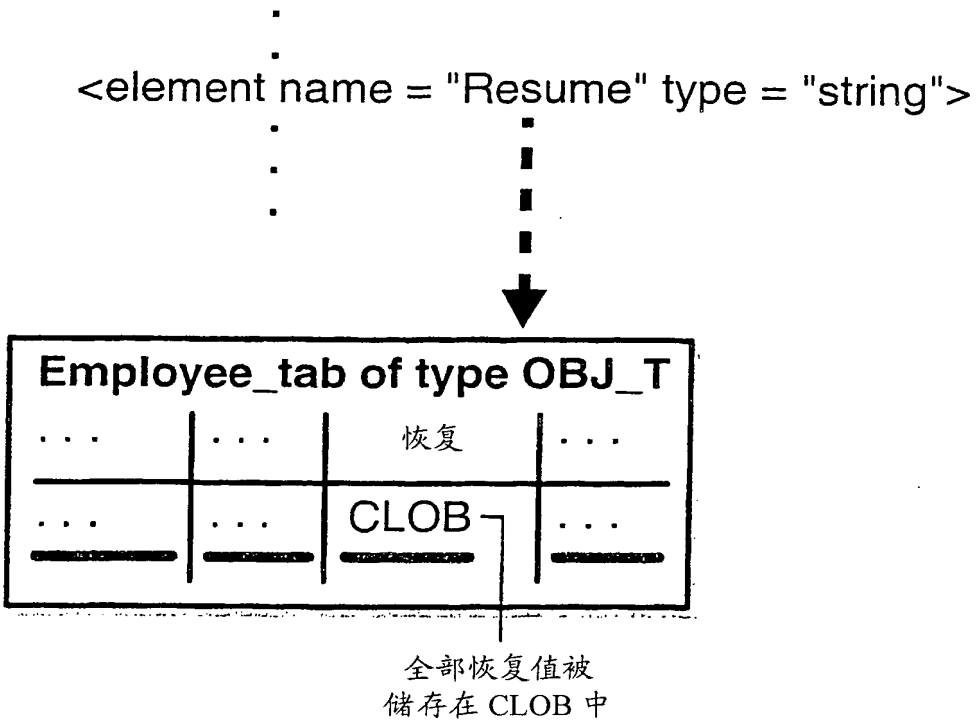

<element name = "Resume" type = "string">
Employee_tab of type OBJ_T
恢复
CLOB
全部恢复值被
储存在 CLOB 中

