IBM WebSphere DataStage

学习

目　　录

目　　录 I

1. 引言 1

1.1. 编写目的 1

1.2. 帮助使用 1

2. 产品概述 1

3. 简单 JOB 示例 4

4. 常规应用 26

4.1. 常用组件使用方法 26

4.1.1. Sequential file 26
4.1.2. Annotation 29
4.1.3. Change Capture Stage 30
4.1.4. Copy Stage 32
4.1.5. Filter Stage 33
4.1.6. Funnel Stage 34
4.1.7. Tansformer Stage 35
4.1.8. Sort Stage 36
4.1.9. LookUp Stage 37
4.1.10. Join Stage 38
4.1.11. LookUp Stage 和 Join Stage的区别 39
4.1.12. Merge Stage 39
4.1.13. Modify Stage 40
4.1.14. Data Set Stage 41
4.1.15. File Set Stage 43
4.1.16. Lookup File Set Stage 44
4.1.17. Oracle Enterprise Stage 47
4.1.18. Aggregator Stage 49
4.1.19. Remove Duplicates Stage 51
4.1.20. Compress Stage 52
4.1.21. Expand Stage 53
4.1.22. Difference Stage 54
4.1.23. Compare Stage 56
4.1.24. Switch Stage 57
4.1.25. Column Import Stage 58
4.1.26. Column Export Stage 60

5. 高级应用 62

5.1. DATASTAGE BASIC接口 62

5.2. 自定义STAGE TYPE 63

5.2.1. Wrapped Stage 63
5.2.2. Build Stage 68
5.2.3. Custom Stage 76

5.3. 性能调优 76

5.3.1. 优化策略 76
5.3.2. 关键问题分析 80
5.3.3. 并行度 81

5.3.4. 处理建议 81
5.3.5. 其它 82
5.3.6. 机器的对称性 82
5.3.7. 并行调度测试说明： 82
6. 开发经验技巧汇总 83

1. 引言

1.1. 编写目的

为了使大家尽快掌握DataStage的功能及其使用，特此编写此文档。

1.2. 帮助使用

由于DataStage产品功能强大，配置复杂，本文档只是对DataStage作个简单介绍，和一些简单操作的示例，只能起个抛砖引玉的作用，具体的操作及方法还需要查看相应的帮助文档

我们通常使用的帮助有如下两个途径。

a. 智能化的帮助功能；产品在几乎所有的操作窗口都有一个Help键，点击该键可以显示出当前使用的界面的功能和各项选项的具体的说明和操作方法。
b. Online Manuals；就是产品安装后程序组中的DataStage Documents，里面更加综合、全面的对整个产品从普通到高级，从Server版到Enterprise Edition版，从For Windows到For Unix等等方面的详尽叙述。能够帮助更加系统、全面的掌握该产品。

2. 产品概述

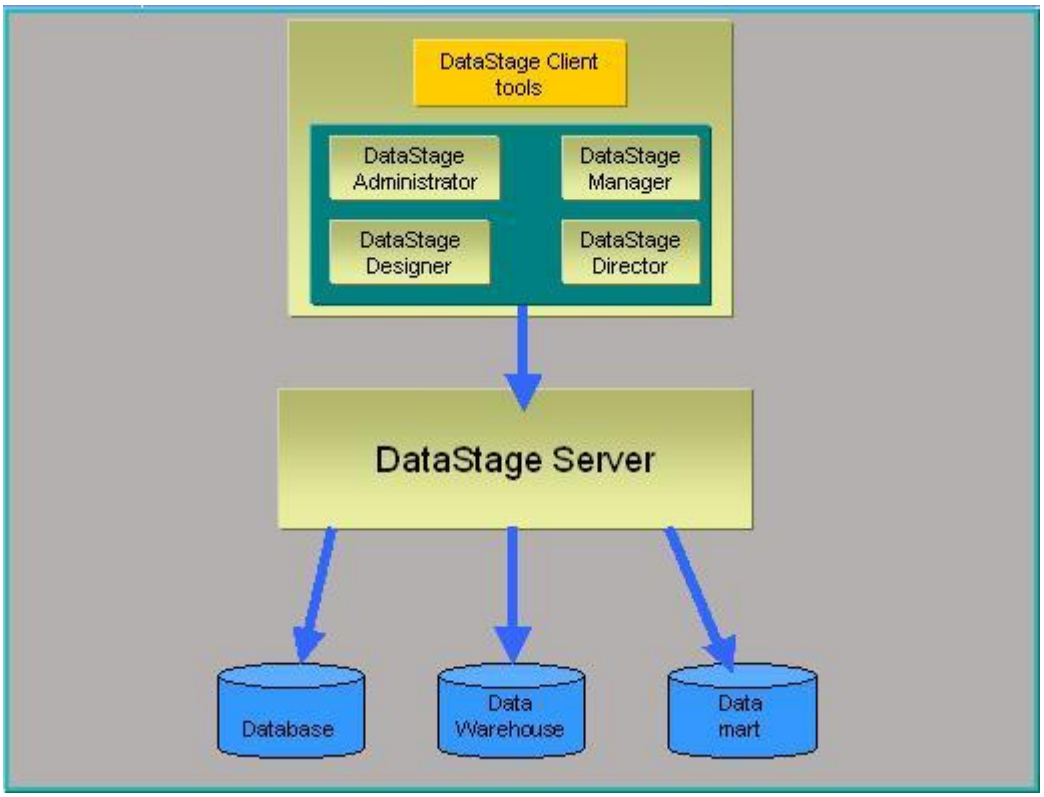
DataStage支持大容量数据的收集、整合和转换，数据从简单结构到很复杂的结构。基于高可扩展性的软件架购，企业版使得企业能够通过高性能来解决大部分业务问题，并行处理大容量数据。强大的企业元数据管理能力使得可以在数据整合生命周期中在所有工具中共享和使用工具。

DataStage发布了四个核心功能来成功实施企业数据整合：先进的开发和简单化的维护；企业级别的开发、监测和管理；在吞吐量和性能方面提供了无限制的高扩展的体系架构；端对端的企业级元数据管理。

DataStage 能够直接连接非常多的数据源，包括：

1、 文本文件
2、 XML 文件
3、 企业应用程序，比如 SAP、Siebel、Oracle 以及 PeopleSoft
4、 几乎所有的数据库系统，比如 DB2、Oracle、SQL Server、Informix 等
5、 Web services
6、 WebSphere MQ

DataStage使用了Client-Server架构，如下所示。



图一、DataState企业版Client-Server架构

DataStage 的开发环境是基于 C/S 模式的，通过 DataStage Client 连接到DataStage Server 上进行开发。这些工具包括：

DataStage Administrator 的主要功能有以下几个：

1. 设置客户端和服务器连接的最大时间。
2. 添加和删除项目

在 Projects 这个标签中，可以新建或者删除项目，以及设置已有项目的属性。要用 DataStage 进行 ETL 的开发，首先就要用 DataStage Administrator 新建一个项目，然后在这个项目里面进行 ETL Job 的开发。

3. License 的管理

可以在 Licensing 标签中更新 License。

DataStage Designer 是 ETL Job 开发的核心环境，它的主要功能可以概括为以下三个方面：

1. ETL Job 的开发

DataStage Designer 里面包含了 DataStage 为 ETL 开发已经构建好的组件，主要分为两种，一种是用来连接数据源的组件，另一种是用来做数据转换的组件。利用这些组件，开发人员可以通过图形化的方式进行 ETL Job 的开发。

2. ETL Job 的编译

开发好 ETL Job 后，可以直接在 DataStage Designer 里面进行编译。如果编译不通过，编译器会帮助开发人员定位到出错的地方。

3. ETL Job 的执行

编译成功后，ETL Job 就可以执行了，在 DataStage Designer 里面可以运行 ETL Job。ETL Job 的运行情况可以在 DataStage Director 中看到，这方面的内容将在介绍 DataStage Director 的时候提到。

DataStage Manager

DataStage Manager 主要用来管理项目资源。一个项目可能包含多个 ETL Job，可以用 DataStage Manager 把一个项目里面的 ETL Job 导出来。然后再用 DataStage Manager 导入到另外一个项目中去，利用这个功能一方面可以实现 ETL Job 的备份，另一方面就是可以在多个项目之间来重复使用开发好的 ETL Job。在 DataStage Manager 里面可以把数据库中的表结构直接导入到项目中来，供这个项目中的所有 ETL Job 使用。DataStage Designer 也提供了从数据库中直接导入表结构的功能。

DataStage Director

DataStage Director 主要有以下两个功能：

1. 监测 ETL Job 的运行状态

ETL Job 在 DataStage Designer 中编译好后，可以通过 DataStage Director 来运行它。前面在介绍 DataStage Designer 的时候提到在 DataStage Designer 中也可以运行 ETL Job，但是如果要监测 ETL Job 的运行情况还是要登陆到 DataStage Director 中。在这里，你可以看到 ETL Job 运行的详

细的日志文件，还可以查看一些统计数据，比如 ETL Job 每秒所处理的数据量。

2. 设置何时运行 ETL Job

ETL Job 开发完成后，我们可能希望 ETL Job 在每天的某个时间都运行一次。DataStage Director 为这种需求提供了解决方案。在 DataStage Director 中可以设置在每天、每周或者每月的某个时间运行 ETL Job。

3. 简单Job示例

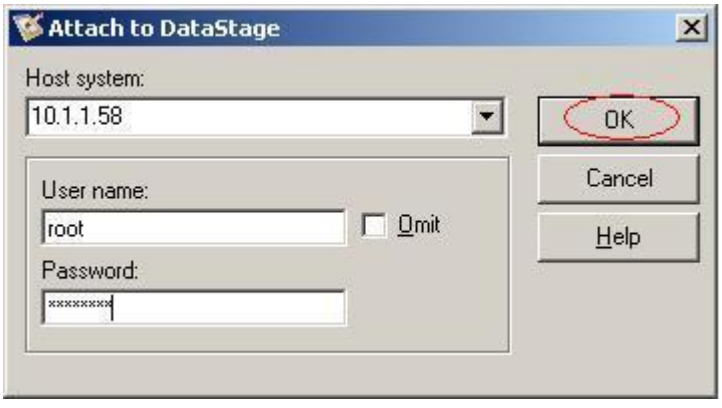
将要开发的 ETL Job 是把 DB2 数据库 Source 中的表 employee 的内容导入到另外一个 DB2 数据库 Target 中的表 employee 中去。其中两个数据库中的 employee 表的结构是相同的。employee 表的结构为：

字段名	类型	可否为空	是否为主键
ID	Varchar(20)	否	是
Name	Varchar(50)	是	否

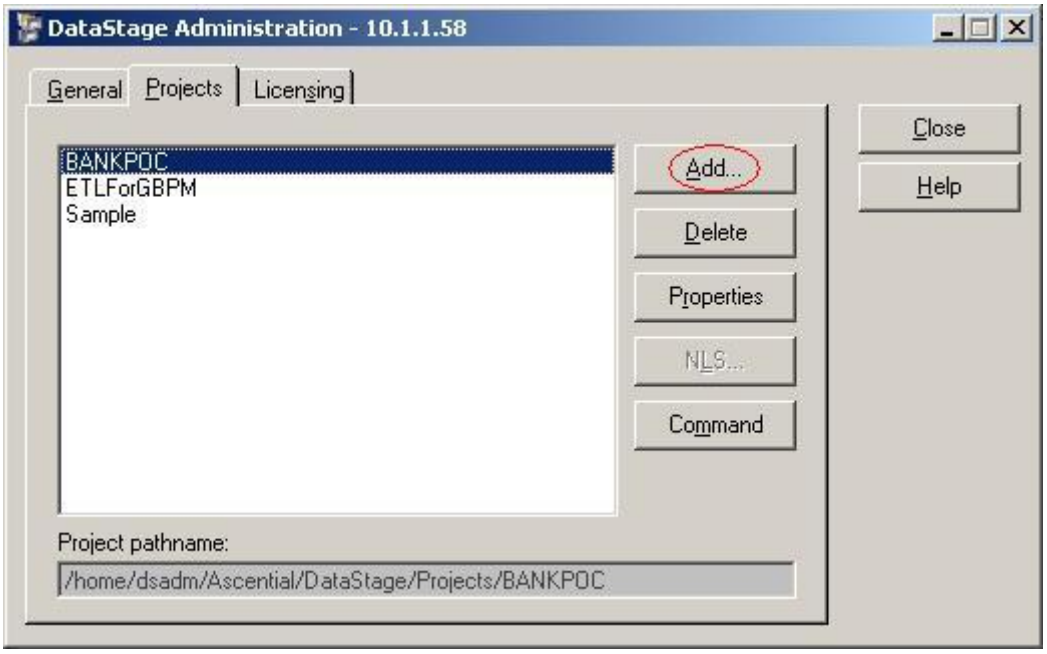
注：DB2 数据库的 Client 端必须和 DataStage Server 装在同一台机器上面

2. 新建项目

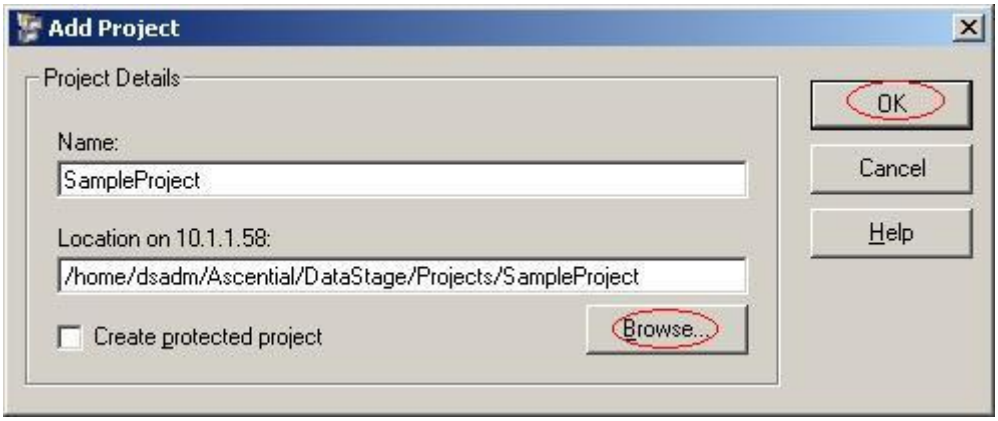
（1） 用 DataStage Administrator 登录到 DataStage Server。Host system 是安装 DataStage Server 的主机，输入它的 IP 地址或者主机名。另外再输入用户名和密码后,单击按钮"OK"；



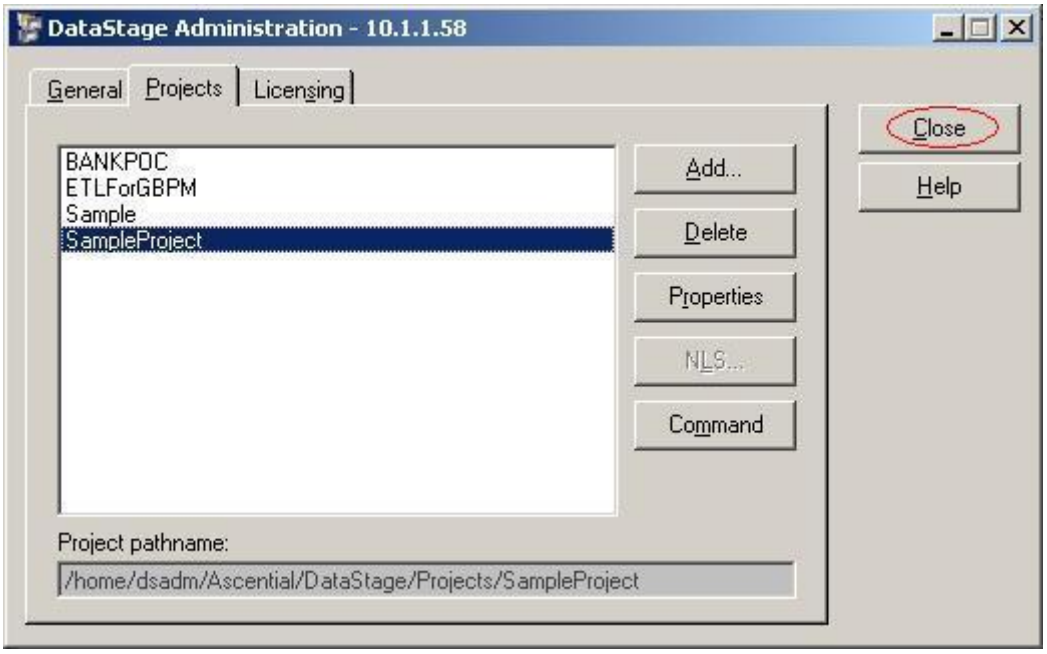
（2） 登录后，在标签 Projects 中可以看到目前这个 DataStage Server 上面所有的项目。单击按钮"Add"新建一个项目；



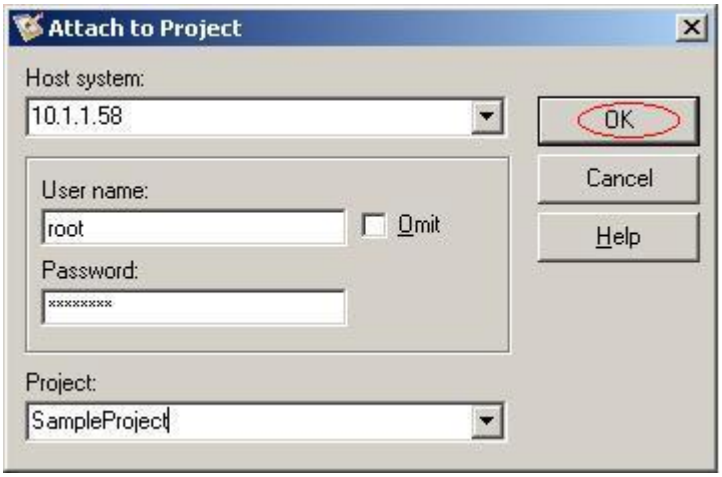
（3） 在弹出的对话框中输入项目名 SampleProject，项目存储的默认路径是 DataStage 安装路径的 Projects 目录下面，你可以通过单击按 钮"Browse"来改变默认路径。注意不要钩上选择框"Create protected project"，因为如果钩上的话你所创建的工程将没办法被改变。单击按钮"OK"；



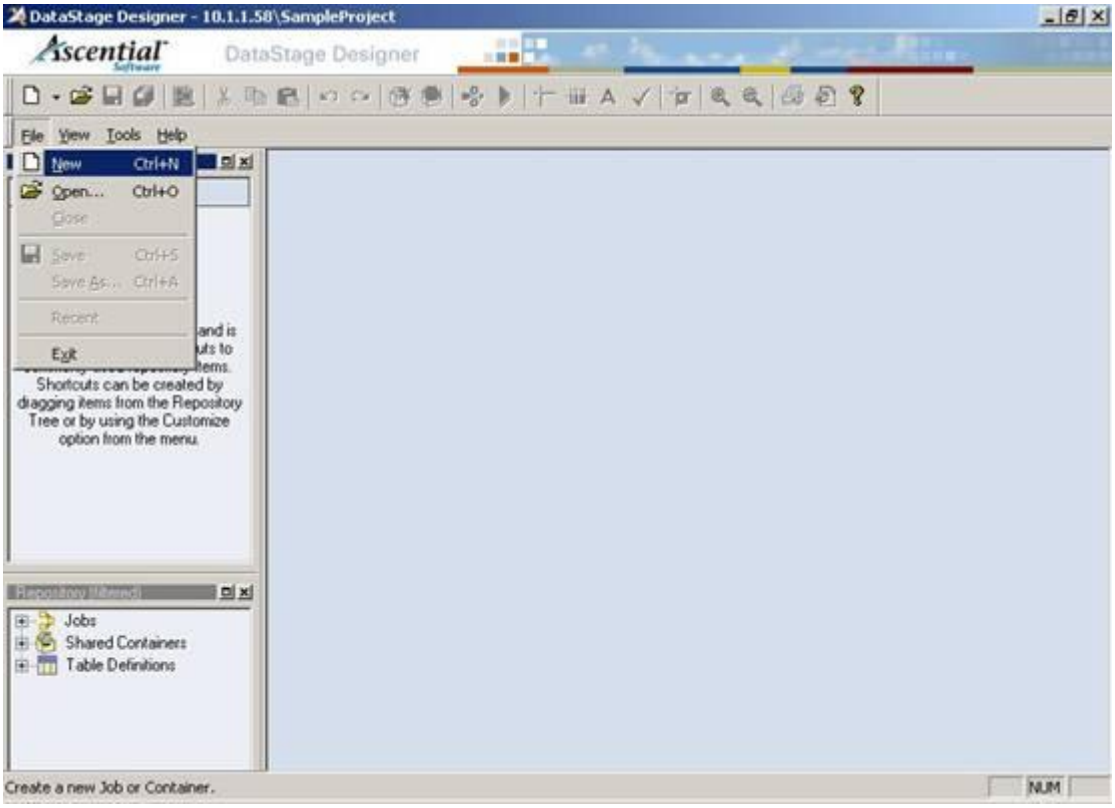
（4） 你会看到在项目列表里面已经有了我们刚创建好的项目 SampleProject，单击按钮"Close"关闭 DataStage Administrator；



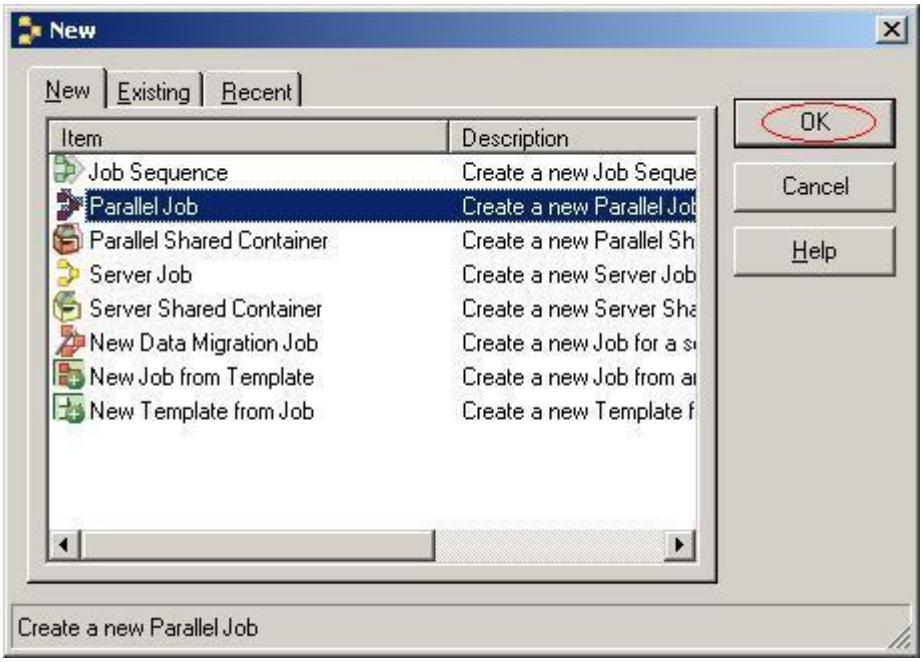
（5） 用 DataStage Designer 登陆到 DataSatge Server，输入 DataStage Server 的 IP 或主机名以及用户名和密码，并指定 Project 为我们刚才创建的项目 SampleProject。单击按钮"OK"；



（6） 在 DataStage Designer 当中单击 File’New 去创建一个新的 ETL Job；



（7） 选择"Parallel Job"，单击按钮"OK"；



There are three different types of job in DataStage:

1、Server jobs. These are available if you have installed Server. They run on the DataStage Server, connecting to other data

sources as necessary.

