第 34 卷 第 11 期 计 算 机 工 程 2008 年 6 月
Vol.34 No.11 Computer Engineering June 2008
·软件技术与数据库· 文章编号：1000—3428(2008)11—0074—03 文献标识码：A 中图分类号：TP311

Unicode 数据仓库 ETL 的设计与实现

许 威，李茂青
(厦门大学自动化系，厦门 361005)

摘 要：在 Unicode 数据装载过程中，如源字符集中的某个字符在目标字符集中没有定义，将会出现错误，产生信息丢失的现象。针对这种情况，该文提出一种从源 Oracle 数据库到目标 Teradata 数据仓库字符集转换的 ETL 设计方法和实现。实践表明该方案有效可行，能提高 ETL 过程的容错率。

关键词：字符集；数据仓库；统一字符编码标准

Design and Implementation of Unicode Data Warehouse ETL

XU Wei, LI Mao-qing
(Dept. of Automation, Xiamen University, Xiamen 361005)

【Abstract】During the process of loading Unicode data, the phenomenon so-called a loss of information will be encountered while the source characters are not defined in the target character sets. In light of this situation, this paper delivers a fault tolerant ETL solution on how to resolve the source/target character set conversion problem while populating Unicode data from Oracle 9i database to Teradata data warehouse. Practical result proves its efficiency and accuracy.

【Key words】character set; data warehouse; Unicode

1 概述

对语言字符集处理方案的探索[1]是建立包含多国语言字符集数据仓库技术研究的关键和难点，它主要存在以下问题：如果源数据库、目标数据库的字符集设置不一致，那么 ETL 过程中传递的数据将在多套编码方案间自动切换，对于在源字符集中存在而在目标字符集中不存在的符号，理论上转换将会产生信息丢失。

Unicode 被定义为支持所有主要语言的统一字符编码标准，该标准同时也被数据库产品 Oracle 和 Teradata 所采用。目前 Oracle 9i 版本使用 Unicode3.1[2]而 Teradata V2R5[3]版本支持 2.1 标准。目前版本的 Teradata 不能够完全支持 3.1 标准所定义的全部字符。

2 数据转换

本文针对数据仓库项目开发实践过程中碰到的字符集不兼容问题，提出在采用不同 Unicode 标准存储信息符号的源和目标库之间实现数据转换的解决办法。问题描述见图 1。

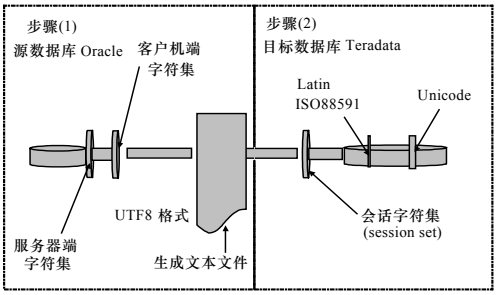

图 1 Oracle 至 Teradata 的数据流转换

源数据库为 Oracle9i 数据库，支持 Unicode3.1 标准，采用 UTF8 字符集存储英语、简体中文、繁体中文等数据。目标数据仓库为 Teradata V2R5 数据库，支持 2.1 标准，采用 UTF16 字符集格式存储数据。数据 ETL 过程分解为 2 步：(1)从源 Oracle 中导出 UTF8 格式的中间文本文件；(2)将文本文件通过 Teradata 的客户端载入目标数据仓库。Teradata 客户端的会话设置(Session Set)等于 UTF8。

在步骤(1)中，只须将 Oracle 数据卸载的客户端参数 NLS_LANG 配置为和源一致，在生成 UTF8 格式的文本文件就不会碰到问题。源客户端参数设置语句为 EXPORT NLS_LANG=AMERICAN_AMERICA.AL32UTF8。但是在步骤(2)中，因为目标数据仓库内部支持的字符要少于源字符集中的字符，所以 Teradata 在内部字符映射的过程中将会出现错误，产生信息丢失的现象。系统返回错误信息：6705 An illegally formed character string was encountered during translation.

3 术语定义和 ETL 字符集兼容性定理

UTF(UCS Transformation Format)[2]是 Unicode 的一种储存和传送的格式。UTF 规范定义了传输和存储 UCS 编码的具体规定。常见的 UTF 规范包括 UTF8(以 8 位为单元对 UCS 进行编码，使用变长字节表示一个字母或汉字)、UTF-16(双字节表示一个字母或汉字)、UTF-32(4 个字节表示一个字母或汉字)。

定义 1 超集(Superset) 假设 $A\{a_1, a_2,\cdots, a_i,\cdots, a_{m-1}, a_m\}$，$B\{b_1, b_2,\cdots, b_j,\cdots, b_{n-1}, b_n\}$分别代表不同的字符集，其中，$a_i$, b_j 分别代表字符集 A, B 中所包含的具体字符。如果对于 B 中的每个字符 b_j 都可在 A 中找到字符 a_i 和其对应，即字符集 A 包含字符集 B 定义编码的所有字符，就将 A 称做字符集 B 的超