图 5

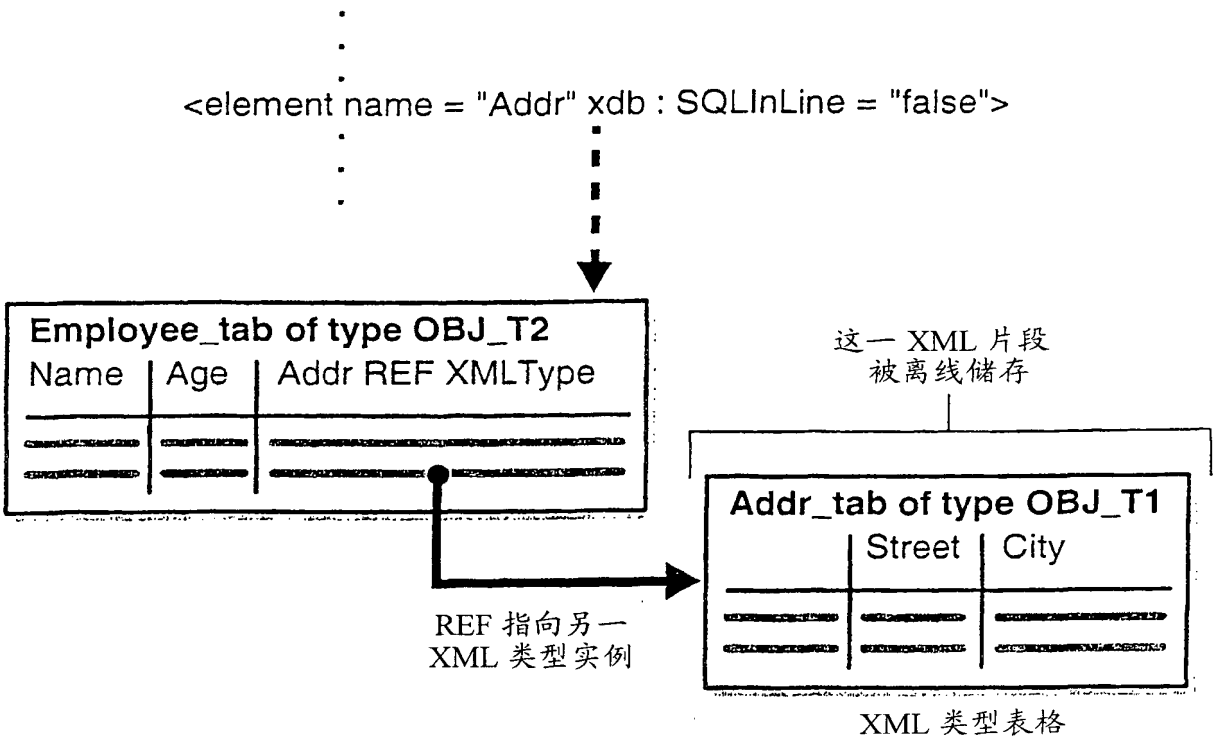

<element name = "Addr" xdb : SQLInLine = "false">
Employee_tab of type OBJ_T2
Name
Age
Addr REF XMLType
这一 XML 片段
被离线储存
Addr_tab of type OBJ_T1
Street
City
REF 指向另一
XML 类型实例
XML 类型表格

图 6

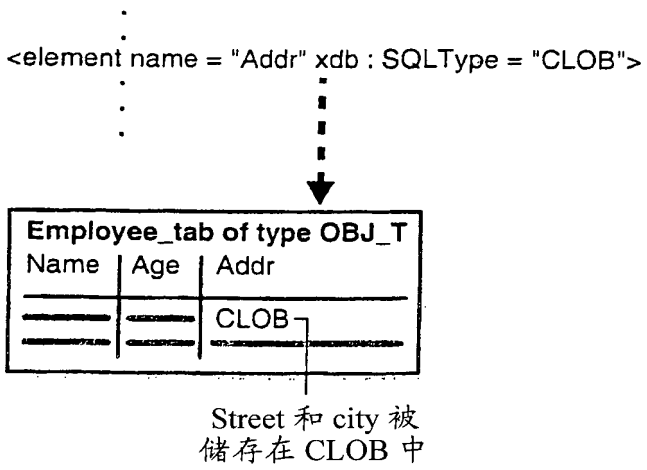

<element name = "Addr" xdb : SQLType = "CLOB">
Employee_tab of type OBJ_T
Name
Age
Addr
CLOB
Street 和 city 被
储存在 CLOB 中

图 7

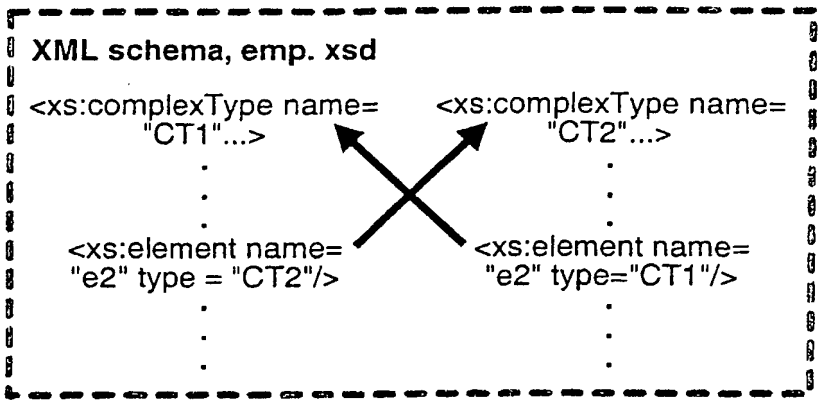

XML schema, emp. xsd
<xs:complexType name= "CT1"...>
<xs:complexType name= "CT2"...>
<xs:element name= "e2" type = "CT2"/>
<xs:element name= "e2" type="CT1"/>