2、Mainframe jobs. These are available only if you have installed Enterprise MVS Edition. Mainframe jobs are uploaded to a mainframe, where they are compiled and run.

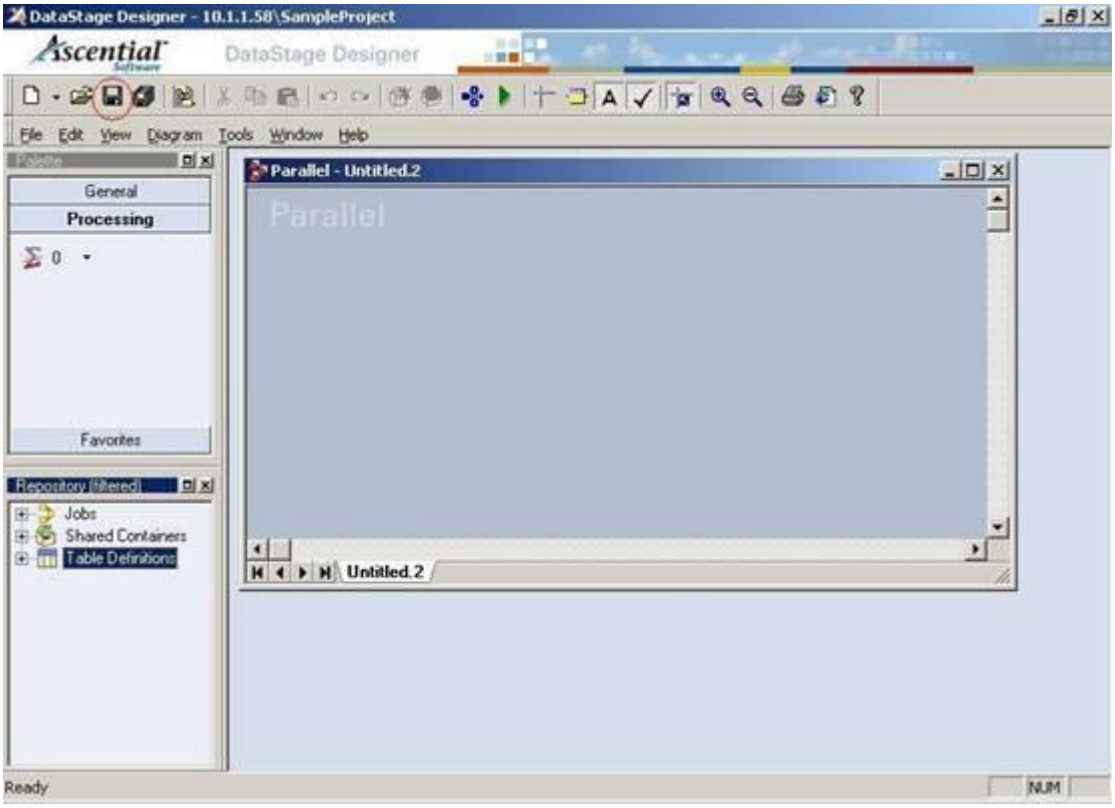
3、Parallel jobs. These are available only if you have installed the Enterprise Edition. These run on DataStage servers that are SMP, MPP, or cluster systems.

There are two other entities that are similar to jobs in the way they appear in the DataStage Designer, and are handled by it. These are:

1、Shared containers. These are reusable job elements. They typically comprise a number of stages and links. Copies of shared containers can be used in any number of server jobs and parallel jobs and edited as required.

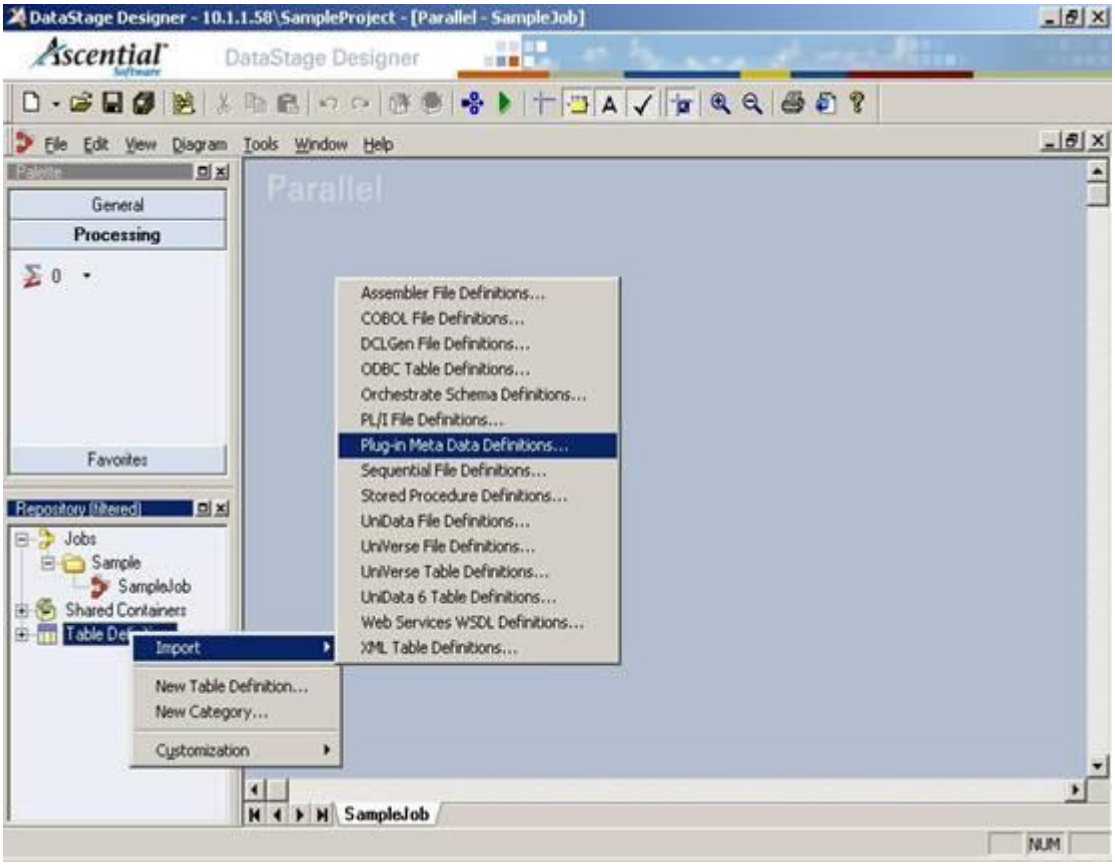
2、Job Sequences. A job sequence allows you to specify a sequence of DataStage server or parallel jobs to be executed, and actions to take depending on results. Job sequences are described

(8) 一个新的 ETL Job 已经创建了，单击工具栏上的图标"保存"，或者用快捷键"Ctrl＋S"来保存，这时候一个保存 ETL Job 的对话框会弹出来；

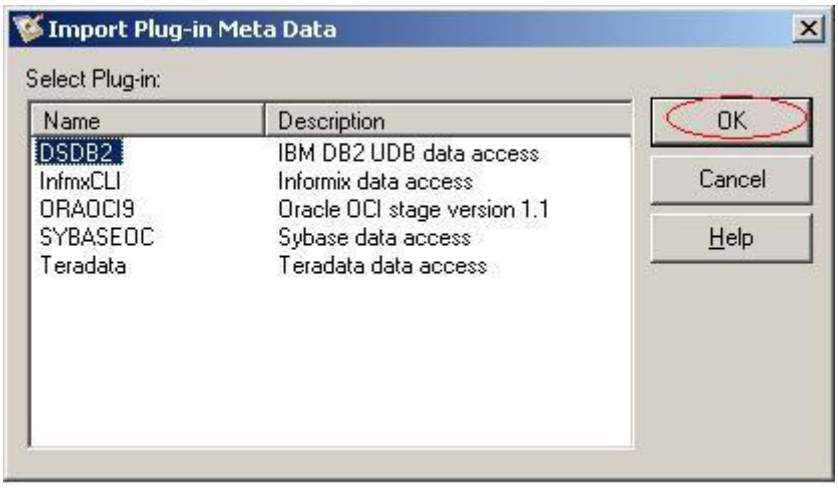


（9） 在弹出的对话框中。在 Job name 一栏输入"SampleJob"，在 Category 中输入"Sample"。单击按钮"OK"；

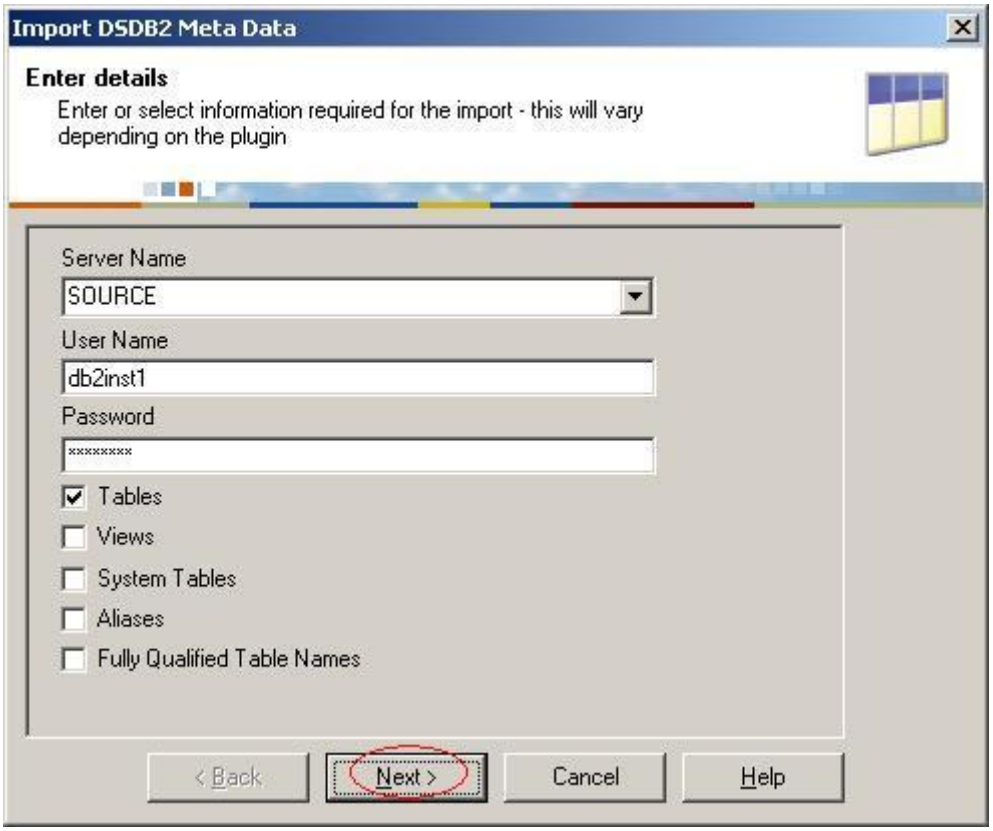
(10) 保存好刚创建的 ETL Job 后，我们用 DataStage Designer 来导入数据库的表结构。在 DataStage Designer 的左下方的 Repository 中右键单击"Table Definition"。然后选择 Import'Pug-in Meta Data Definitions…;



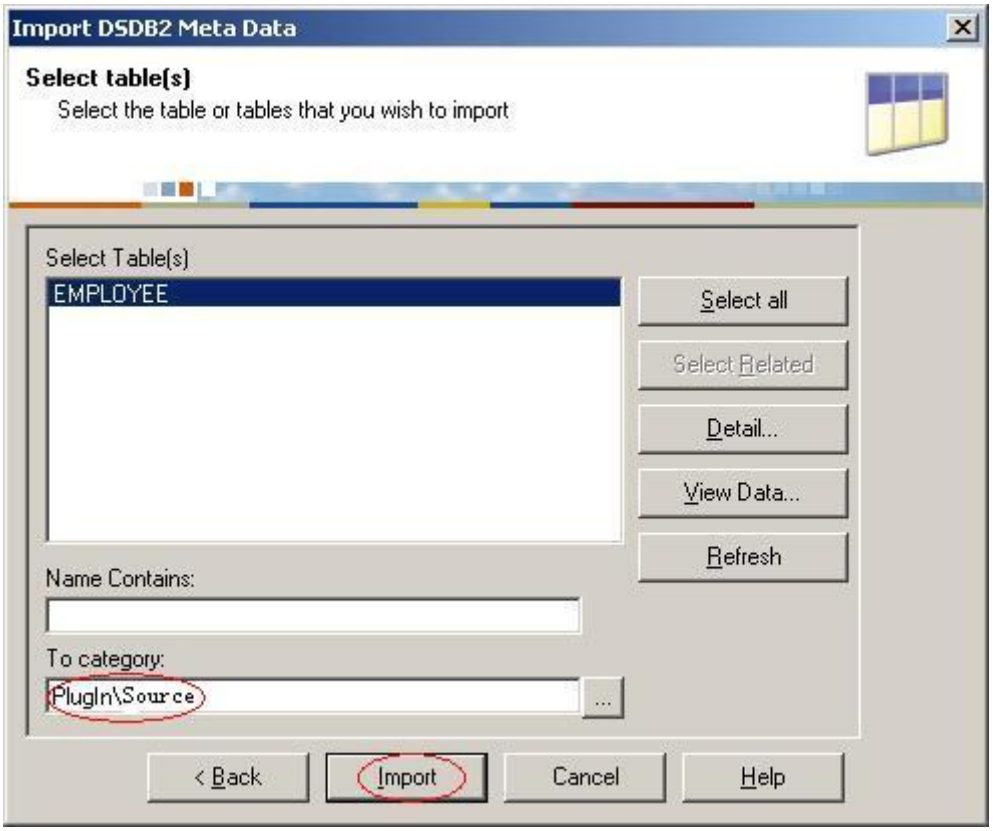
（11） 在弹出的对话框中选择 DSDB2，单击按钮 OK；



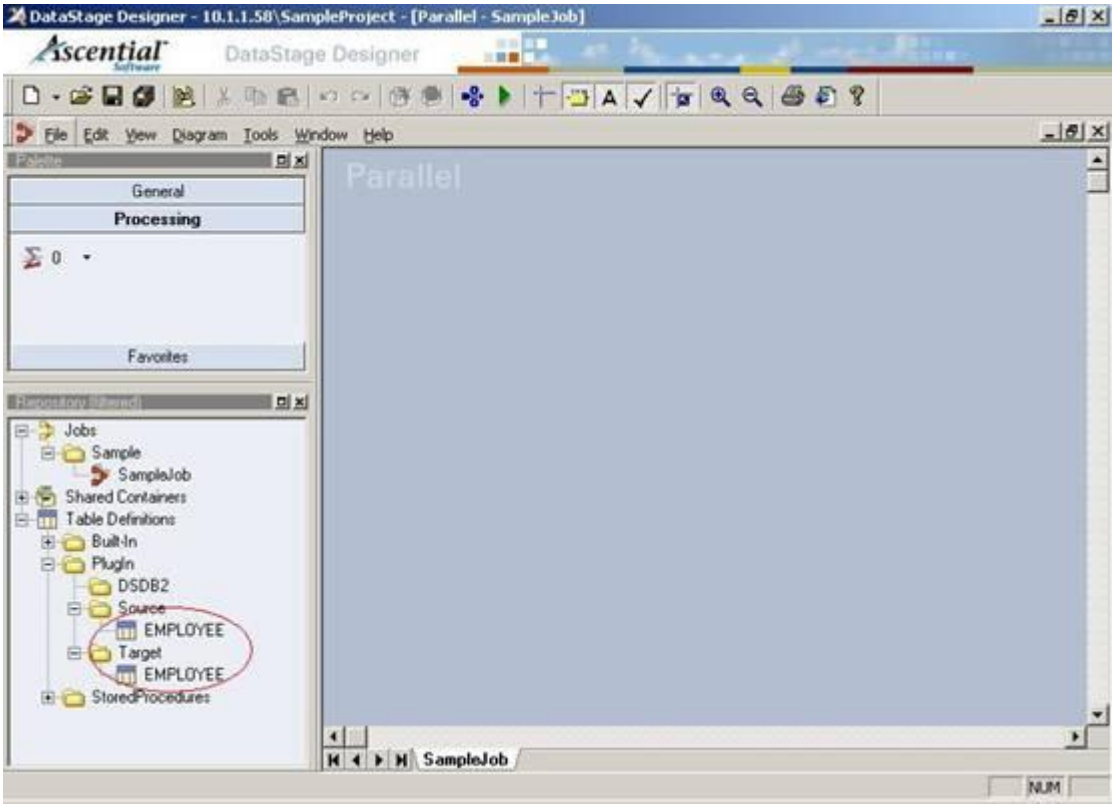
(12) 在弹出的对话框中，Server Name 选择 Source。输入用户名和密码，再钩上 Tables 选择框之后单击按钮 Next；



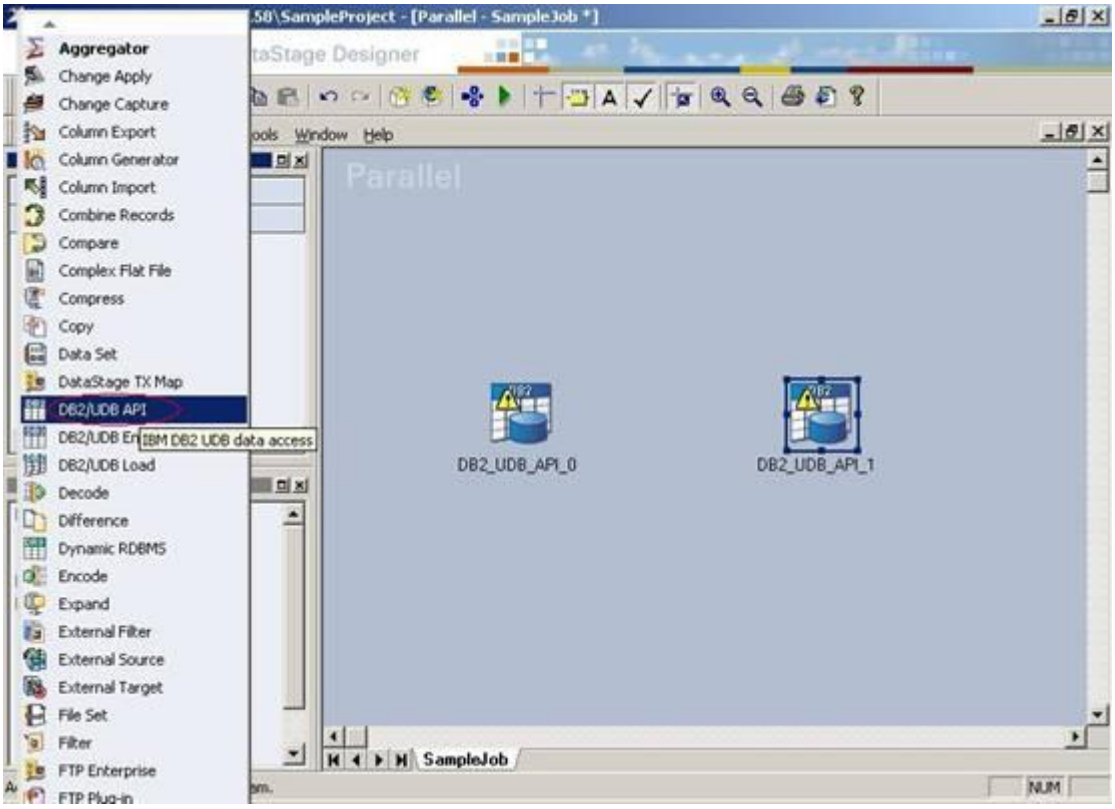
（13） 选择表 employee，把要保存到的目录改成 PlugIn\Source。然后单击按钮 Import.；



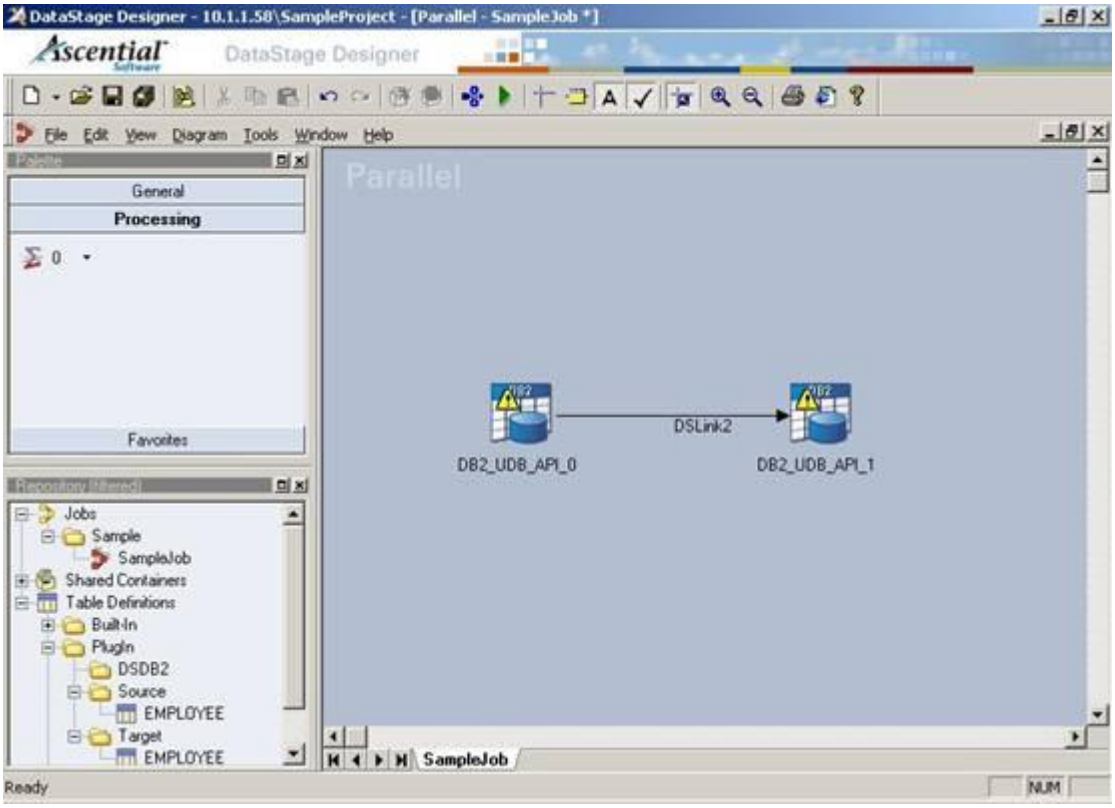
（14） 重复步骤 10－13 把存储在 Target 数据库中的表 employee 的表结构导入进来，这次存放的路径改成 PlugIn\Target。完成后，你会在 Repository 中看到你导入的表结构；