基金项目：国家“985”工程二期基金资助项目(0000-X07204)；福建省自然科学基金资助项目(E0520001)
作者简介：许 威(1975－)，男，博士研究生，主研方向：数据库，数据仓库；李茂青，教授、博士生导师
收稿日期：2007-07-08 **E-mail：**wei.xu.xmu@126.com

集，记为 $A \supseteq B$。

上述定义可以表示为：Iff $\forall\ b_j(b_j \in B), \exists\ a_i(a_i \in A) \Leftrightarrow A \supseteq B$。如果超集 A 中还包含不属于 B 的附加字符，就称 A 为 B 的全超集，记做 $A \supset B$。例如，ISO88591 字符集是 ASCII 字符集的超集，因为它包含 US-ASCII 字符集的所有字符。根据定义 1 可以得到：

定理 1 一个字符集可以认为是它自己和其他所有兼容字符集的超集。

证明：反证法证明。若设字符集 A 不是其自身的超集，根据定义 1，可以在 A 中找到某个字符 a_i，使 $a_i \notin A$，这实际上和 a_i 代表 A 中某个字符发生矛盾。由此得证。

定义 2 子集(Subset) 假设 $A\{a_1, a_2,\cdots, a_i,\cdots, a_{m-1}, a_m\}$，$B\{b_1, b_2,\cdots, b_j,\cdots, b_{n-1}, b_n\}$分别代表不同的字符集，其中，$a_i, b_j$ 分别代表字符集 A, B 中所包含的具体字符。如果对于 A 中的每个字符 a_i，都可在 B 中找到字符 b_j 和其对应，即字符集 A 中的所有字符同时也在另一个字符集 B 中有编码，就将 A 称做字符集 B 的子集，记为 $A \subseteq B$。

上述定义可以表示为：Iff $\forall\ a_i(a_i \in A), \exists\ b_j(b_j \in B) \Leftrightarrow A \subseteq B$。如果超集 B 中还包含不属于 A 的附加字符，就称 A 为 B 的全子集，记做 $A \subset B$。例如，US-ASCII 字符集是 ISO88591 字符集的子集，因 US-ASCII 字符集中的所有字符同时也在 ISO88591 字符集中有编码。同理可以得到：

定理 2 一个字符集可以认为是它自己和其他所有兼容字符集的子集。

基于上述定义，可以得到在实际的数据移动过程中的源、目标字符集兼容性定理，如下所示：

定理 3 假设 A 为源数据库字符集，B 定义为目标数据库字符集，如果要正确移动数据，那么 B 必须是 A 的超集或者 $A=B$(字符集设置相同)。即 $B \supseteq A$。

根据定理 3，源字符集必须是目标字符集的子集，这就是本文引用的例子会出现 6705 错误的原因。

4 解决方案

造成数据装载错误的主要原因在于目标数据仓库内部会自动对输入字符集进行转换，当发现某个字符在其内部不存在映射时，就会返回错误信息 6705。对于这个问题，本文的解决办法是将图 1 中的步骤(2)再分解为以下几个子过程：

(1)在目标库 Teradata 中创建临时表，临时表的结构除可以被定义为 UNICODE 类型的字段外都和目标表一致，临时表中该部分对应字段被定义为 LATIN/ISO88591(西欧字符)类型，这样就保证了在进行图 1 中步骤(2)的数据装载时，Teradata 不会自动对原有定义为 UNICODE 字段部分进行转换，而是将输入的 UTF8 字节流保持不便并以 Latin/ ISO88591 的格式存放在中间表中。中间表和目标表差别见表 1、表 2，例如，字段 SHPMT_MTHD 的字段名称相同，但数据类型不同。

表 1 中间表 STG_ SHPMT_MTHD_W 的数据结构

字段名称	字段数据类型
SHPMT_MTHD_ID	DECIMAL(9,0)
SHPMT_MTHD	VARCHAR(100) CHARACTER SET LATIN

表 2 目标表 DW_SHPMT_MTHD 的数据结构

字段名称	字段数据类型
SHPMT_MTHD_ID	DECIMAL(9,0)
SHPMT_MTHD	VARCHAR(100) CHARACTER SET UNICODE

(2)将 UTF8 格式的文本文件载入过程(1)中定义的中间表。此时 Teradata 的会话设置(Session Set)等于 ASCII，以此保证载入过程不产生任何字符转换操作。