图 8

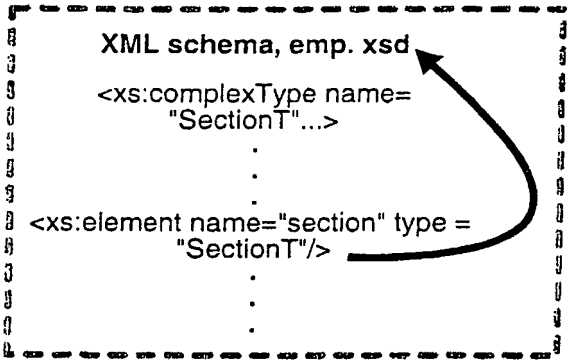

XML schema, emp. xsd
<xs:complexType name= "SectionT"...>
<xs:element name="section" type = "SectionT"/>

图 9

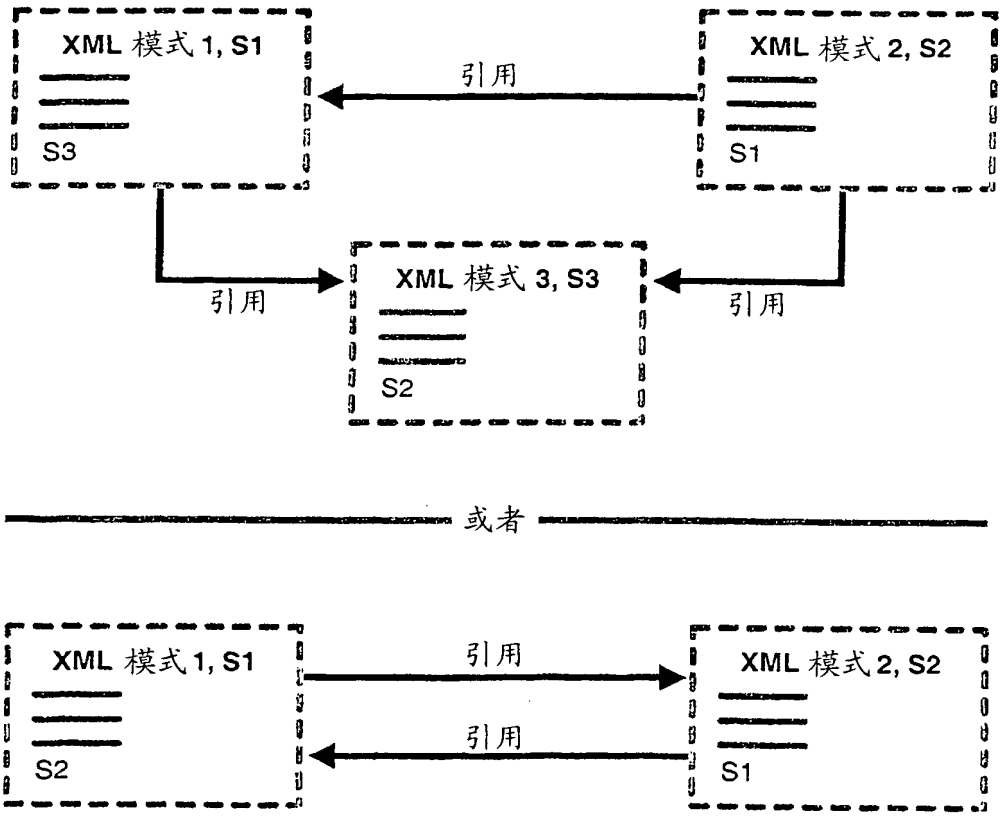

XML 模式 1, S1
S3
引用
XML 模式 2, S2
S1
引用
引用
XML 模式 3, S3
S2
或者
XML 模式 1, S1
S2
引用
引用
XML 模式 2, S2
S1

图 10