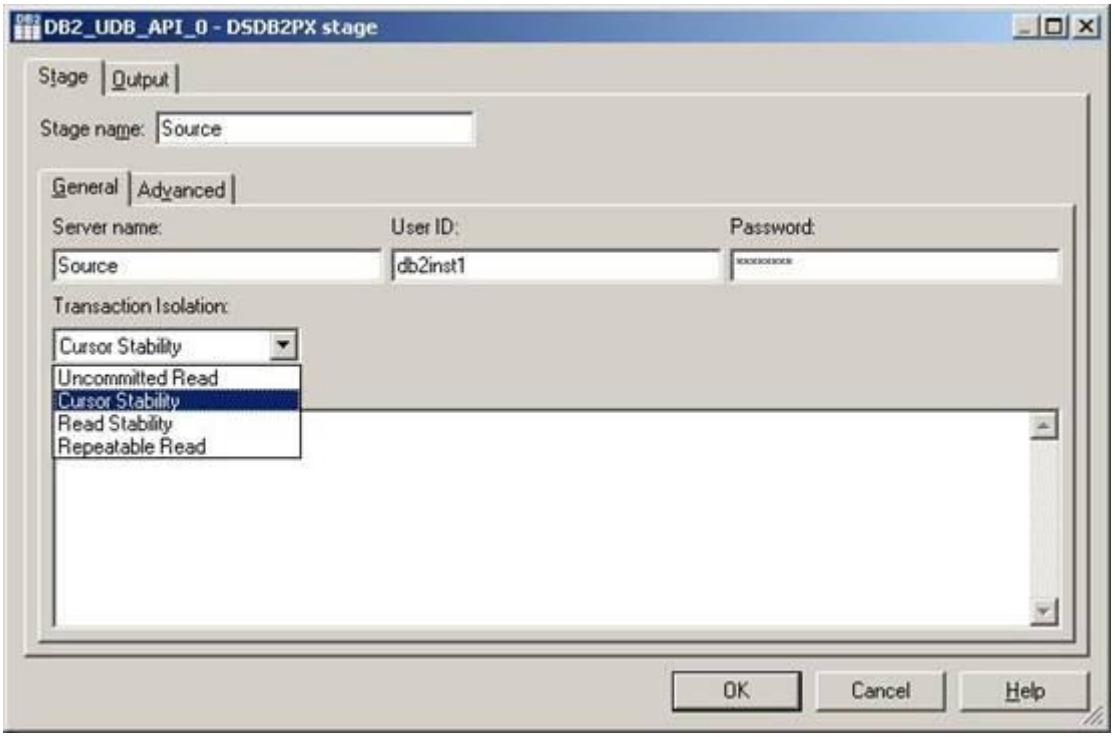
（15） 从左边的 palette 中拖入两个 DB2/UDB API Stage 到右边的面板上。DB2/UDB2 API Stage 是用来连接 DB2 数据库的，我们这两个 DB2/UDB API Stage 一个用来连接数据库 source，另一个用来连接数据库 Target；



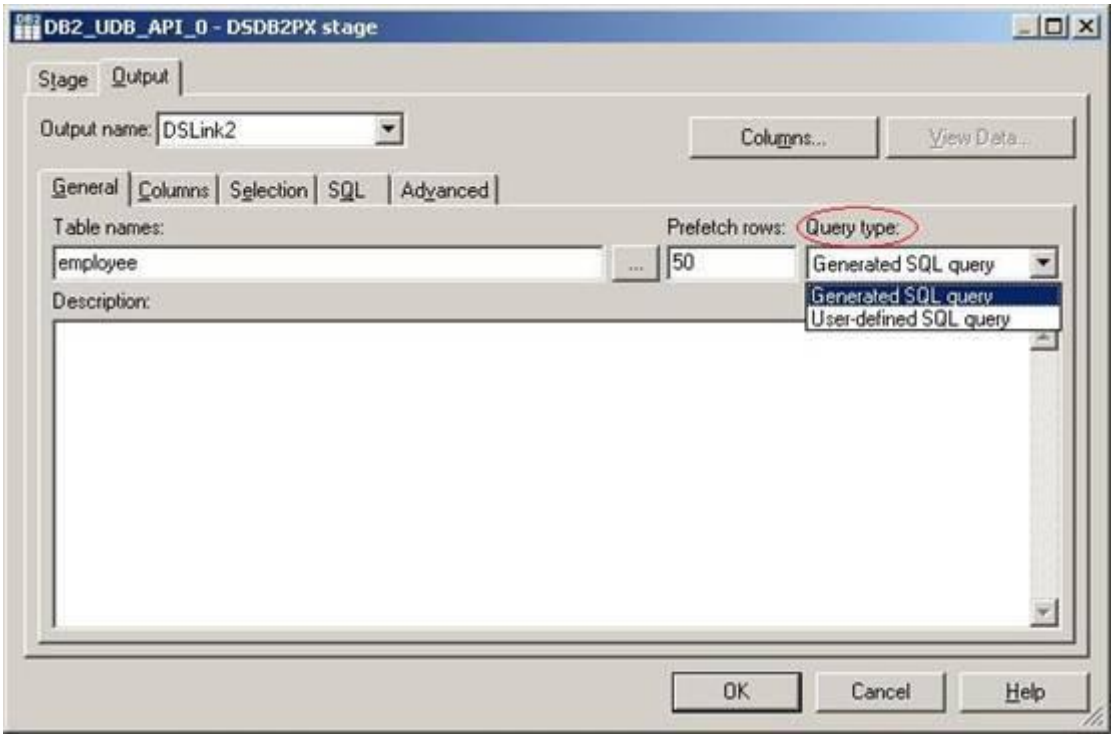
（16） 右键单击左边的 DB2/UDB API Stage 不要放开，一直拖拽鼠标到右边的 DB2/UDB2 API Stage 上面。这时候在这两个 Stage 之间会出现一条连线，代表了数据的流向。下面我们将配置这两个 Stage 的属性；



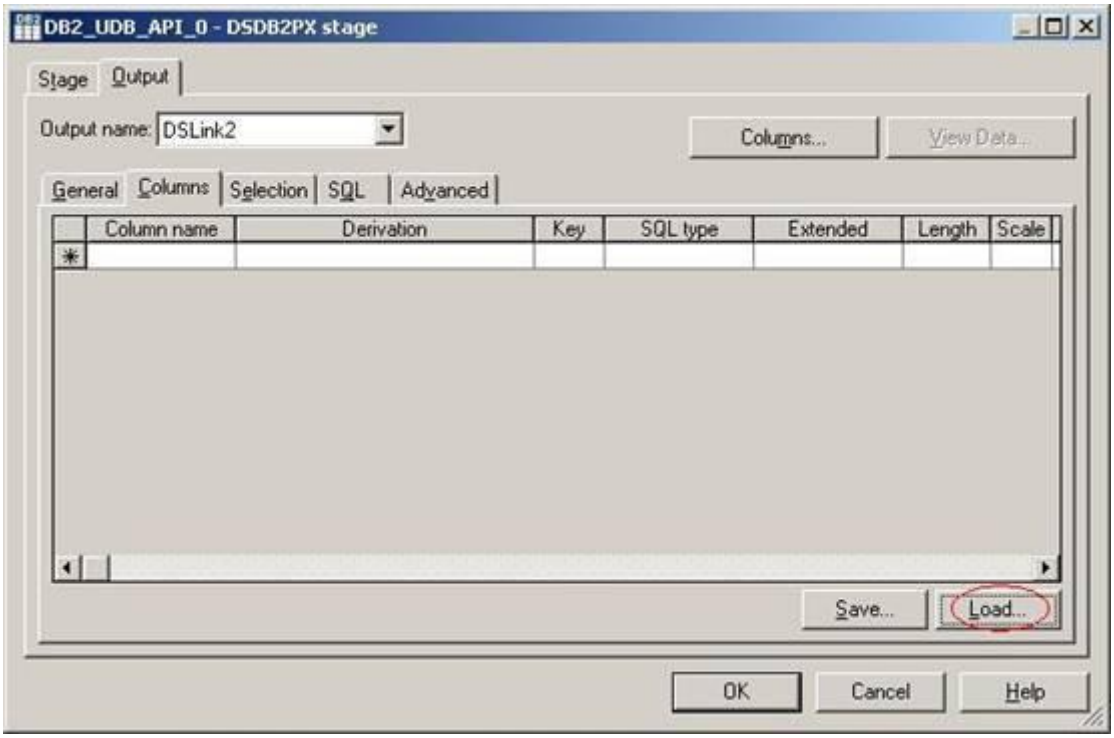
（17） 左键双击左边的 DB2/UDB API Stage，会弹出如下图所示的属性框。在标签 Stage 的子标签 General 中，设置 Stage name 为 Source，Server name 为 Source，User ID 和 Password 设置为右权限访问这个数据库的用户名和密码。Transaction Isolation 的默认的选项是 Cursor Stability，保持默认选项然后单击标签 Output；



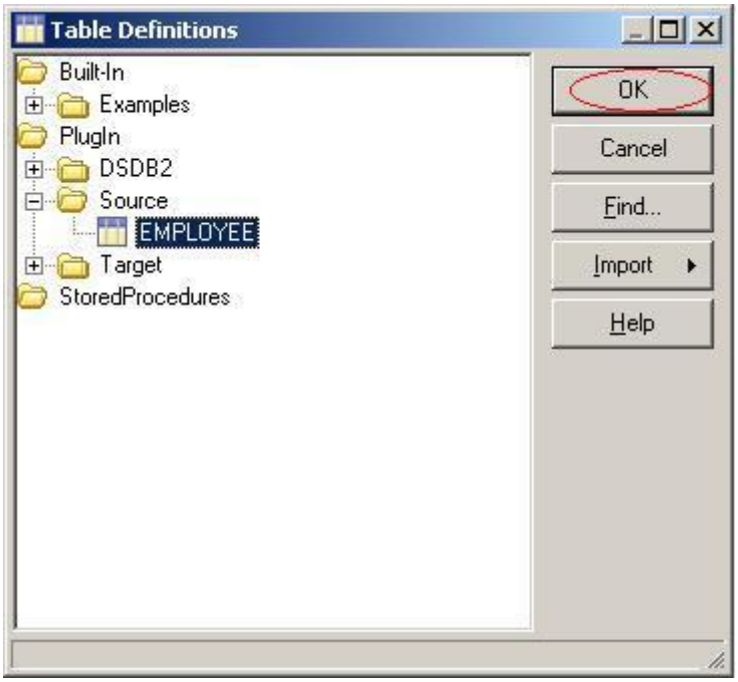
（18） 在标签 Output 的子标签 General 中，输入 Table names 为 employee，并在 Query type 下拉框中选择 Generated SQL Query。这样 DataStage 会自动帮你生成大部分的 SQL 代码。然后单击子标签 Columns；



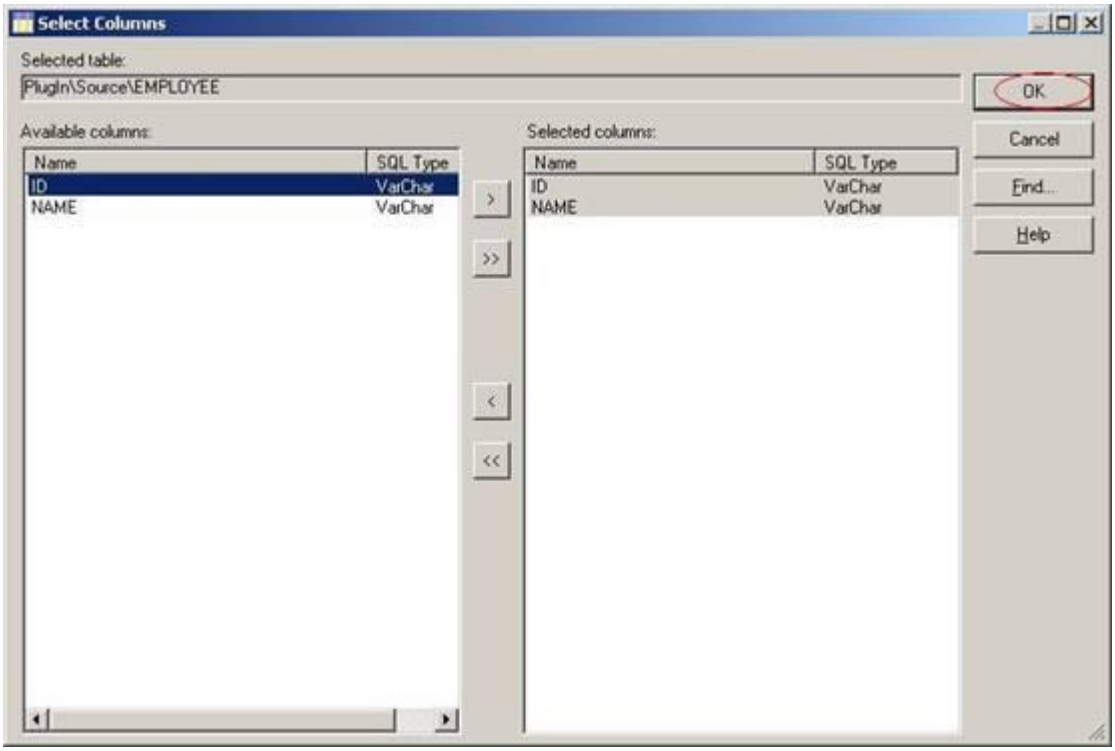
（19）在 Columns 子标签中单击按钮 Load 去导入刚才从数据库中导进来的表结构；



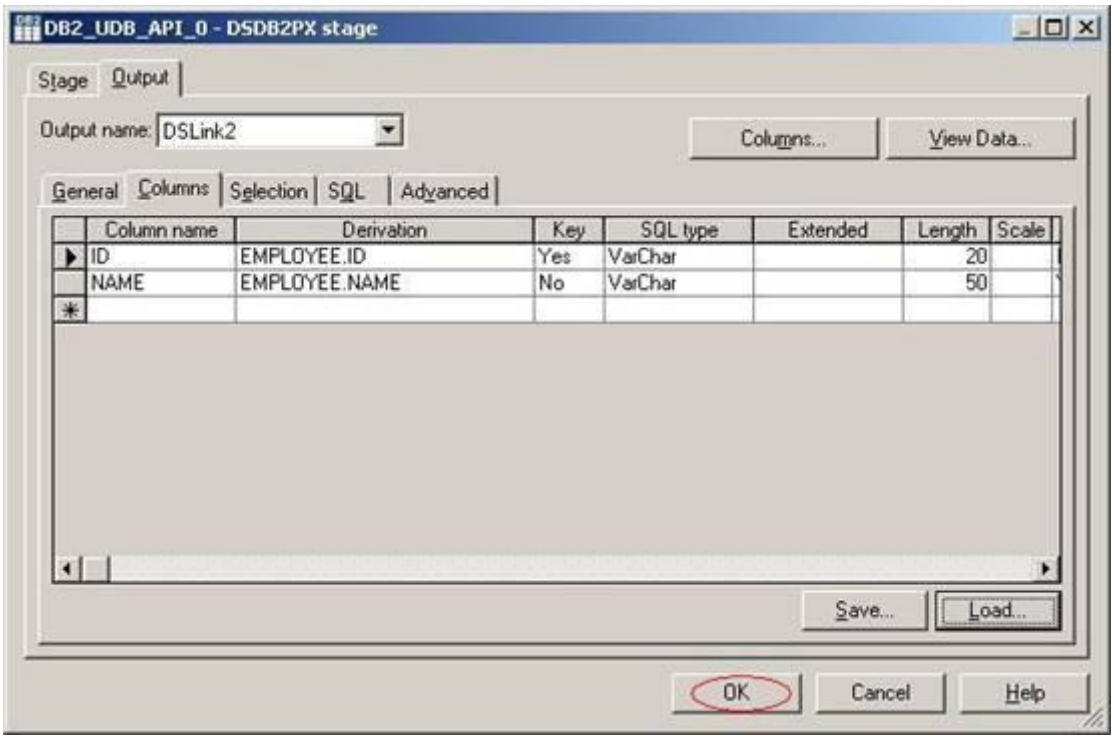
（20） 在弹出的对话框中选择目录 PlugIn\Source 中的表结构 employee，然后单击按钮 OK；



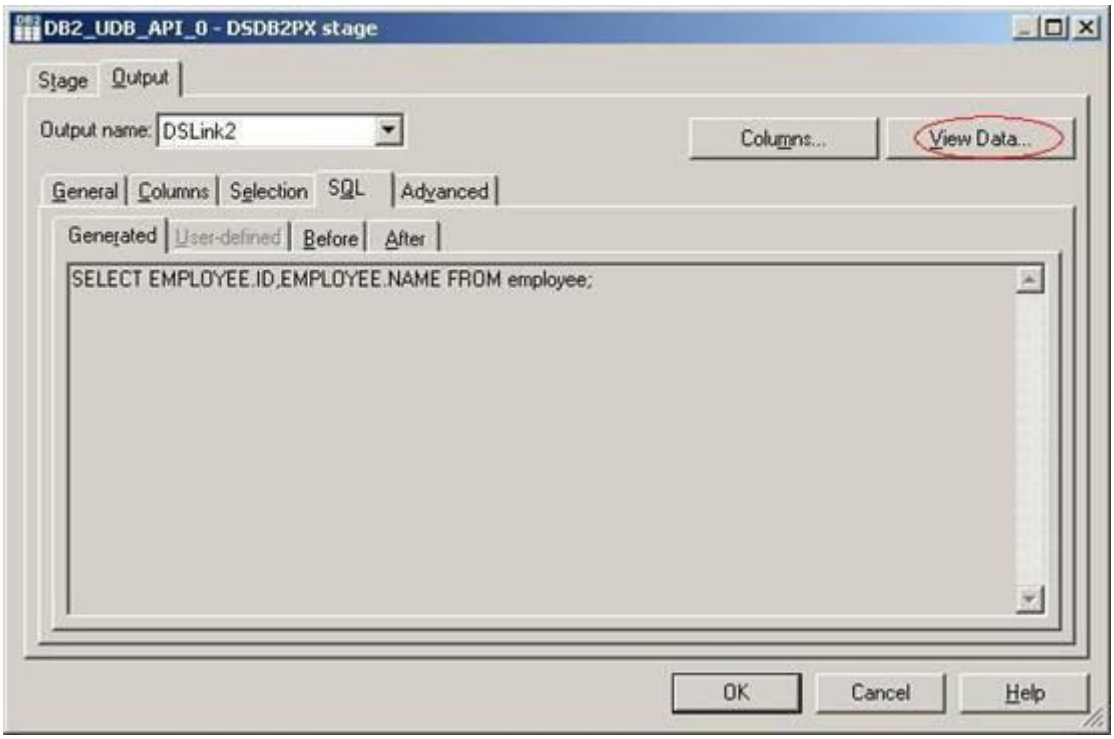
(21) 在弹出的对话框中选择要导入的表的列，默认是全选，保持默认并单击按钮 OK；



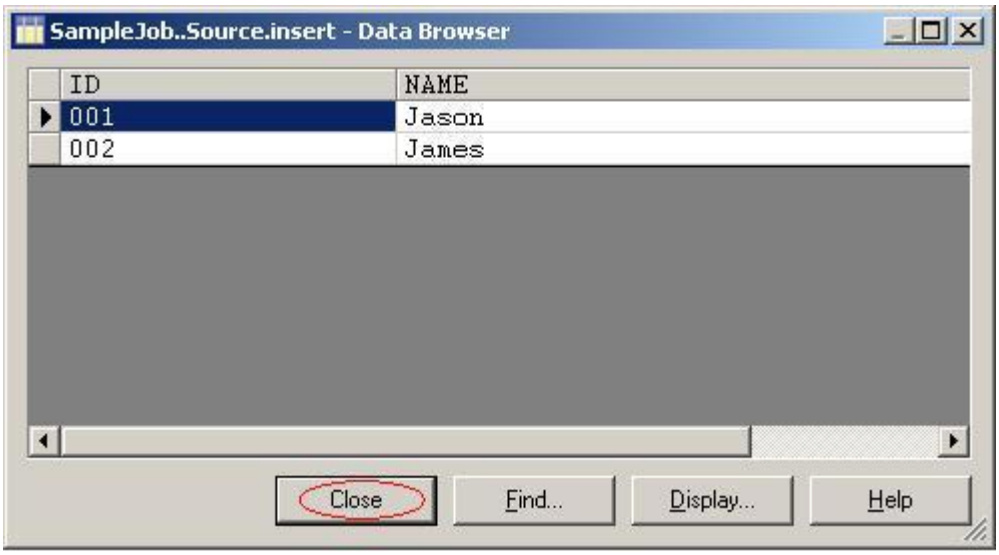
(22) 这时候你会看到表的字段已经被导入进来。单击子标签 SQL；



(23) 在子标签 SQL 中，你会看到系统自动生成的 SQL 语句。单击按钮 View Data 查看表 employee 中的数据；

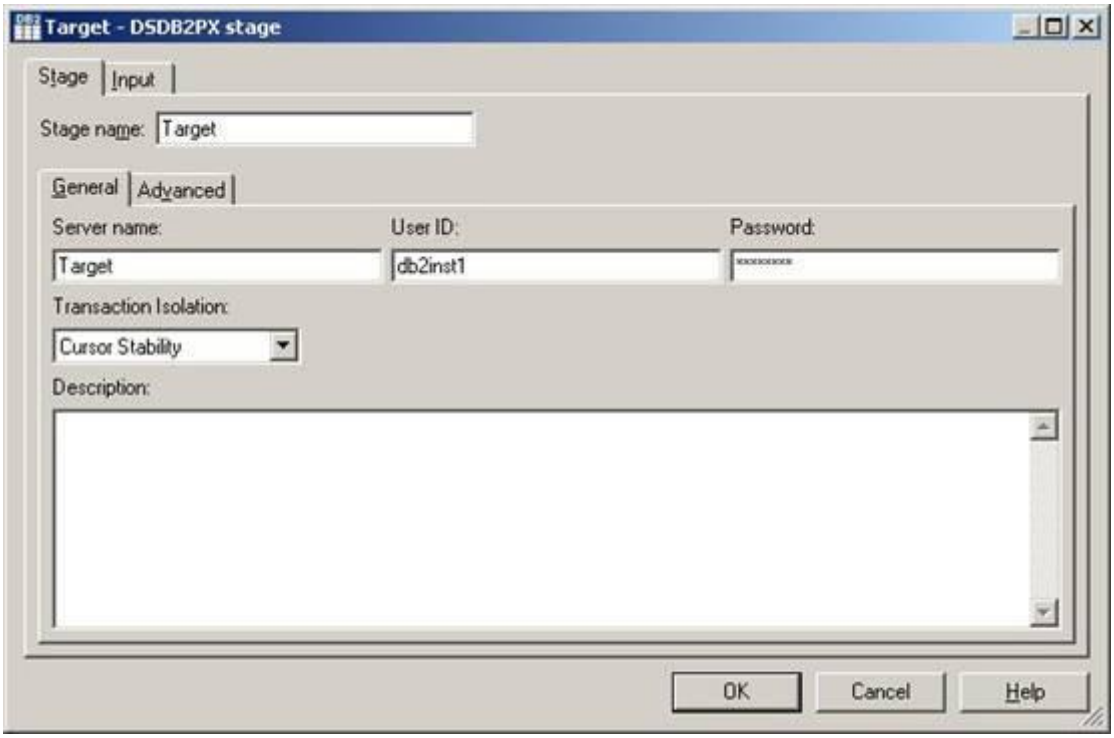


（24） 当前 employee 表中有两条数据。单击按钮 Close 关掉数据查看窗口；

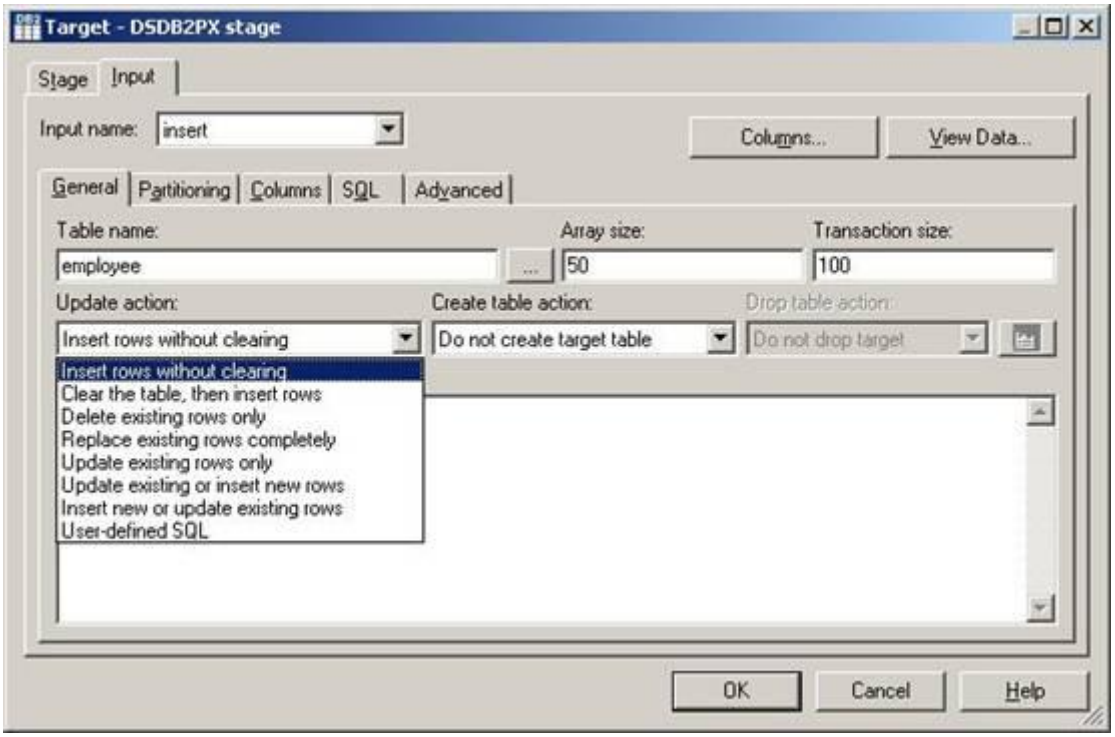


（25） 现在我们开始编辑用来连接目标数据库的 DB2/UDB API Stage 的属性。双击这个 Stage，弹出的属性设置窗口如下图所示。在标签 Stage 的子标签 General 中，Stage name 设置为 Target，Server name 设置为 Target，User ID