(3)将中间表记录插入目标表，处理语句中包括自定义 UNICODE 转换函数 UTF8Decode2BytesUnicode，其目的是实现从 UTF8 到 UNICODE 格式字符集的转换。采用自定义函数转换的方法可以避开 Teradata 内部自身的字符转换和报错机制，从而增强数据处理的容错性。中间表内容插入目标表语句的范例如下所示。

```
INSERT INTO DW_SHPMT_MTHD (
  SHPMT_MTHD_ID,
  SHPMT_MTHD
)
SELECT SHPMT_MTHD_ID,
SYSLIB. UTF8Decode2BytesUnicode (A.SHPMT_MTHD) AS SHPMT_MTHD
FROM STG_HPMT_MTHD_W STG
LEFT JOIN DW_SHPMT_MTHD DW
ON STG.SHPMT_MTHD_ID = DW.SHPMT_MTHD_ID
WHERE STG.SHPMT_MTHD_ID IS NULL
```

5 UTF8 转换为 Unicode 的算法与分析

5.1 UTF8 编码规则描述

在 UTF8 编码中，所有的字符均使用 1 ~6 个字节进行编码。由于实际上在转换为 UTF-8 编码时最多用到 3 个字节，因此下文给出的算法和代码中只处理 UTF-8 编码长度不大于 3 个字节的情况。通常来说，在只包含 1 个字节的 UTF8 编码中，其最高位置 0，其余的 7 个二进制位用来对字符进行编码；在含 n ($1<n<6$)个字节的 UTF8 编码中，其第 1 个字节的前 n 位置 1，第 n+1 位置 0，该字节剩余的其他二进制位用来对字符进行编码，后续字节的最高位均置 1，次高位均置 0，每个字节中剩余的 6 个二进制用来对字符进行编码。一个字符的 UTF8 编码中包含字节个数的多少是由其 Unicode 编码所处的范围来决定的，具体见表 3，其中给出了不同范围内的 Unicode 字符对应的 UTF8 编码的格式，已填入的 1、0 数据表示约定的标记位，x 表示对字符编码时可用的二进制位。将 Unicode 编码的二进制位按由低到高的次序放入 x 表示的空位中即可得到其所对应的 UTF8 编码。

表 3 Unicode 与 UTF8 的编码格式对照表

Unicode 编码(十六进制)	UTF8 编码(二进制)
U+0000 0000 - U+0000 007F	0xxxxxxx
U+0000 0080 - U+0000 07FF	110xxxxx 10xxxxxx
U+0000 0800 - U+0000 FFFF	1110xxxx 10xxxxxx 10xxxxxx

5.2 UTF8 转换为 Unicode 的算法描述

UTF8Decode2BytesUnicode 算法描述如下：

输入 以 UTF8 格式编码的字节流。定义 β 为输入的字节流，$\beta=\{\beta_1, \beta_2,\cdots, \beta_i,\cdots, \beta_n, n\leqslant 6\}$，本文考虑的情况 $n\leqslant 3$。其中，单字节 β_i 可以分解为标志位(每个字节的前导字符位，格式为 0,110,1110,11110 等，见表 1 中第 2 列)和有效位(每个字节中剔除前导字符位外的其余字位，表 3 列 2 中使用 x 表示)，记做 $\beta_i=\sigma_i+\delta_i$，$\sigma_i$ 代表标志位，δ_i 代表有效位。

输出 字符的 Unicode 形式编码。

常量定义如下：

MASKBYTE = 80 /*二进制表现格式为 01110000，用于判定字符是否由单字节组成*/

MASK2BYTES = xC0 /*二进制表现格式为 11000000 用于判定字符是否由双字节组成*/

MASK3BYTES = xE0 /*二进制表现格式为 11100000 用于判定字符是否由 3 字节组成*/

(1)Scan β /*根据 UTF8 编码的大小确定该编码由几个字节组成，通过和掩码常量 MASKλ(λ=1,2,3)TYPE 进行位的与运算得到需要处理输入 β 的字节数 ξ=LENGTH(β)*/

1)CASE β_i<*MASKBYTE*，表示 β_i 的标志位 σ_i 为 0，THEN ξ=1；

2)CASE β_i & *MASK2BYTES*= *MASK2BYTES*，表示 β_i 的标志位 σ_i 为 110，THEN ξ=2；

3)CASE β_i & *MASK3BYTES*= *MASK3BYTES*，表示 β_i 的标志位 σ_i 为 1110，THEN ξ=3。

(2)根据表 1 中的第 2 列剔除所有字节中的标记位 σ_i，依照 ξ=1,2,3 的不同情况进行处理。ξ=1(0xxxxxxx)：直接将输入返回为输出即可；ξ=2(110xxxxx 10xxxxxx)：取出 UTF8 编码最低字节的低 6 位，取出 UTF 编码高字节的低 2 位组合成 Unicode 编码的低字节。取出 UTF 编码高字节的第 4 位~第 6 位作为 Unicode 编码的高字节。ξ=3(1110xxxx 10xxxxxx 10xxxxxx)：取出 UTF8 编码最低字节的低 6 位和 UTF8 编码中间字节的低 2 位组合成 Unicode 编码的低字节。取出 UTF8 编码高字节的低 4 位和 UTF8 编码中间字节的第 3 位~第 6 位组合成 Unicode 编码的高字节。