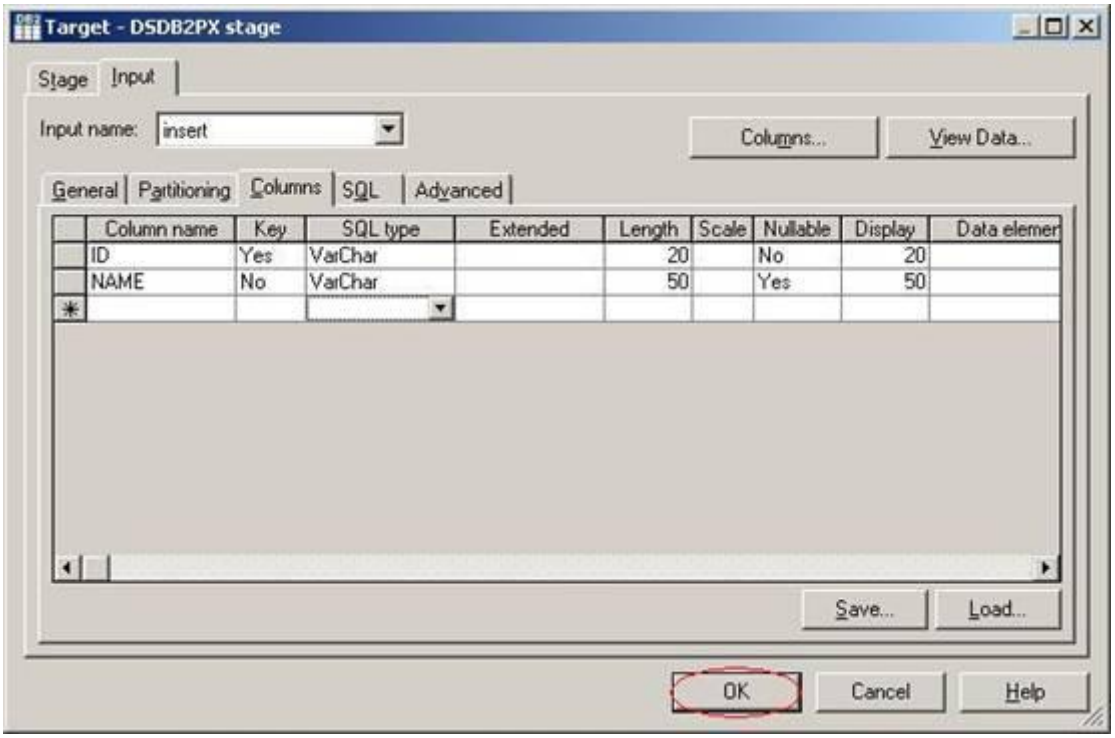
和 Password 分别设置为有权限对 Target 数据库进行操作的用户名和密码。其他属性保持默认值，然后单击标签 Input；



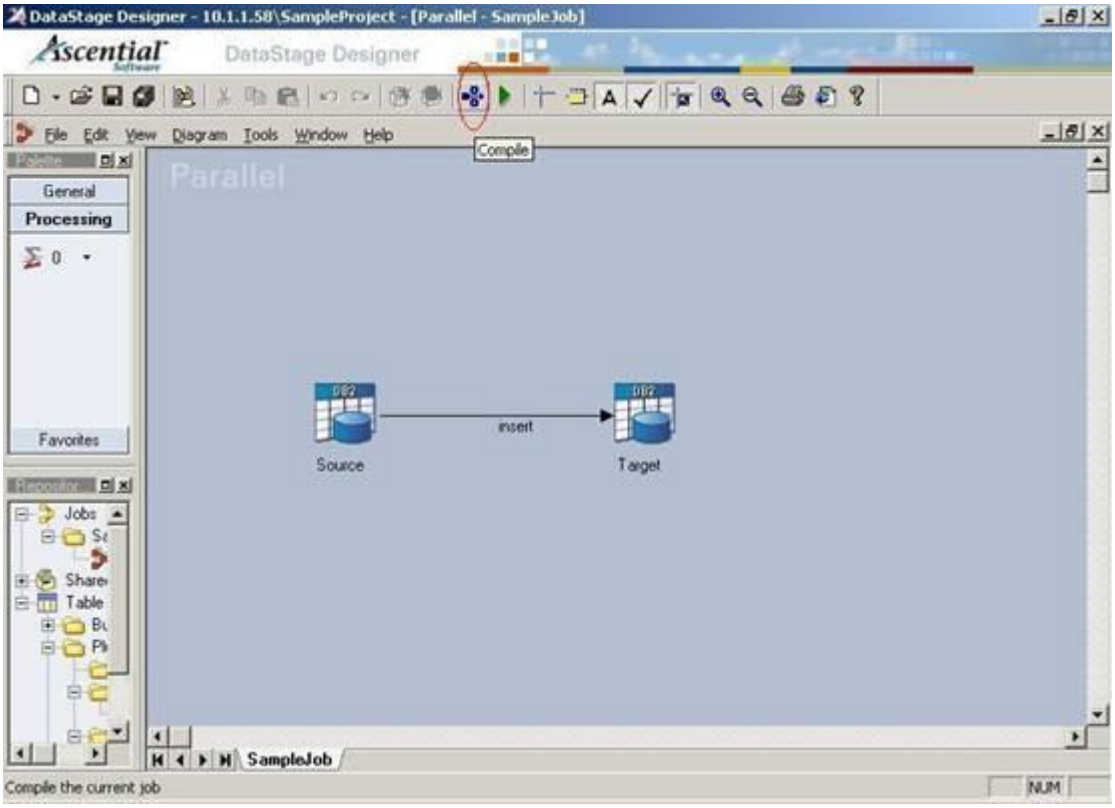
（26）在标签 Input 的子标签 General 中，设置 Table name 为 employee，Update action 选择 Insert rows without cleaning。Create table action 选择 Do not create target table。然后单击子标签 Columns；



（27） 在子标签 Columns 中，你会发现已经有表结构 load 进来了，这个表结构是和 source 数据库中的 employee 表的结构一致的。单击按钮 OK 并保存 ETL Job；

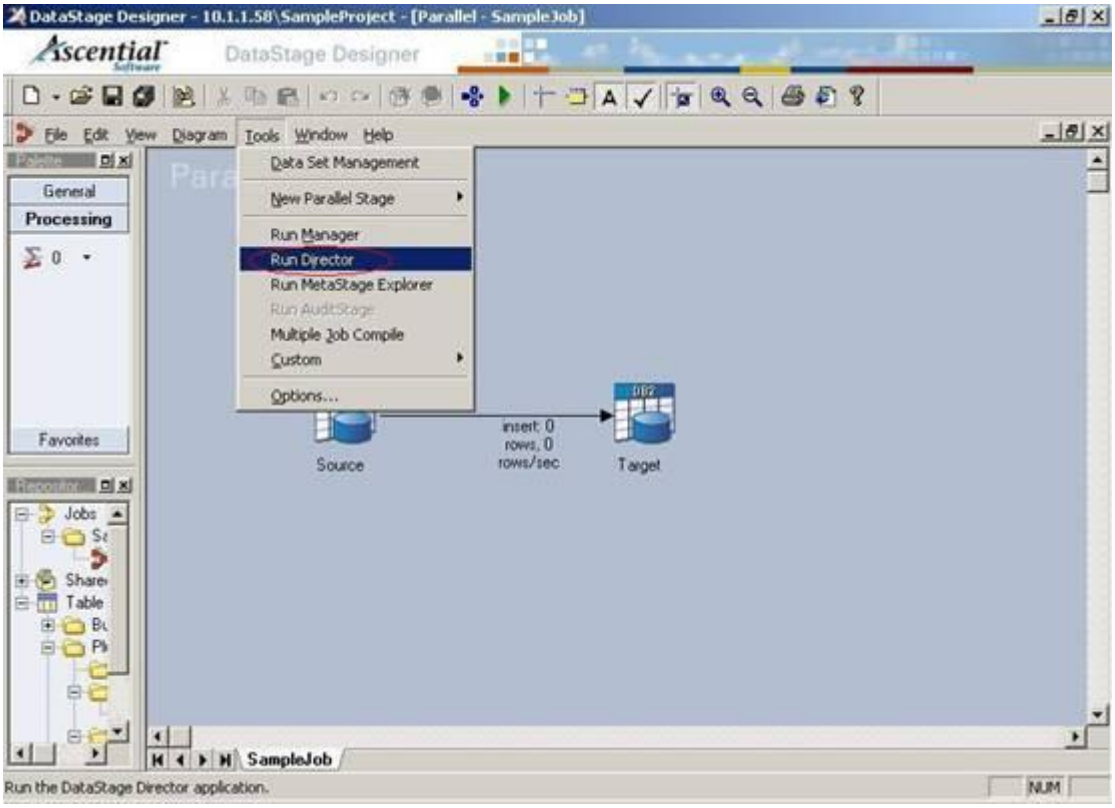


（28） 在工具栏中单击图标"编译"对刚开发完的 ETL Job 进行编译；

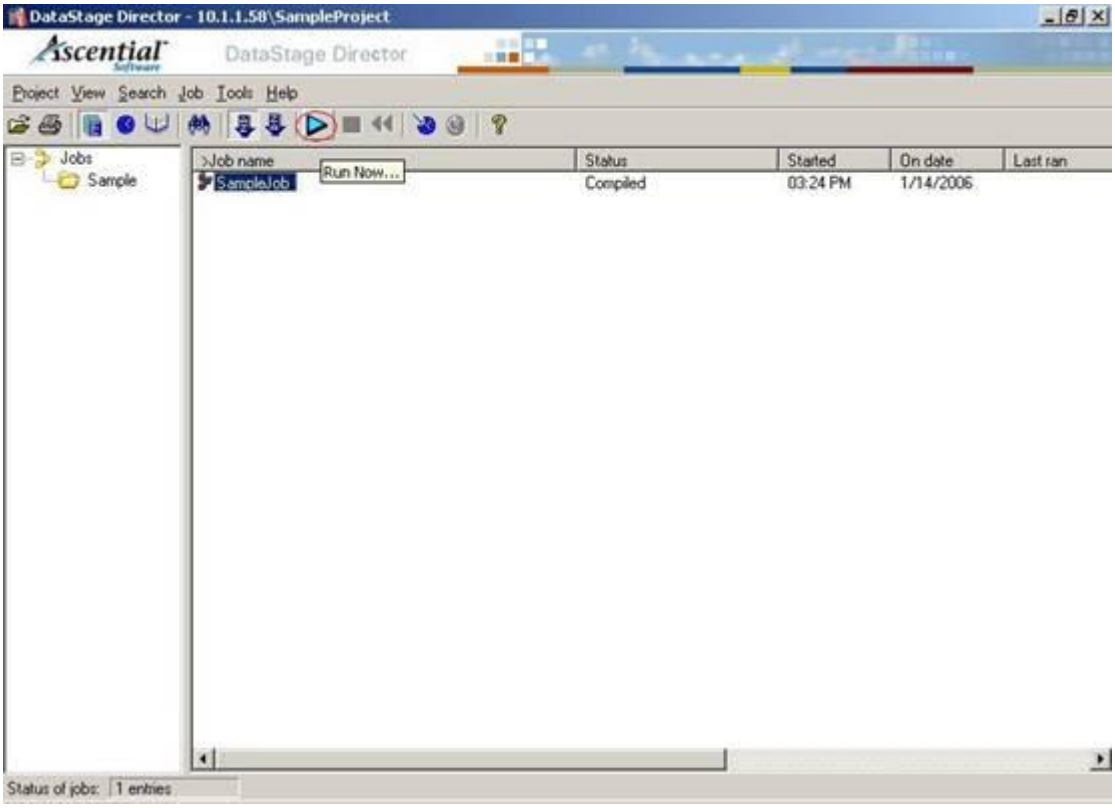


（29） 编译过程中会弹出一个对话框显示编译的进行情况。最终 ETL Job 编译成功后对话框中会显示如下图中所示的消息："Job successfully compiled with no errors"；

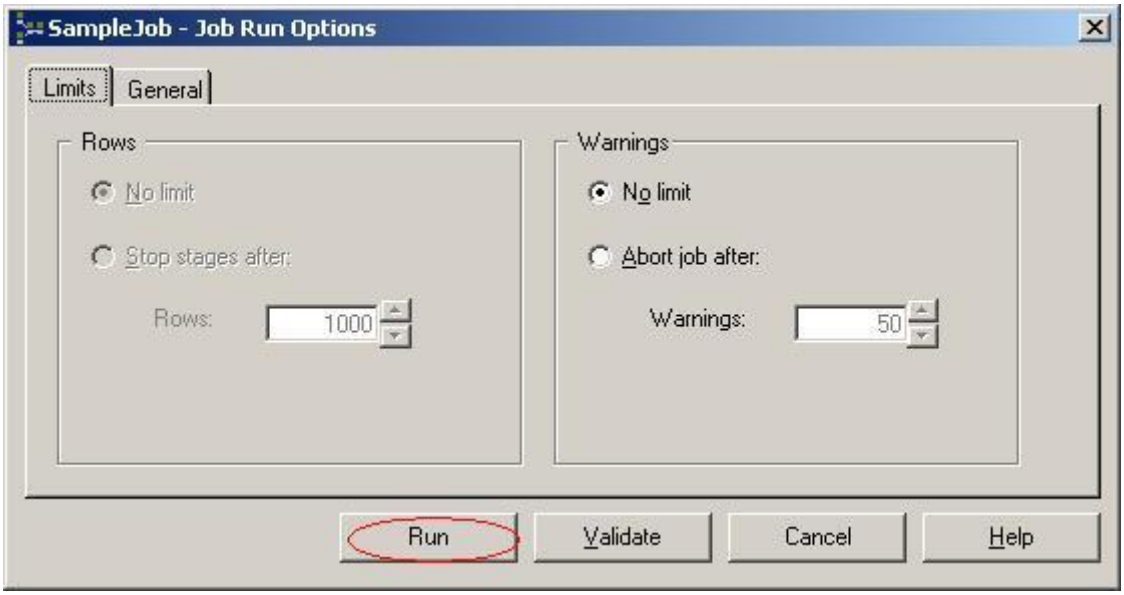
（30） 编译成功后，我们打开 DataStage Director 来运行我们开发的 ETL Job。从 DataStage Designer 中打开 DataStage Director 的方法为：从菜单栏中选择 Tools'Run Director；



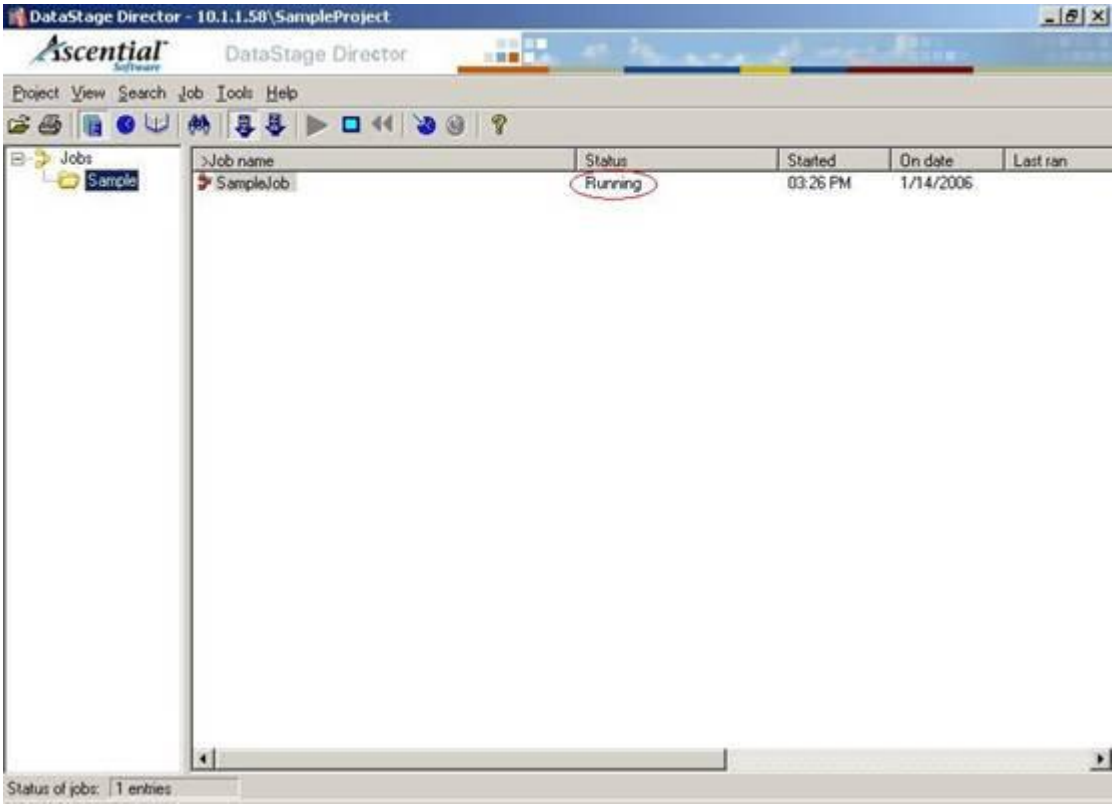
（31） 打开 DataStage Director 后你会在 Sample 目录下面发现我们开发好的 ETL Job SampleJob，状态为 Compiled。选择 SampleJob，然后单击工具栏中的"运行"按钮；



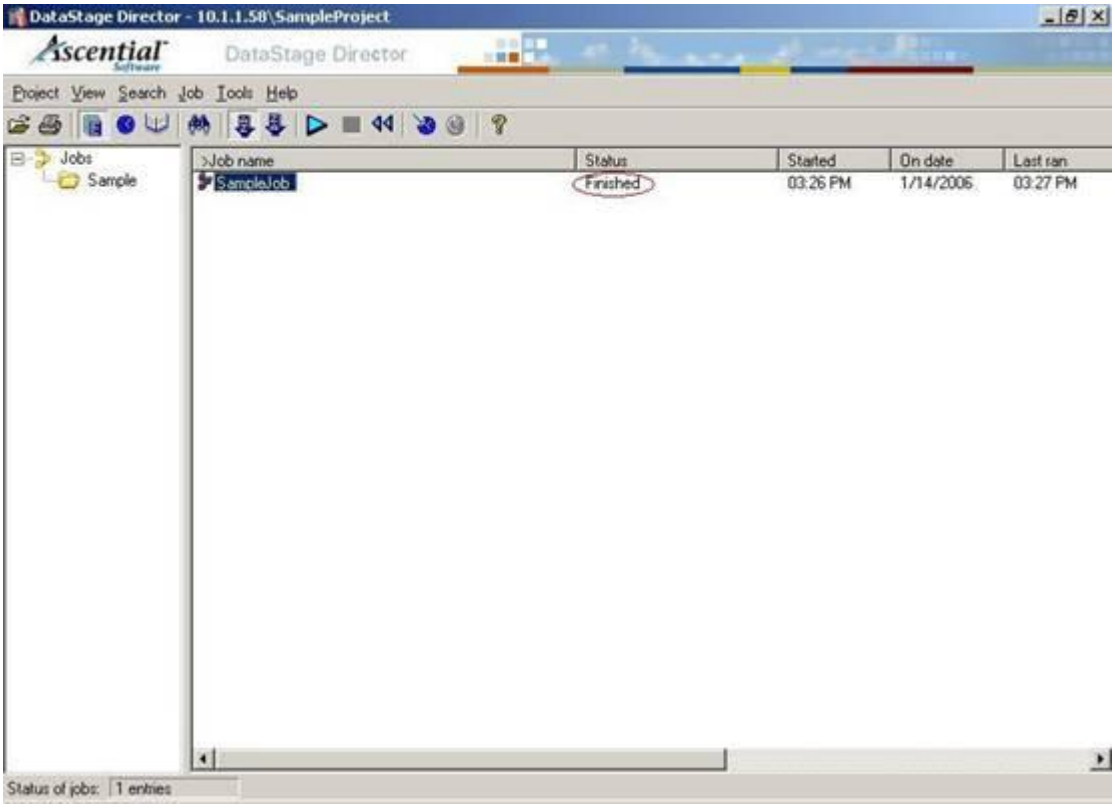
（32） 在弹出的对话框中可以设置运行的参数，比如出现多少个 warning 后 ETL Job 会自动中止掉。 我们保持这个对话框中的默认设置，单击按钮 Run；



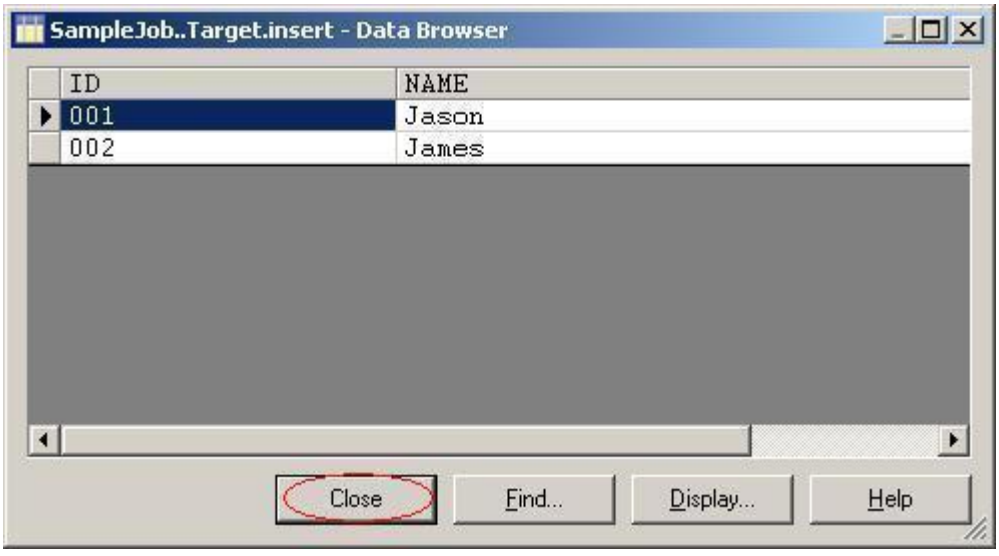
(33) 这时候你会注意到 SampleJob 的状态从 Compiled 变成了 Running，等到 SampleJob 的状态变成 Finished 后，该 ETL Job 的运行就结束了；



（34） 如下图所示，SampleJob 的状态变成了 Finished。SampleJob 成功结束运行；



（35） 到 DataStage Designer 中，用 View Data 功能查看目标数据库 Target 中 employee 表中的数据。你会发现和源数据库 source 中的 employee 表中的数据是一样的。也说明我们开发的 ETL Job 成功的完成了我们想要它完成的任务。



4. 常规应用

4.1. 常用组件使用方法

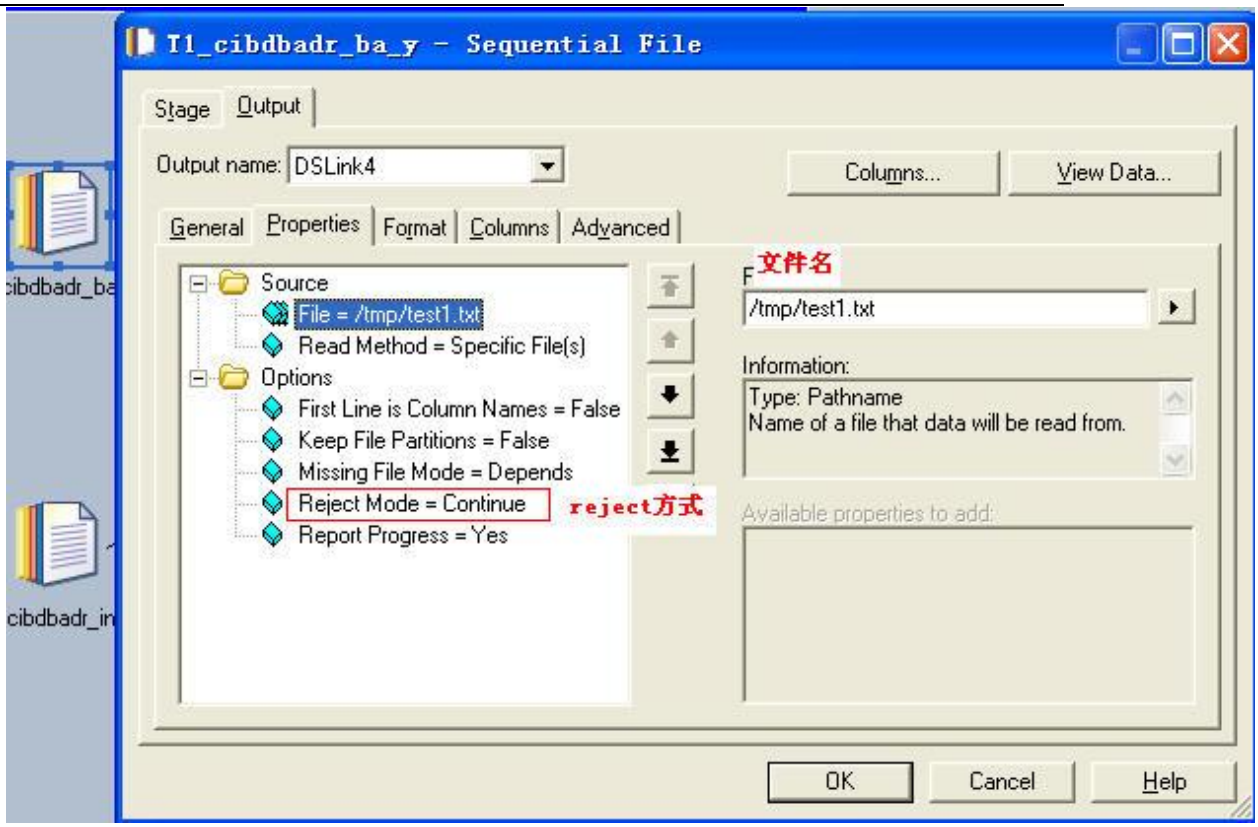
4.1.1. Sequential file

功能特点：适用于一般顺序文件（定长或不定长），可识别文本文件或IBM大机ebcdic文件。

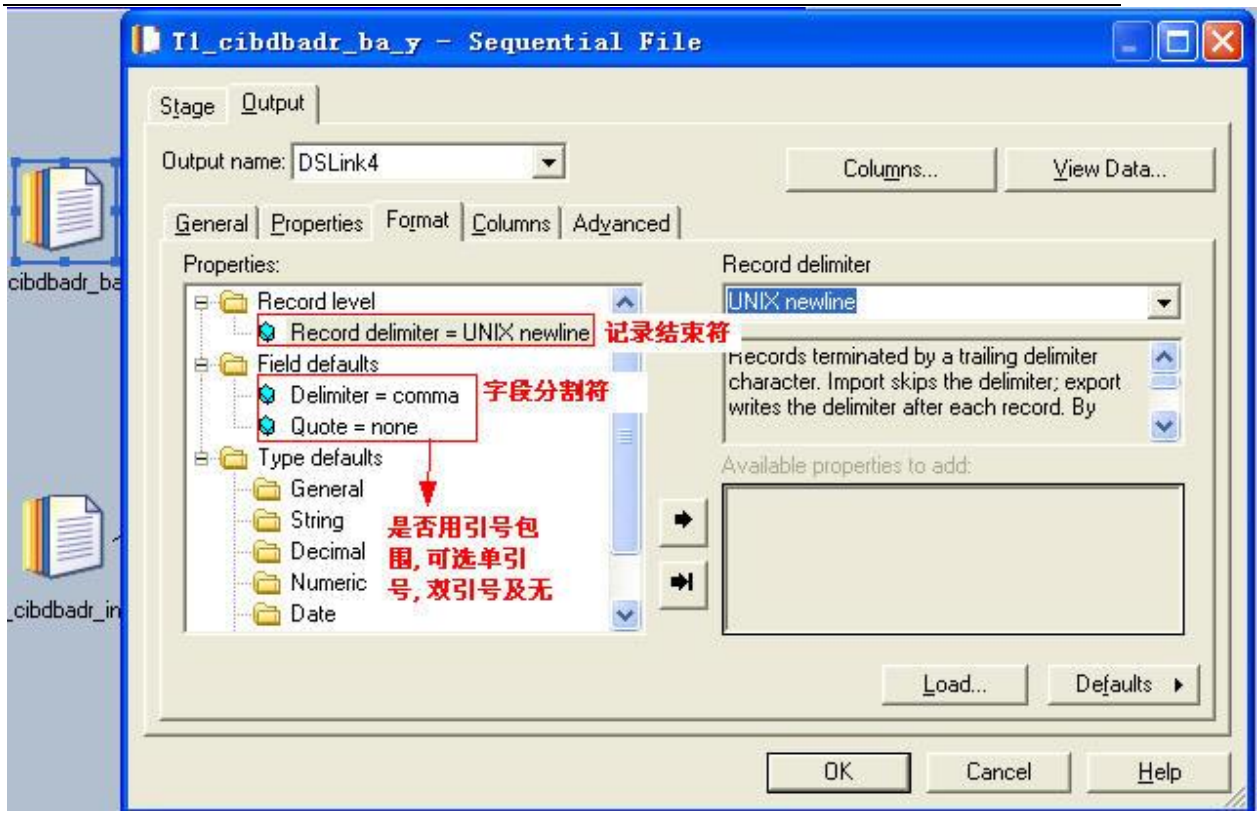
使用要点：

- 按照命名规范命名
- 点住文件，双击鼠标，在 general 说明此文件内容，格式，存储目录等

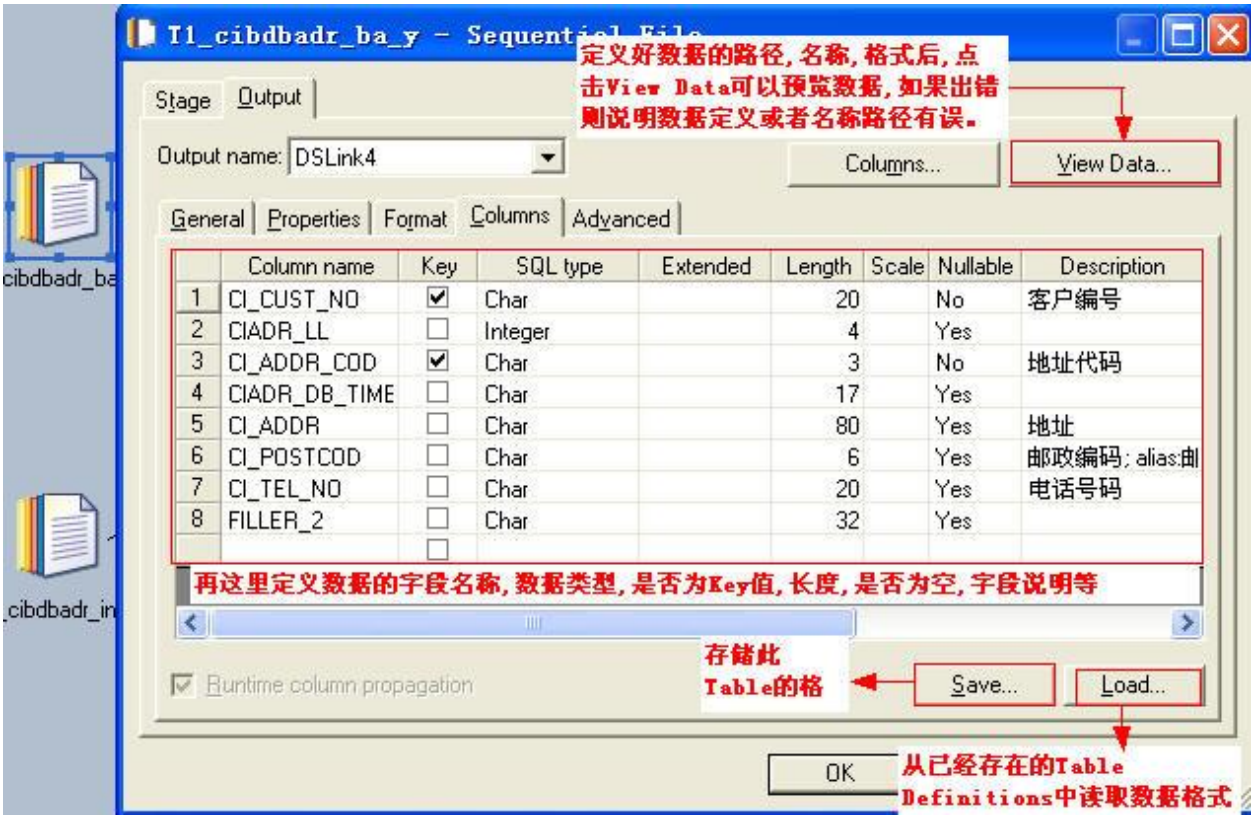
- 修改文件属性，文件名称，reject 方式等到



- 修改文件格式，比如记录结束符是什么，字段分隔符，字符串是用什么区别等


T1_cibdbadr_ba_y - Sequential File
Stage
Output
Output name: DSLink4
Columns...
View Data...
General
Properties
Format
Columns
Advanced
Properties:
Record level
Record delimiter = UNIX newline
记录结束符
Field defaults
Delimiter = comma
字段分割符
Quote = none
Type defaults
General
String
Decimal
Numeric
Date
是否用引号包围,可选单引号,双引号及无
Record delimiter
UNIX newline
Records terminated by a trailing delimiter character. Import skips the delimiter; export writes the delimiter after each record. By
Available properties to add:
Load...
Defaults
OK
Cancel
Help

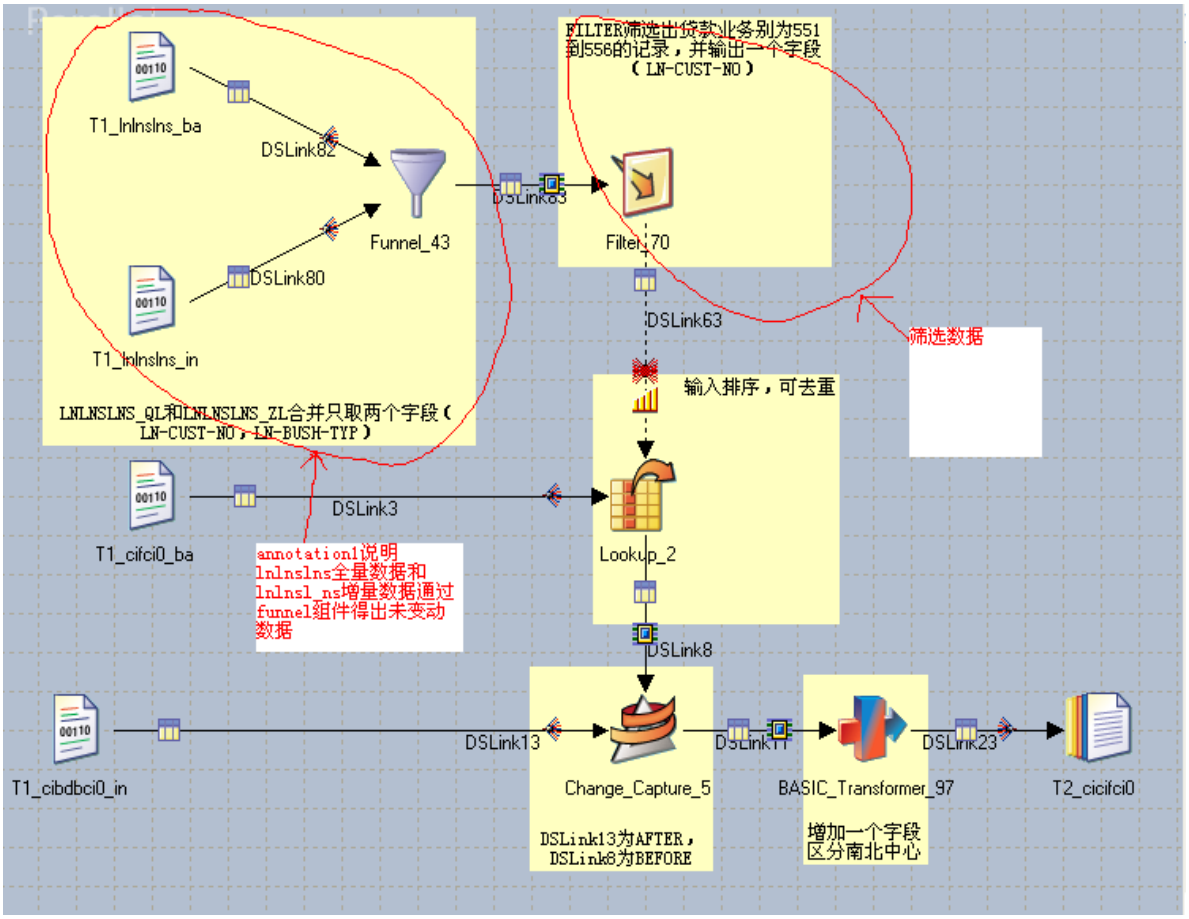
- 输入此文件字段内容

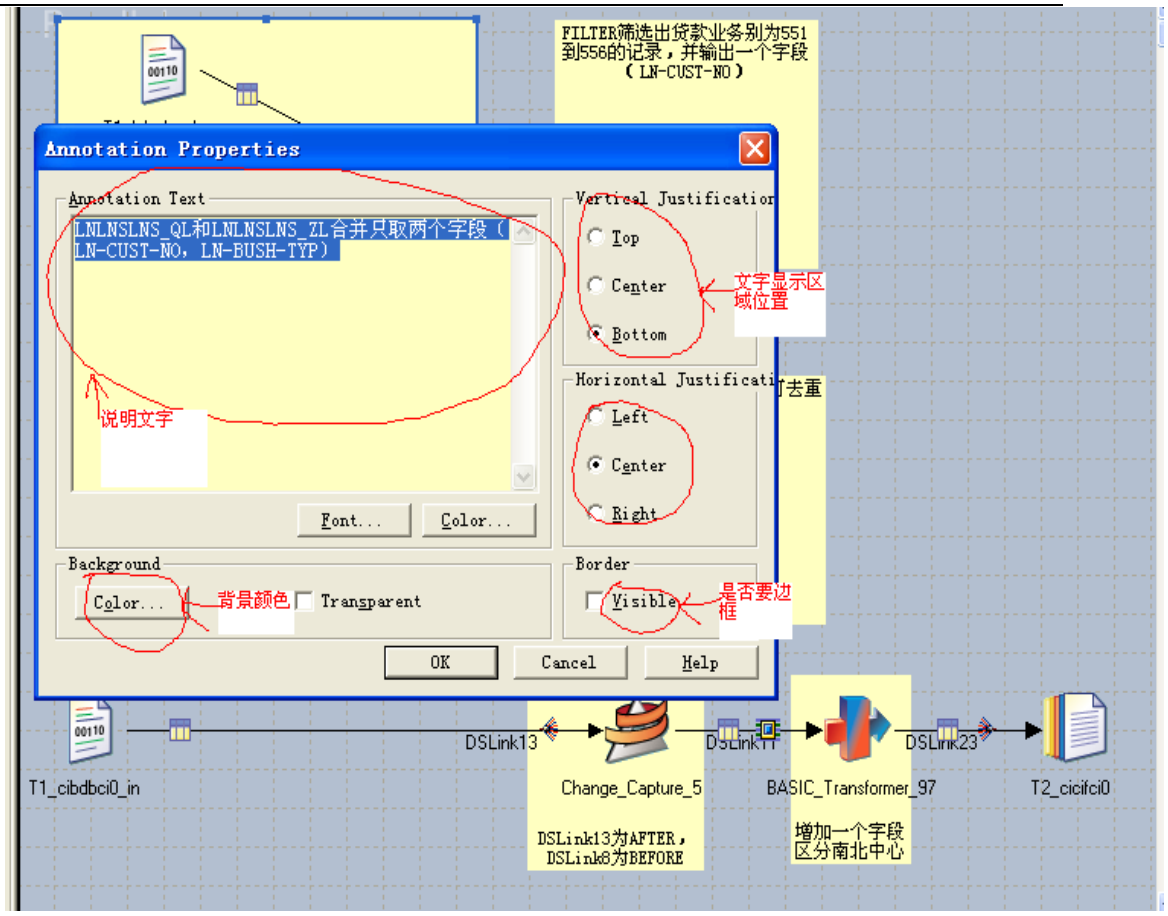

T1_cibdbadr_ba_y - Sequential File
定义好数据的路径,名称,格式后,点击View Data可以预览数据,如果出错则说明数据定义或者名称路径有误。
Stage
Output
Output name: DSLink4
Columns...
View Data...
General
Properties
Format
Columns
Advanced
Column name | Key | SQL type | Extended | Length | Scale | Nullable | Description
1 CI_CUST_NO | Char | 20 | No | 客户编号
2 CIADR_LL | Integer | 4 | Yes
3 CI_ADDR_COD | Char | 3 | No | 地址代码
4 CIADR_DB_TIME | Char | 17 | Yes
5 CI_ADDR | Char | 80 | Yes | 地址
6 CI_POSTCOD | Char | 6 | Yes | 邮政编码; alias:邮
7 CI_TEL_NO | Char | 20 | Yes | 电话号码
8 FILLER_2 | Char | 32 | Yes
再这里定义数据的字段名称,数据类型,是否为Key值,长度,是否为空,字段说明等
Runtime column propagation
存储此Table的格
Save...
Load...
OK
从已经存在的Table Definitions中读取数据格式

4.1.2. Annotation

功能特点：一般用于注释，可利用其背景颜色在job中分颜色区别不同功能块

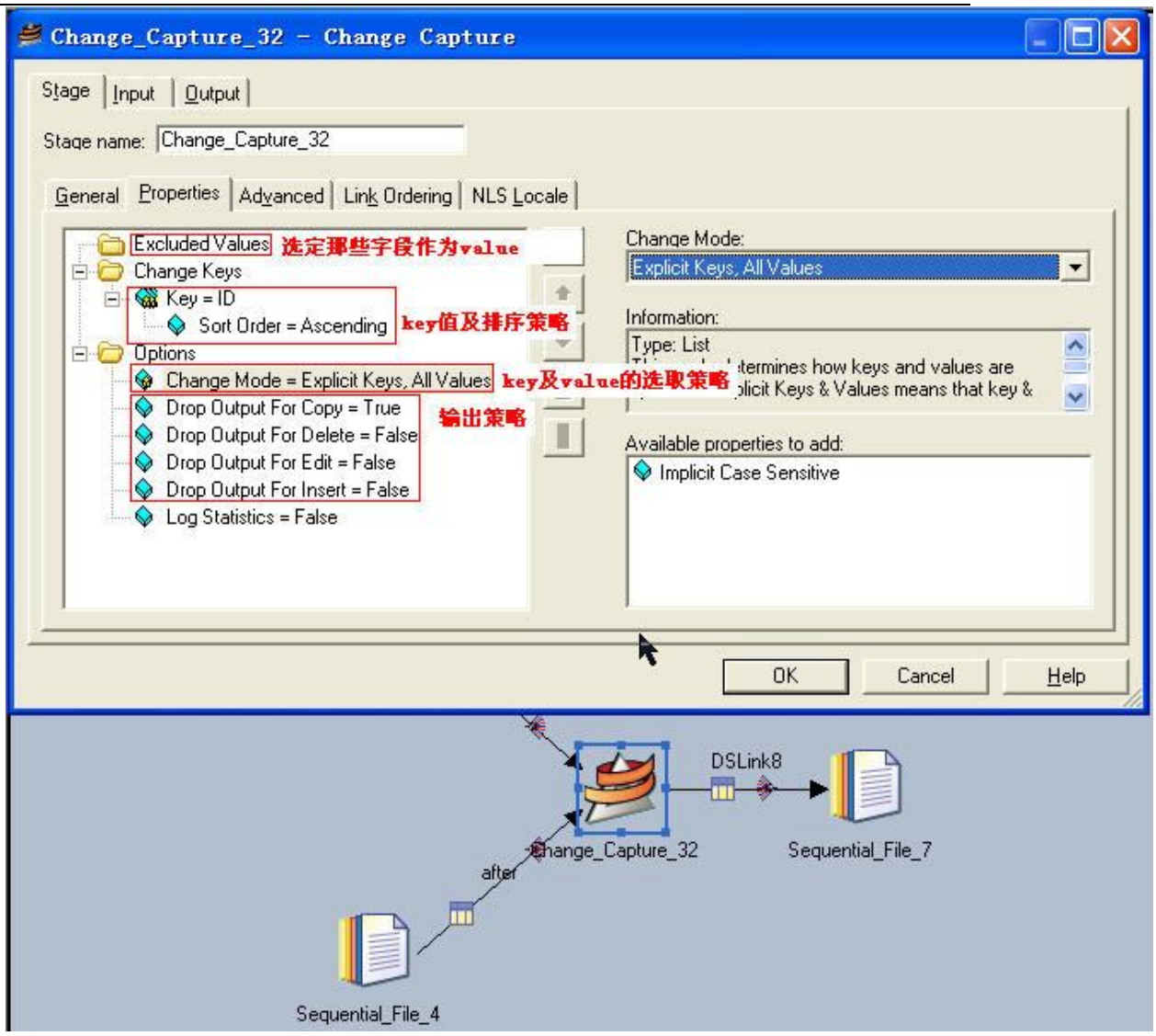
使用要点：





4.1.3. Change Capture Stage

- 功能特点：Change Capture Stage 有两个输入，分别标记为 before link 及 after link。输出的数据表示 before link 和 after link 的区别，我们称作 change set。Change Capture Stage 可以和 Change Apply Stage 配合使用来计算 after set。