(3)将剩余的有效位 δ_i 依次组合在一起，即可得到 Unicode 编码。

(4)下移至 β_{i+1} 进行新循环处理。

(5)结束。

程序代码可参考文献[4]。

5.3 测试用例和实验结果

为保证算法逻辑的正确性，本文在下列字符集合上进行了实验，测试过程中以表 4 中第 2 列作为输入，通过转换函数的处理，以期能够得到第 4 列的输出。测试结果证明本文采用的转换函数算法逻辑完全正确。

表 4　测试用例 β 描述

测试用例 β	UTF8 16 进制	UTF8 二进制	Teradata Unicode 16 进制	Teradata Unicode 二进制
主	E4 B8 8B	11100100 10111000 10111011	4E 3B	01001110 00111011
席	E5 B8 AD	11100101 10111000 10101101	5E 2D	01011110 00101101

6 结束语

本文通过介绍和证明 ETL 字符集兼容性定理，从理论上阐释了 Teradata 数据装载过程产生 6705 错误的原因，通过引入自定义 UTF 字符集转换函数的办法，避开 Teradata 数据库自身的字符集转换机制，从而克服了在 ETL 过程中处理某些非兼容性 Unicode 字符时产生的问题，大大地提高了 ETL 过程容错率。笔者已经在数据容量超过 1 TB 的 Teradata 平台上进行了实践，Unicode 数据存储的容错率可以达到 100%。系统运行结果证明该方法是可行有效的。

参考文献

[1] 黄河清，李治柱. 基于动态数据库的多国语言网站开发[J]. 计算机工程, 2005, 31(2): 80-83.

[2] Baird C, Chiba D, Chu W, et al. Oracle9i Database Globalization Support Guide[Z]. Oracle Corporation, 2002.

[3] NCR Corporation. Teradata RDBMS International Character Set Support [Z]. NCR Corporation, 2002.

[4] 鹿文鹏，薛若娟. Unicode 与 UTF8 编码转换方法研究[J]. 计算机时代, 2005, 11(9): 44-45.

（上接第 73 页）

图 2 显示出两种算法在执行时间及趋势上总体上均较为接近，但相同离群集规模时 AGOC 算法优于 COKAS 算法。

6 结束语

离群数据的出现可能由多种因素导致，其分类在 2 个层次上对知识发现具有影响：(1)在首次获得离群数据集时，需要对其进行详细分析，确定其中哪些部分是需要去除的噪声数据，哪些部分是需要进一步分析的普通离群数据；(2)对普通离群数据进行分析，从中寻求离群数据间的关联规则、分类和聚类，在更细粒度上探索离群数据本身的特征及出现原因。本文提出的基于邻接图的离群聚类算法(AGOC)以离群对象的关键域子空间为基础，在去除不存在关键域子空间的噪声数据点后，计算离群共享属性相似度，构建相应的离群邻接图并进行稀疏化，搜索其中边权值不低于给定离群相似度的最大完全子图，其结点集即对应于一个离群簇。本文对算法的聚类能力及性能进行了分析，实验结果证明了该算法是一个高效可用的方法。

AGOC 算法输出的是各个离群簇及相应的最大离群共享属性集，下一步笔者拟研究离群数据之间的关联规则模式以及结果的可视化技术。

参考文献

[1] Angiulli F, Pizzuti C. Outlier Mining in Large High-dimensional Data Sets[J]. IEEE Trans. on Knowledge and Data Engineering, 2005, 17(2): 203-215.

[2] Ramaswamy S, Rastogi R, Shim K. Efficient Algorithms for Mining Outliers from Large Data Sets[C]//Proc of the ACM SIGMOD International Conference on Management of Data. [S. l.]: ACM Press, 2000.

[3] Ranta R, Dorr V L, Heinrich C, et al. Iterative Wavelet-based Denoising Methods and Robust Outlier Detection[J]. IEEE Signal Processing Letters, 2005, 12(8): 557-560.

[4] Agyemanga M, Barkera K, Alhajj R. Web Outlier Mining: Discovering Outliers from Web Datasets[J]. Intelligent Data Analysis, 2005, 9(5): 473-486.

[5] Knorr E M, Ng R T. Finding Intensional Knowledge of Distance-based Outliers[C]//Proceedings of the 25th International Conference on Very Large Data Bases. New York, USA: Morgan Kaufmann, 1999.

[6] 金义富，朱庆生，邹咸林. 一种基于关键域子空间的离群数据聚类算法[J]. 计算机研究与发展, 2007, 44(4): 651-659.