- key 及 value 的说明

  key值是比较的关键值，value是当key值相同是作进一步比较用的。

- change mode 选项说明：

  All keys，Explicit Values　　需要指定value，其余字段为key

  Explicit Keys&Values　　key及value都需要指定

  Explicit Keys，All Values　　需要指定key，其余的字段为value

- 输出策略说明

Drop Output For Copy	False：保留before及afte link中key值相同的行 True：删除before及afte link中key值相同的行
Drop Output For Delete	False：保留before link中有但是after link中没有的key值所在的行

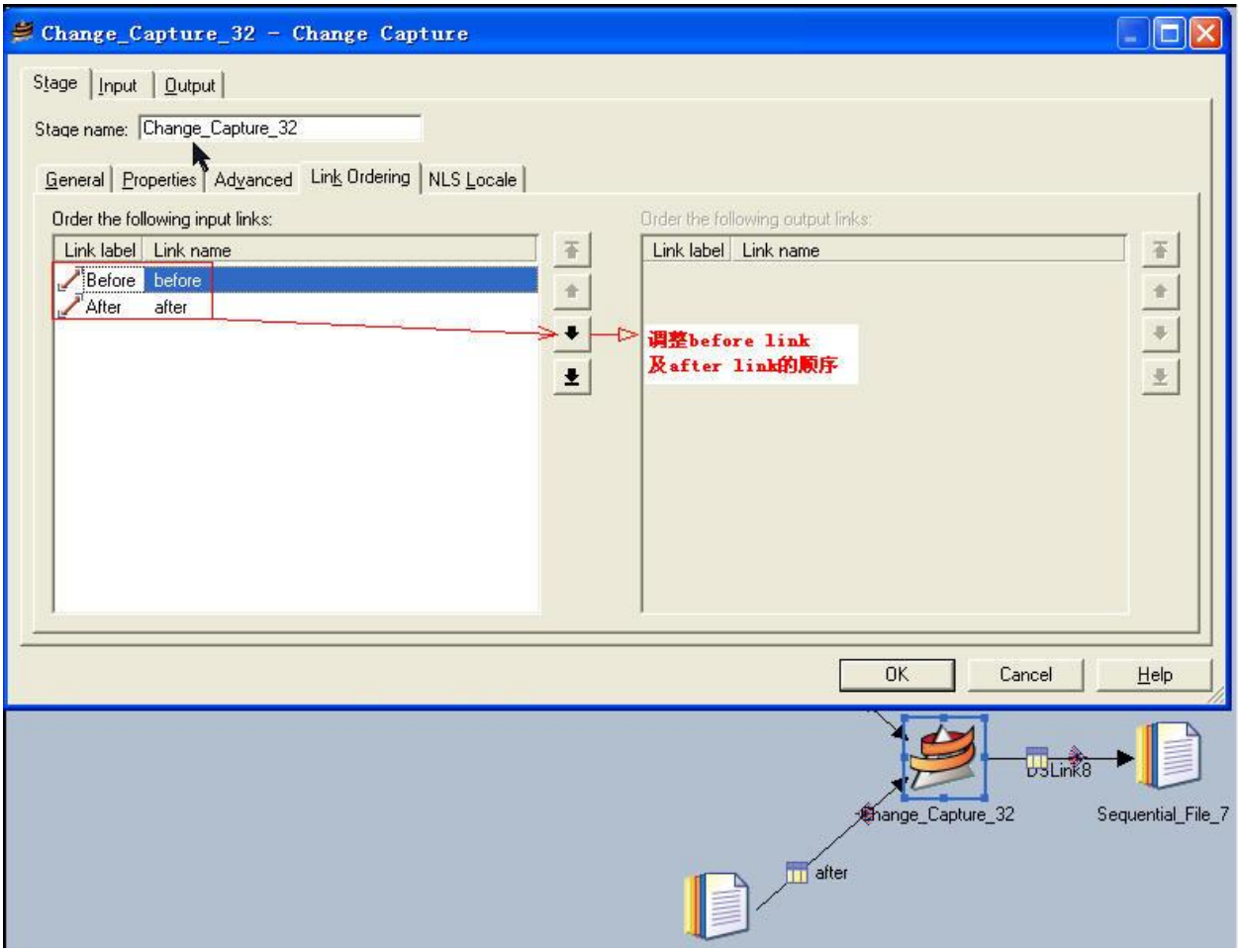
True：删除before link中有但是afte link中没有的key值所在的行

Drop Output For Edit　　False：保留key值相同,value不同的行

True：删除key值相同,value不同的行

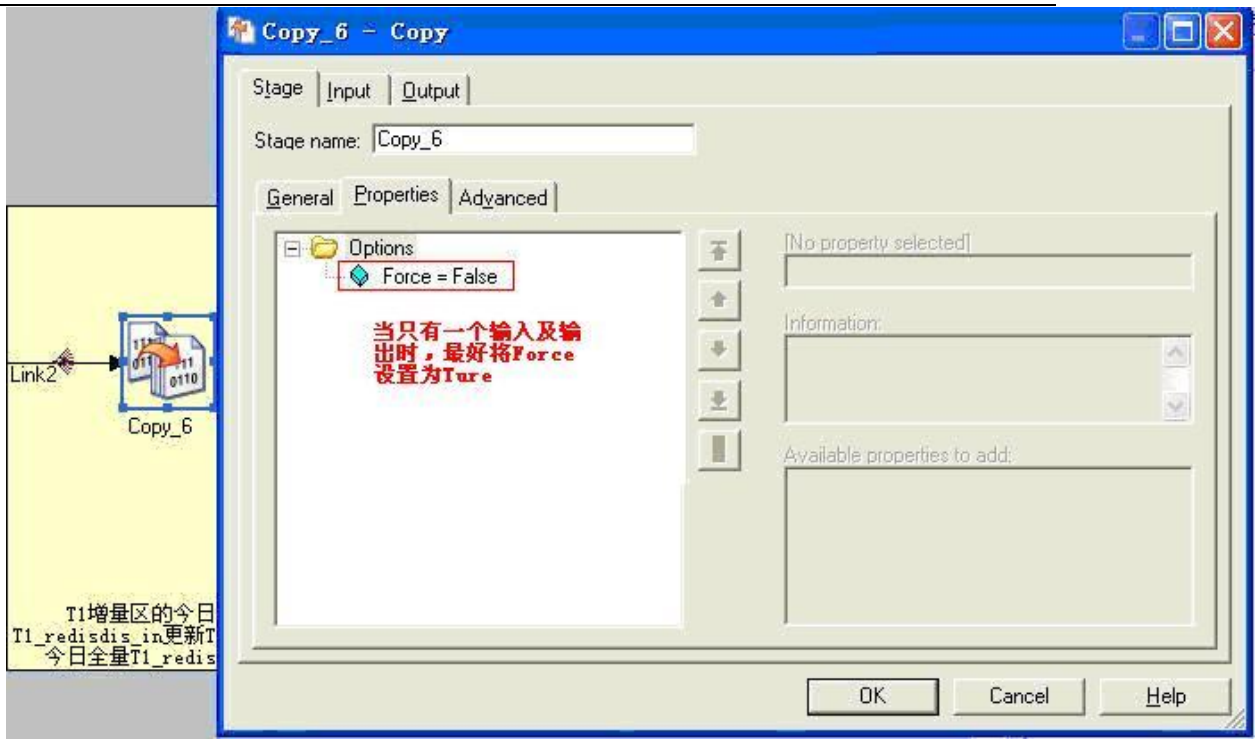
Drop Output For Insert　　False：保留before link中没有但afte link中有的key值所在的行

True：删除before link中没有但afte link中有的key值所在的行



4.1.4. Copy Stage

➢ 功能说明：Copy Stage 可以有一个输入，多个输出。它可以在输出时改变字段的顺序，但是不能改变字段类型。



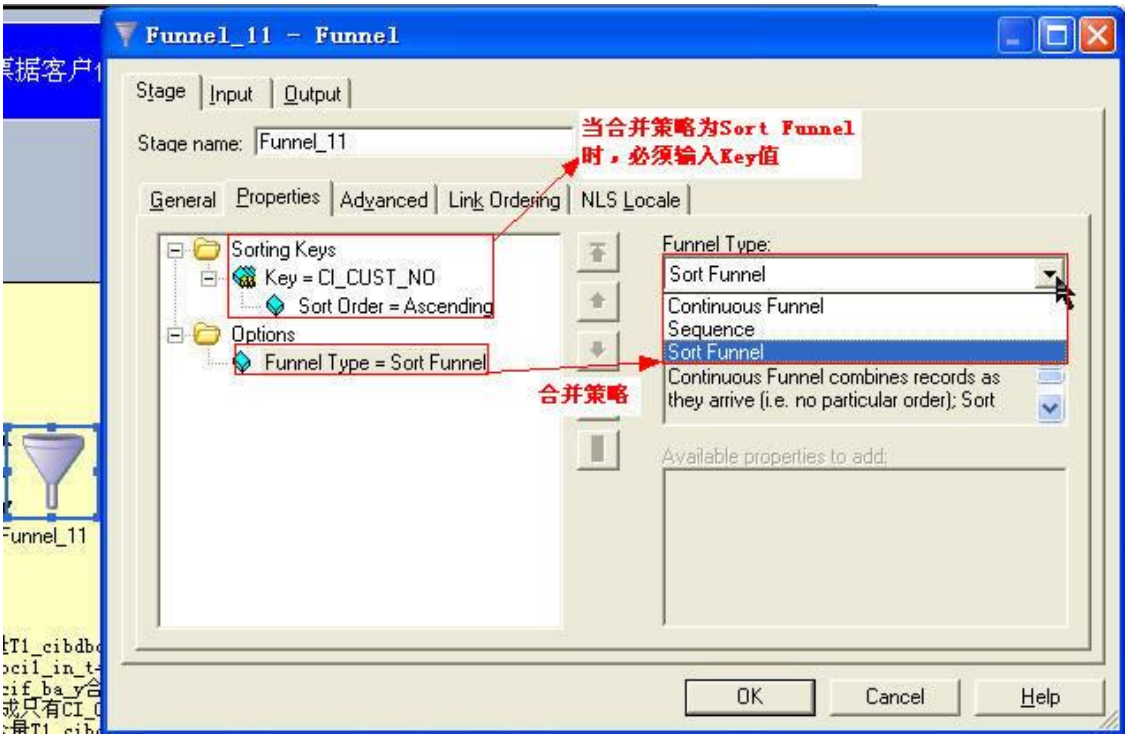
注意：当只有一个输入及一个输出时最好将Force设置为True，这样可以在Designer里看到运行结束，否则将无法标识运行结束，但不会影响运行结果数据。

4.1.5. Filter Stage

- 功能说明：Filter Stage 只有一个输入，可以有多个输出。根据不同的筛选条件，可以将数据输出到不同的 output link。

4.1.6. Funnel Stage

- 功能说明：将多个字段相同的数据文件合并为一个单独的文件输出



- 合并策略说明

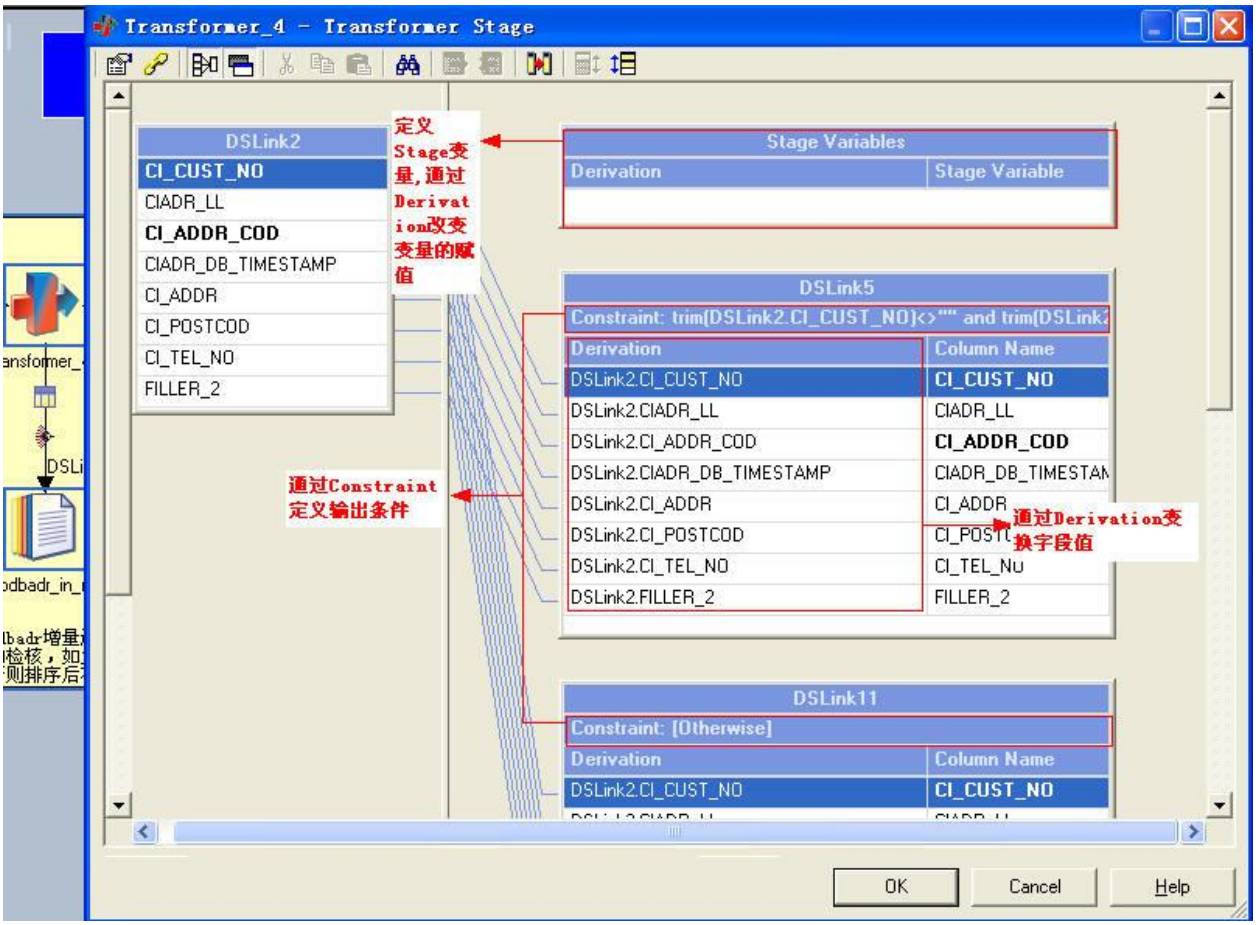
Continuous Funnel：从每一个input link中循环取一条记录

Sort Funnel：按照Key值排序合并输出

Sequence：先输出第一个input link的数据，输出完毕后再输出第二个input link的数据，依此类推，直到结束。(此时可以通过调整link Ordering调整输出顺序)

4.1.7. Tansformer Stage

- 功能说明：一个功能极为强大的 Stage。有一个 input link，多个 output link，可以将字段进行转换，也可以通过条件来指定数据输出到那个 output link。在开发过程中可以使用拖拽。



- Constraint 及 Derivation 的区别

  Constraint通过限定条件使符合条件的数据输出到这个output link。

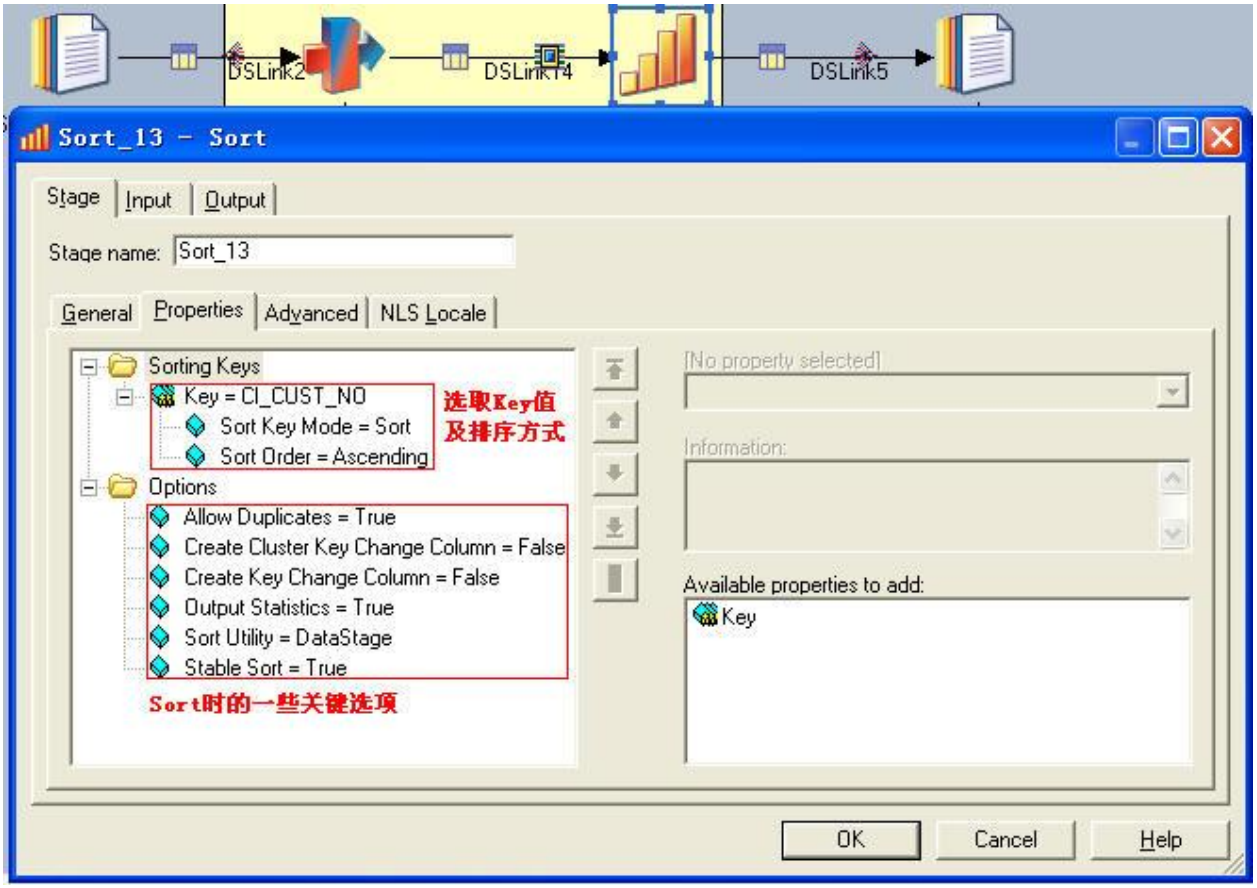
  Derivation通过定义表达式来转换字段值。

  在Constraint及Derivation中可以使用Job parameters及Stage Variables。

- 注意：Transformer Stage 功能强大，但在运行过程中是以牺牲速度为代价的。在只有简单的变换，拷贝等操作时，最好用 Modify Stage，Copy Stage，Filter Stage 等来替换 Transformer Stage。

4.1.8. Sort Stage

功能说明：只能有一个输入及一个输出，按照指定的Key值进行排列。可以选择升序还是降序，是否去除重复的数据等等。



- Option 具体说明

Allow Duplicates: 是否去除重复数据。为False时，只选取一条数据，当Stable Sort为True时，选取第一条数据。当Sort Unility为UNIX时此选项无效。

Sort Utility: 选择排序时执行应用程序，可以选择DataStage内建的命令或者Unix的Sort命令

Output Statistics: 是否输出排序统计信息到job日志

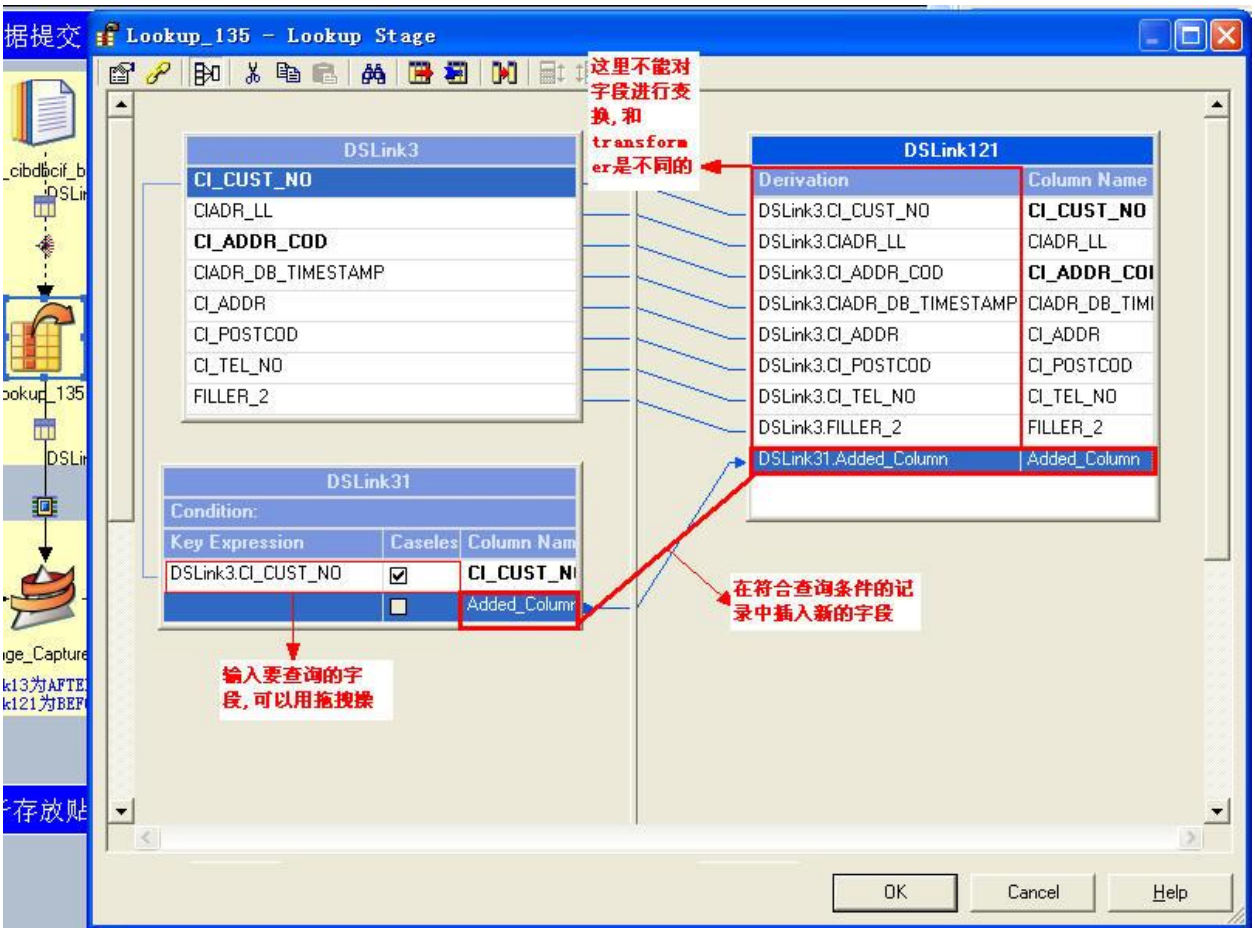
Stable Sort： 是否对数据进行二次整理

Create Cluster Key Change Column：是否为每条记录创建一个新的字段：clusterKeyChange。当Sort Key Mode为Don't Sort(Previously Sorted) 或 Don't Sort (Previously Grouped)时，对于第一条记录该字段被设置为1，其余的记录设置为0。

Create Key Change Column：是否为每一条记录创建一个新的字段KeyChange。

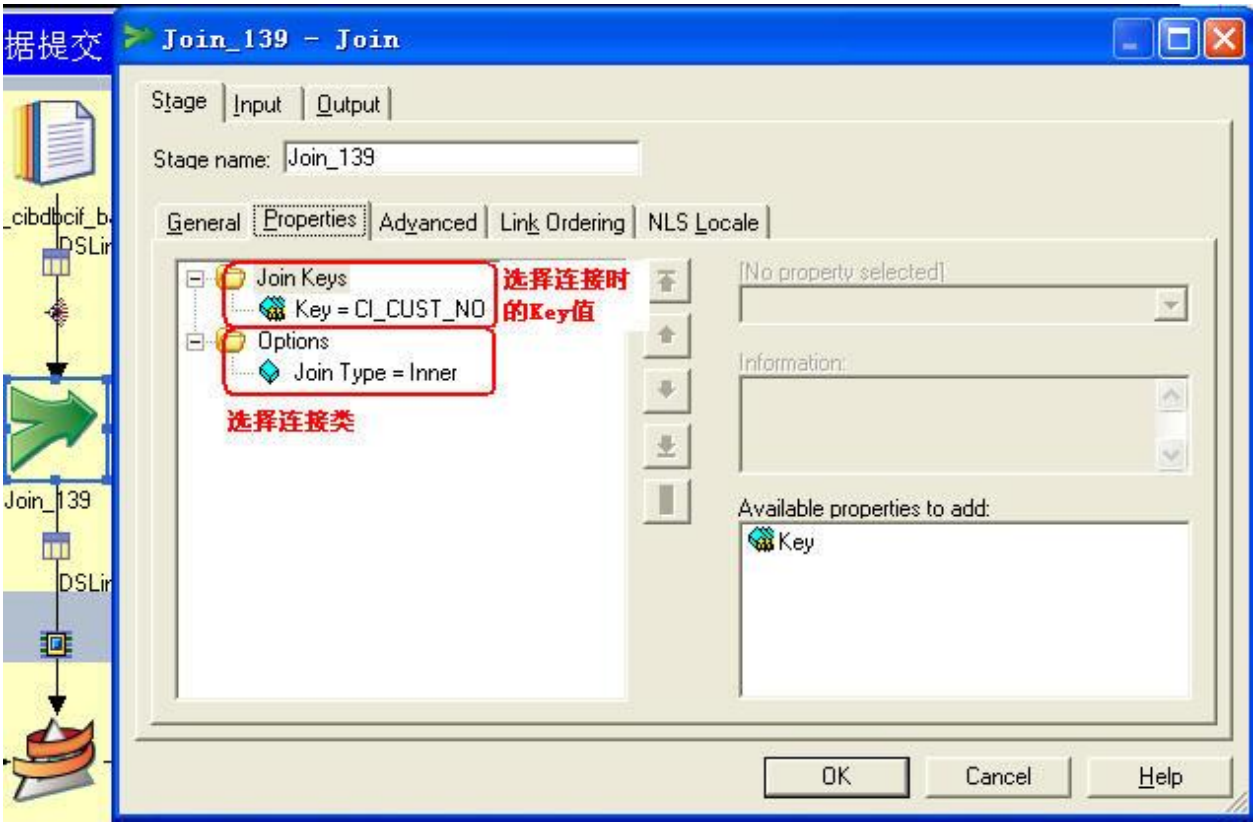
4.1.9. LookUp Stage

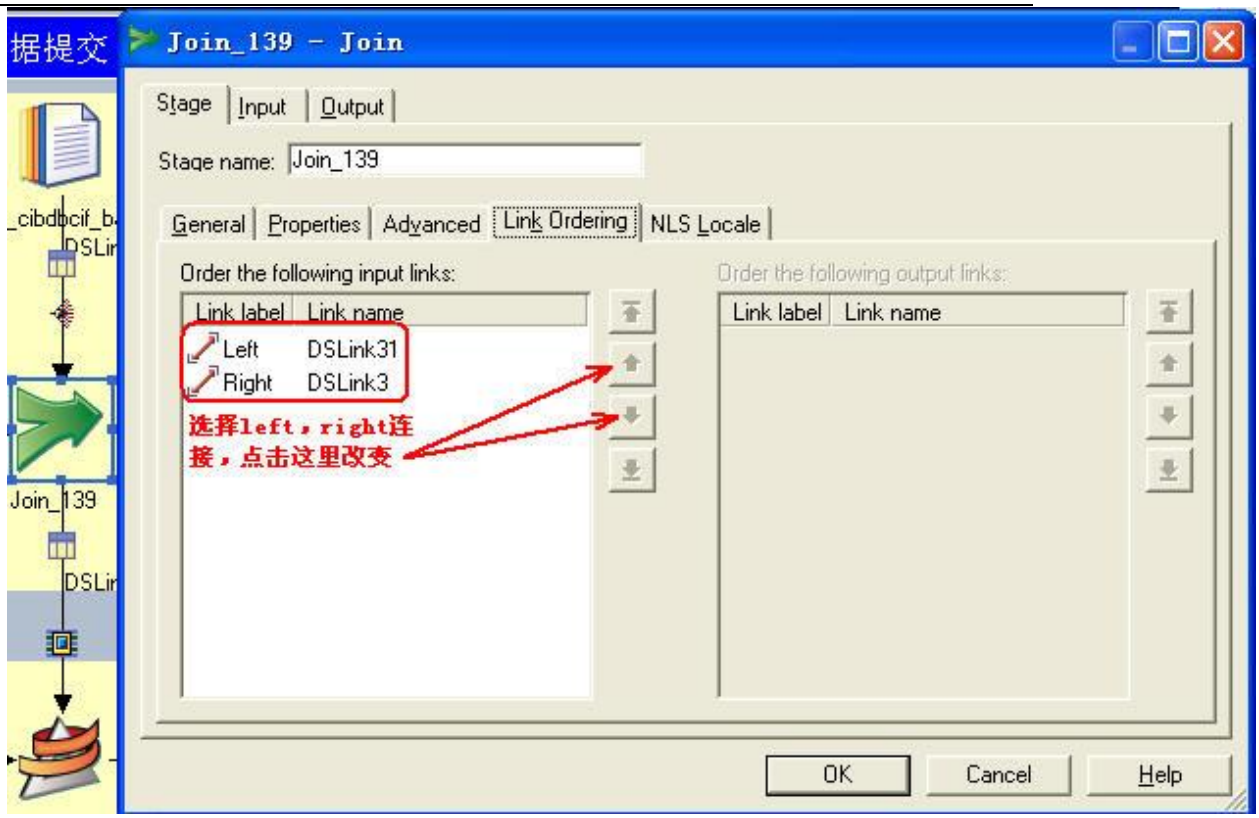
- 功能说明：LookUp Stage 把数据读入内存执行查询操作，将匹配的字段输出，或者在在符 合条件的记录中修改或加入新的字段。



4.1.10. Join Stage

- 功能说明：将多个表连接后输出



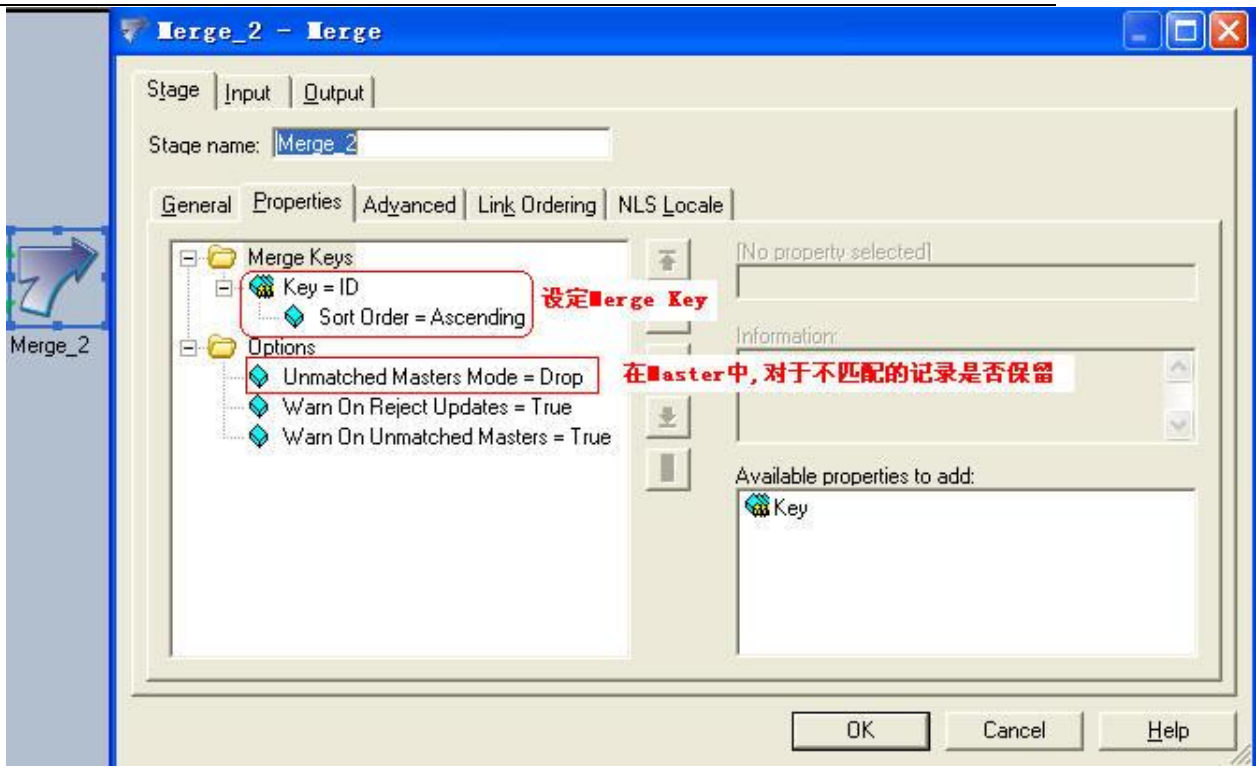


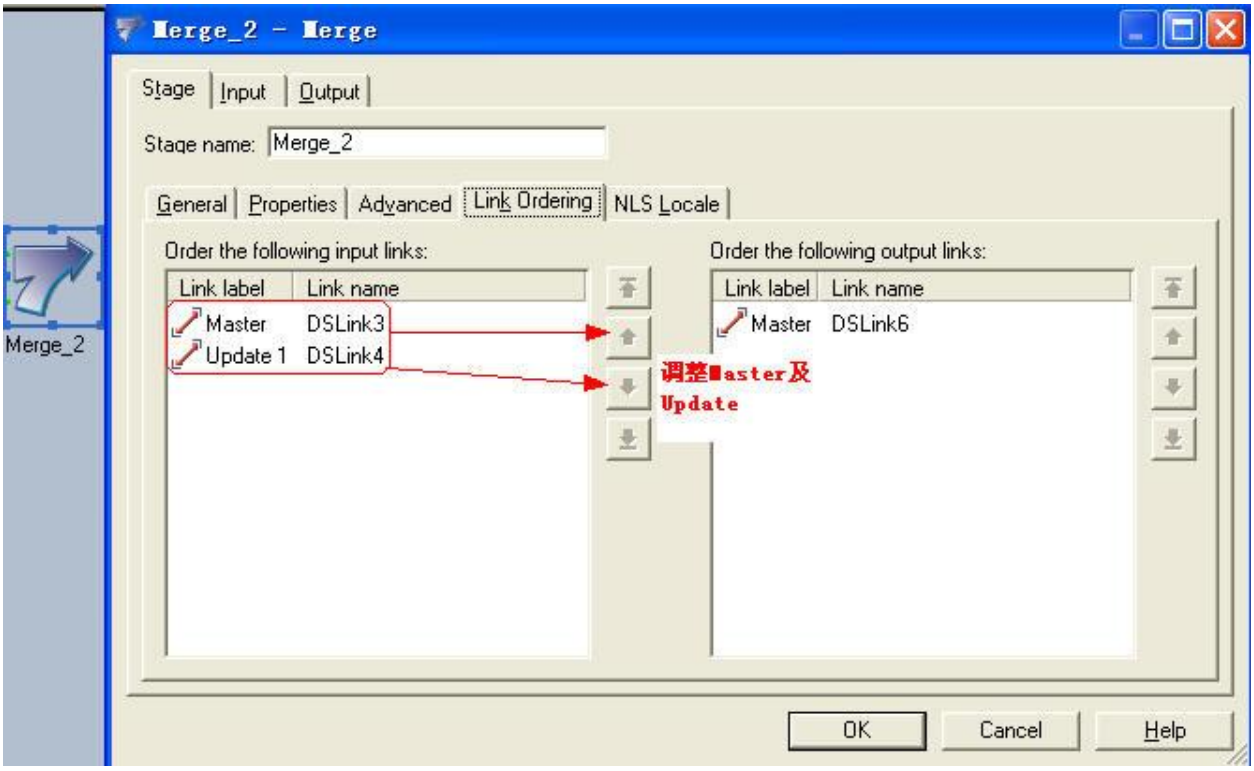
4.1.11. LookUp Stage 和 Join Stage 的区别

LookUp Stage将数据读入到内存中，所以效率很高，但是占用了较多的物理内存。所以当reference data比较小的时候，我们推荐用LookUp Stage；当reference data比较大的时候，我们推荐用Join Stage。

4.1.12. Merge Stage

- 功能说明：将 Merge Key 值相同的记录合并。将其中的一个输入设定为 Master，其余的为 Update。把 Update 中 Merge Key 相同的记录合并入 Master。

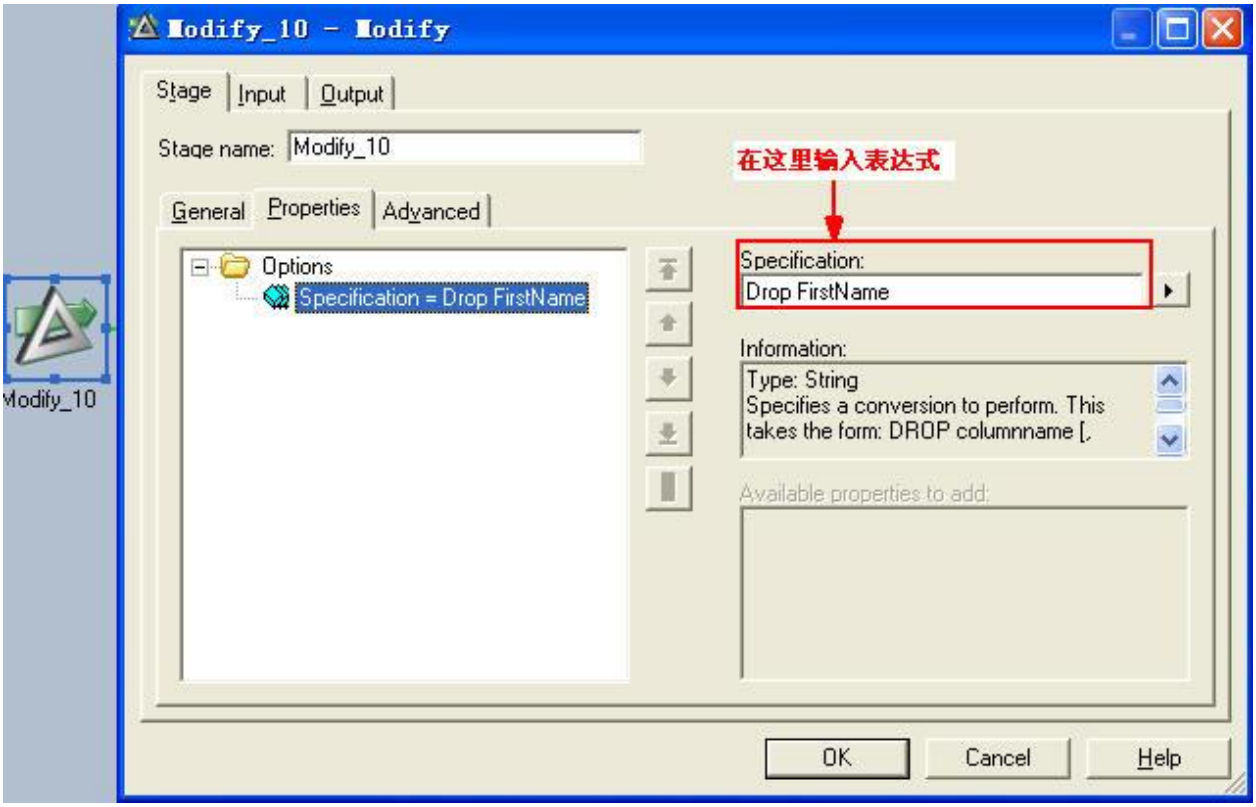




4.1.13. Modify Stage

➢ 功能说明：Modify stage 只能有一个输入及一个输出，它可以修改表结构：

删除，保留一个字段；给字段改名；修改字段的数据类型。



- Specification的具体用法：

删除一个字段： DROP columnname [, columnname]

保留一个字段： KEEP columnname [, columnname]

改变字段： new_columnname [:new_type] = [explicit_conversion_function]old_columnname

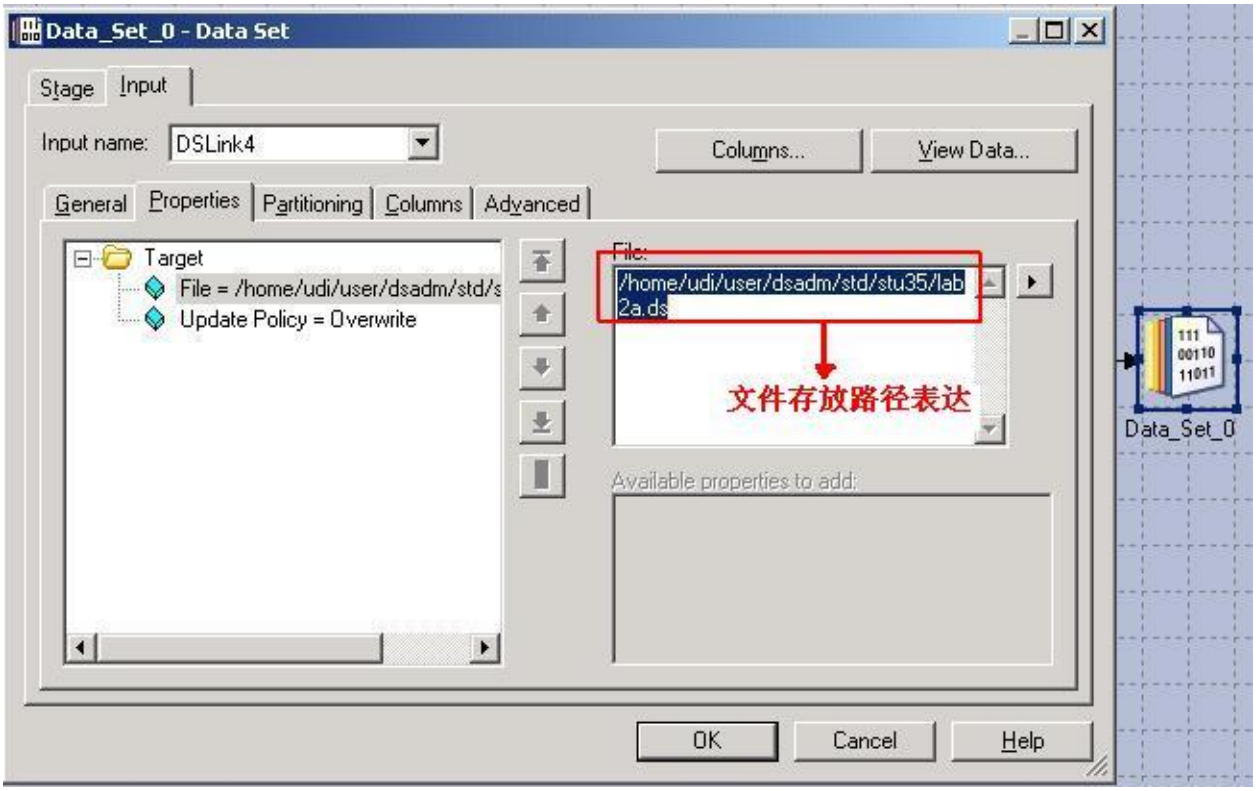
可用的explicit_conversion_function请参看《Parallel Job Developer's Guide》page 28-7

4.1.14. Data Set Stage

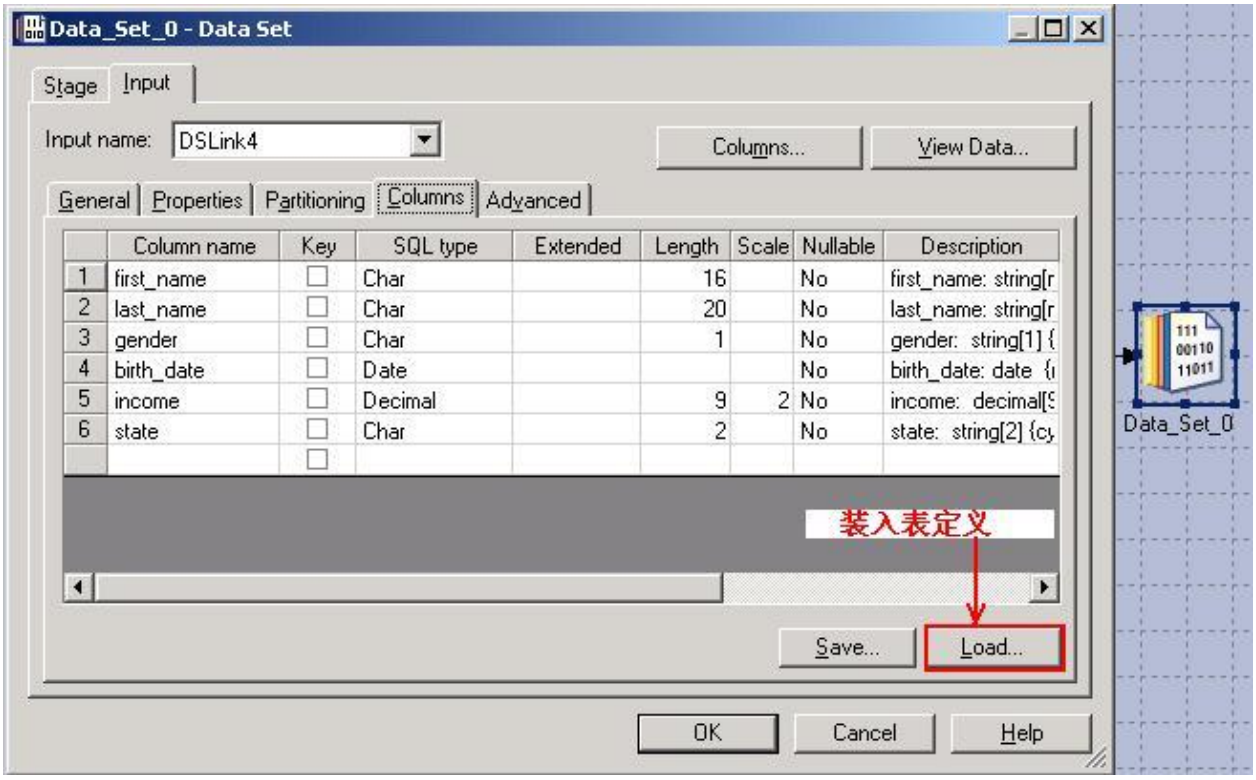
- Stage类型：File stage
- 功能说明：从data set文件中读取数据或者写数据到data set文件中，一个Date Set Stage只能有一个输入连接（input link）或者一个输出连接（output link）。
- 具体用法：包括Stage Page，Inputs Page，Outputs Page
  - ✓ Stage Page通常描述了stage的一般信息，诸如名称等；
  - ✓ Inputs Page描述了即要写入信息的data set文件的详细信息；

  主要是Properties和Column的定义

Properties中配置了文件的存放路径和更新策略



Column详细定义文件中的各个字段

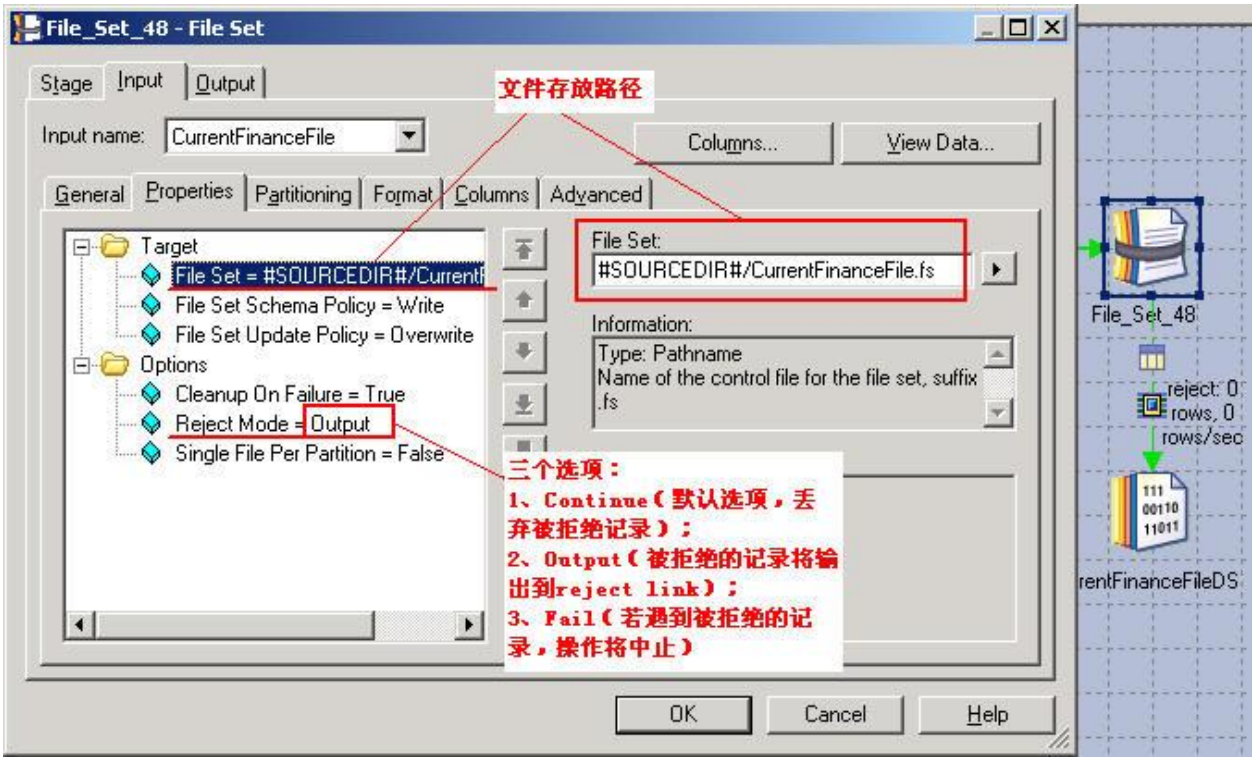


✓ Outputs Page描述了读取信息的data set文件的详细信息；操作过程与Inputs Page类似。

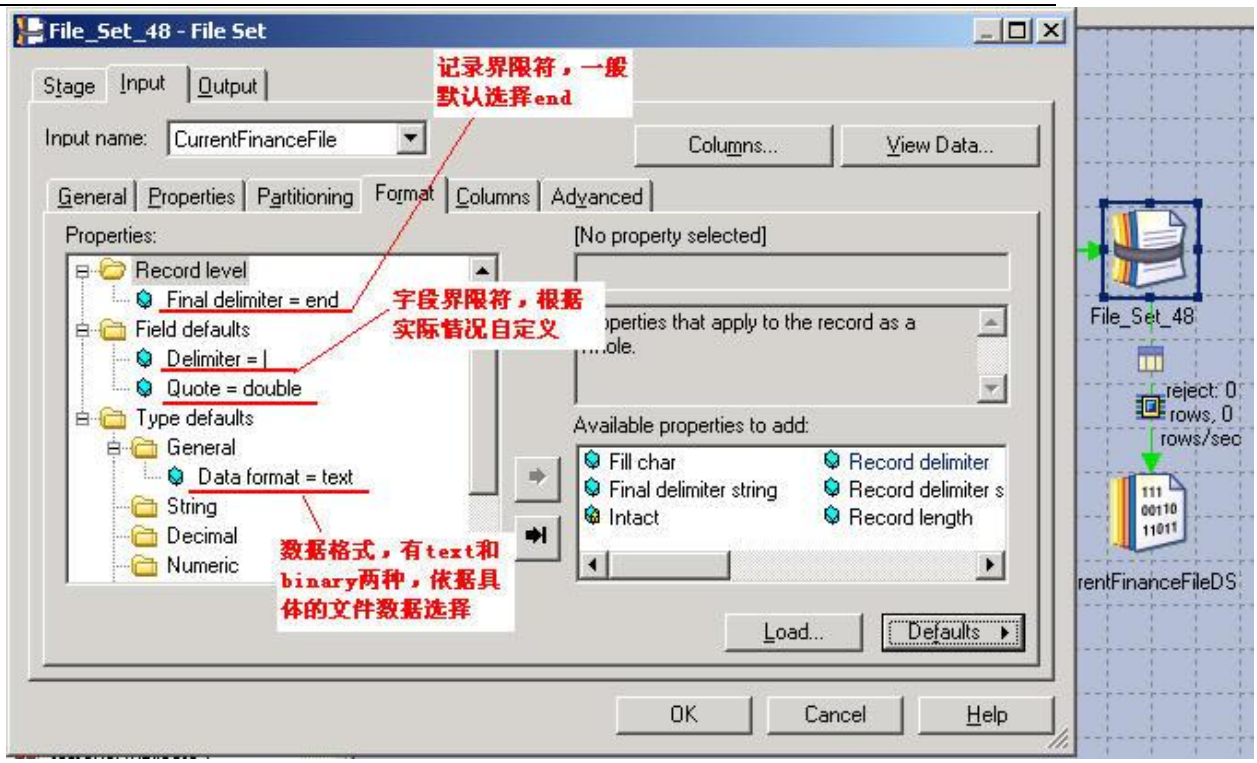
4.1.15. File Set Stage

- Stage类型：File stage
- 功能说明：从file set文件中读取数据或者写数据到file set文件中，一个File Set Stage只能有一个输入连接（input link）、一个输出连接（output link）和一个拒绝连接（rejects link）。并且只能在并行模式下执行。
- 具体用法：
  - ✓ Stage Page：对Stage的基本定义
  - ✓ Inputs Page：主要是Properties和Format的配置

Properties的配置：定义文件的存取路径及其他读写的相关的参数。特别要说明的是Options下的Reject Mode的选择，当stage有reject link的时候，必须选择Output；没有reject link时，可选择其他两个选项。



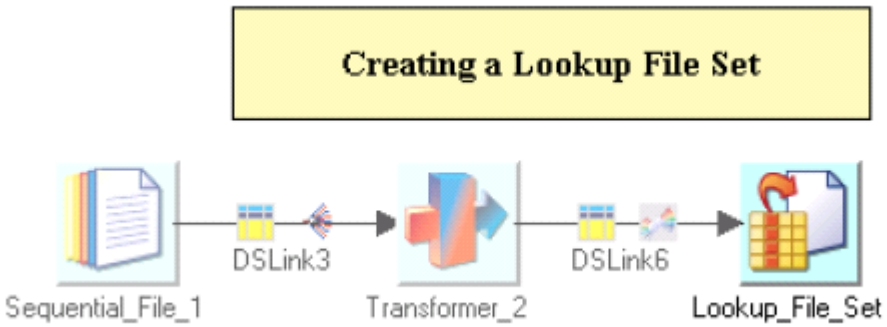
Format的配置：定义了数据写到文件中的格式

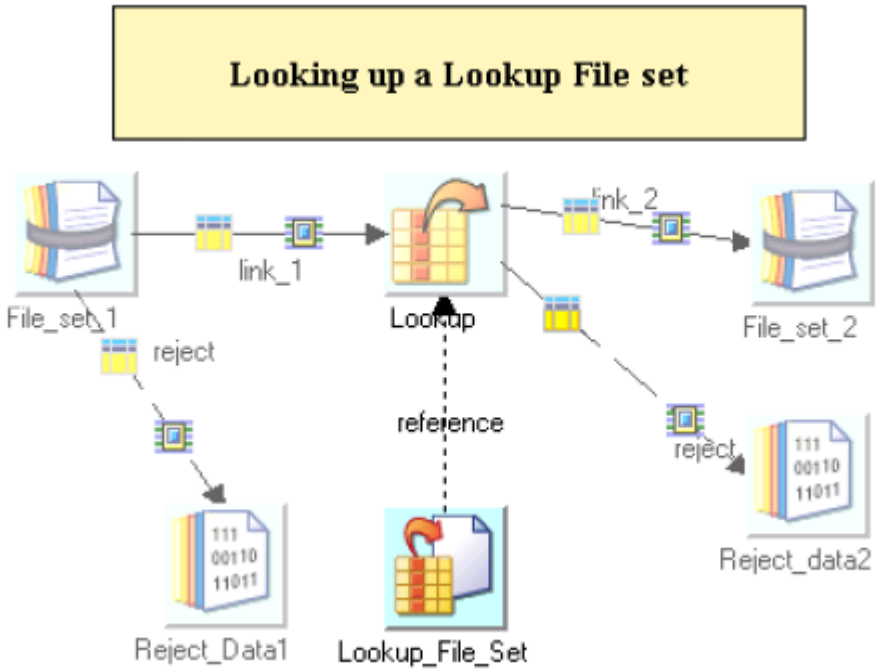


✓ Outputs Page：对stage输出的数据字段的描述，另外，reject link的输出系统将默认，不需要用户自己定义字段。

4.1.16. Lookup File Set Stage

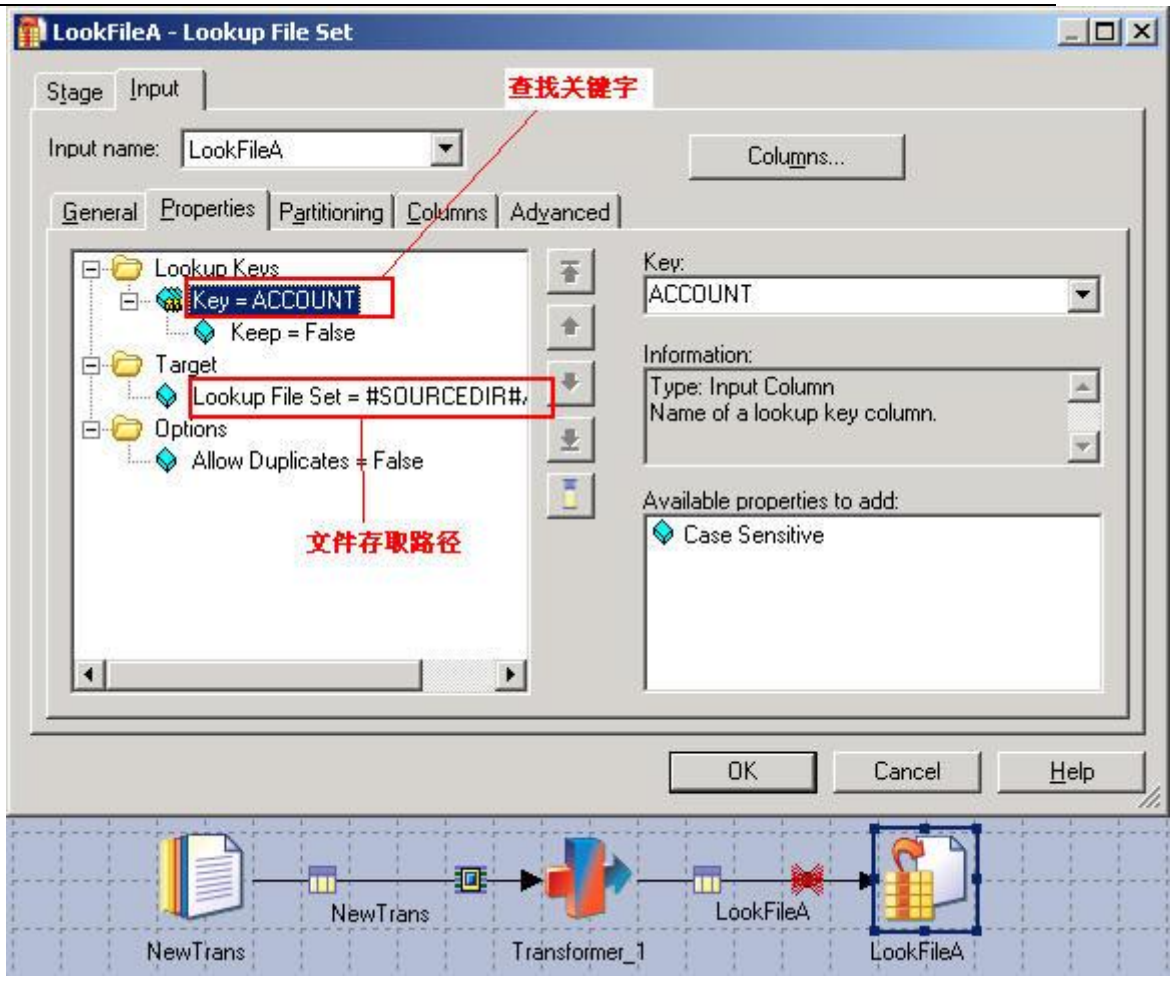
- Stage类型：File stage
- 功能说明： 为执行查找操作而创建的参照文件。
  作为查找的参照数据，通常在参照数据比较大量或者重复使用率较高的情况下，将参照数据生成专门的Lookup File Set文件，以便提高查找的效率



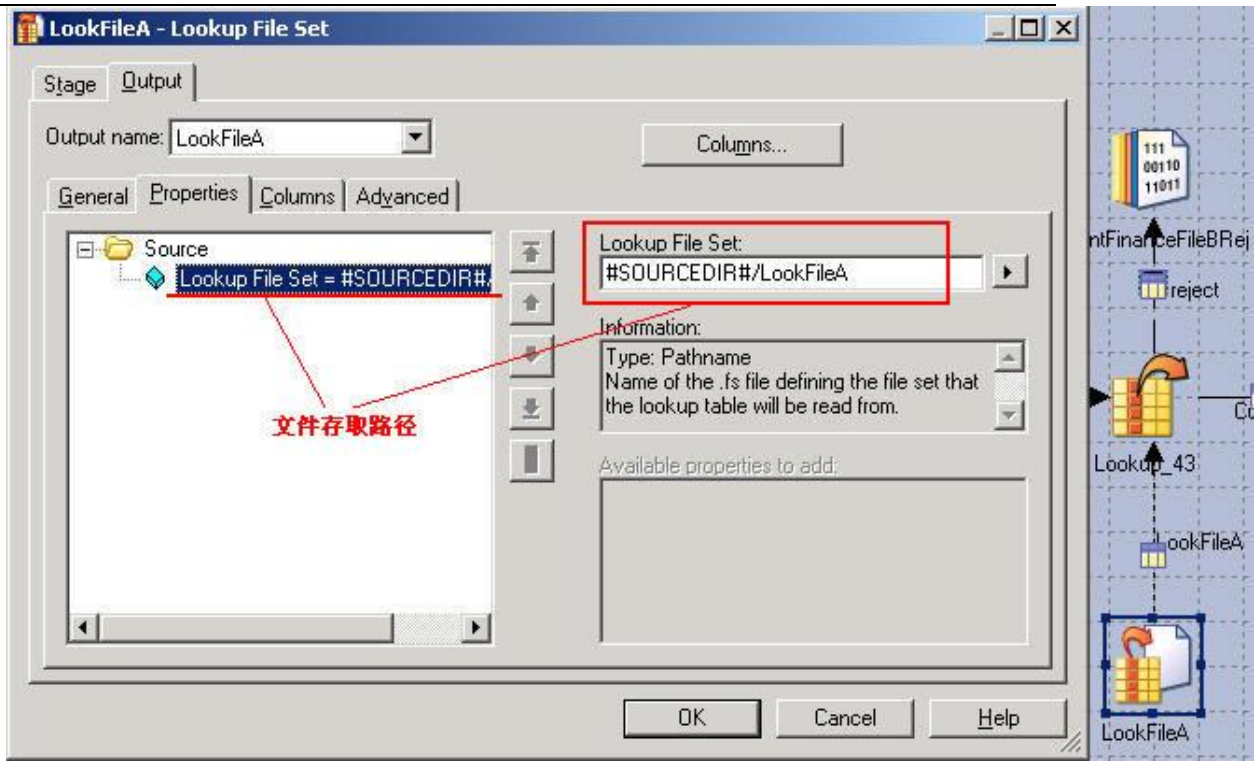


- 具体用法：
  - ✓ Stage Page
  - ✓ Inputs Page：主要定义了查找关键字和存放路径等主要信息

创建一个Lookup File Set文件：

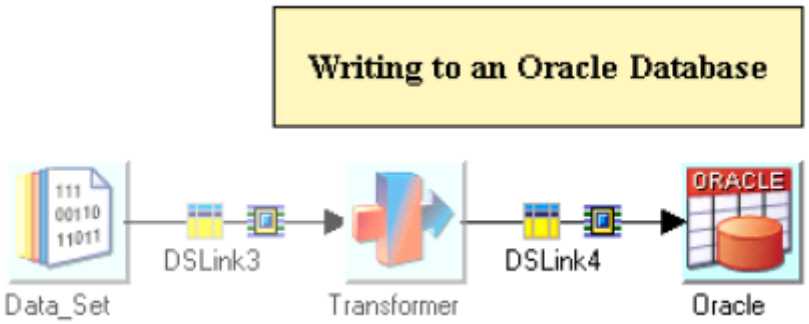


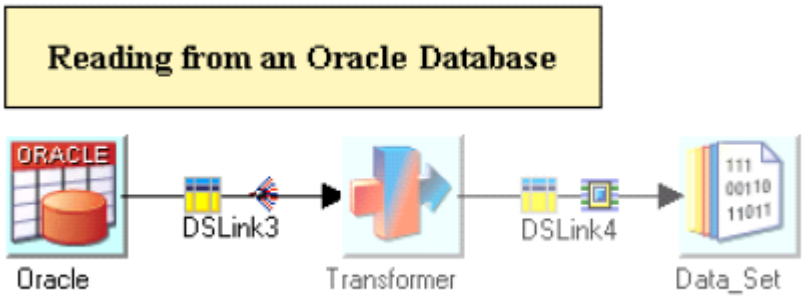
✓ Outputs Page：当作为参照数据进行查找操作时，因为文件是已经生成好的，所以在这里不需要再做详细的定义，只要引入即可。

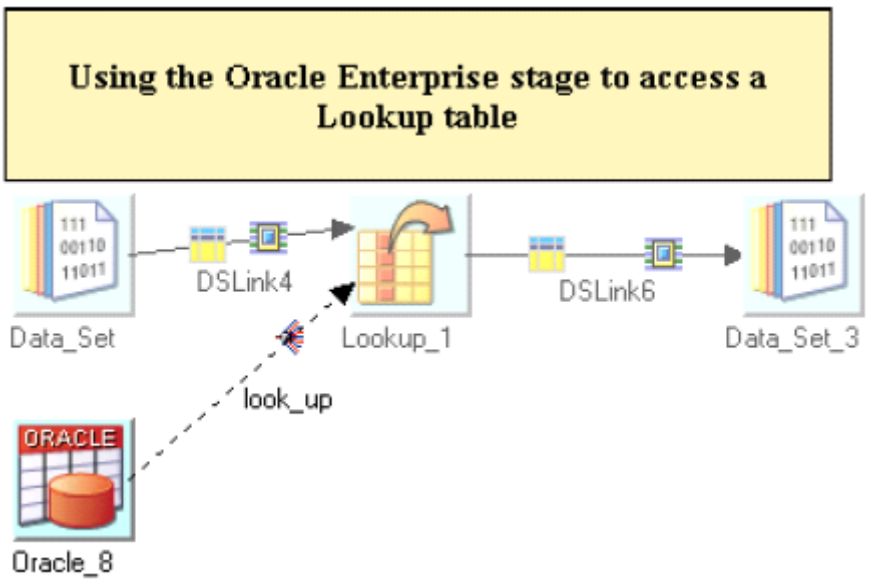


4.1.17. Oracle Enterprise Stage

- Stage类型：Database Stage
- 功能说明：从Oracle数据库中读取数据或者写数据到Oracle数据库中。
  通常完成的操作：
  - ✓ 使用INSERT或UPDATWE命令更新数据库表
  - ✓ 装入数据库表
  - ✓ 读取数据库表
  - ✓ 从数据库表中删除行
  - ✓ 在库表中直接执行查询操作
  - ✓ 将库表装入内存，然后执行查询操作

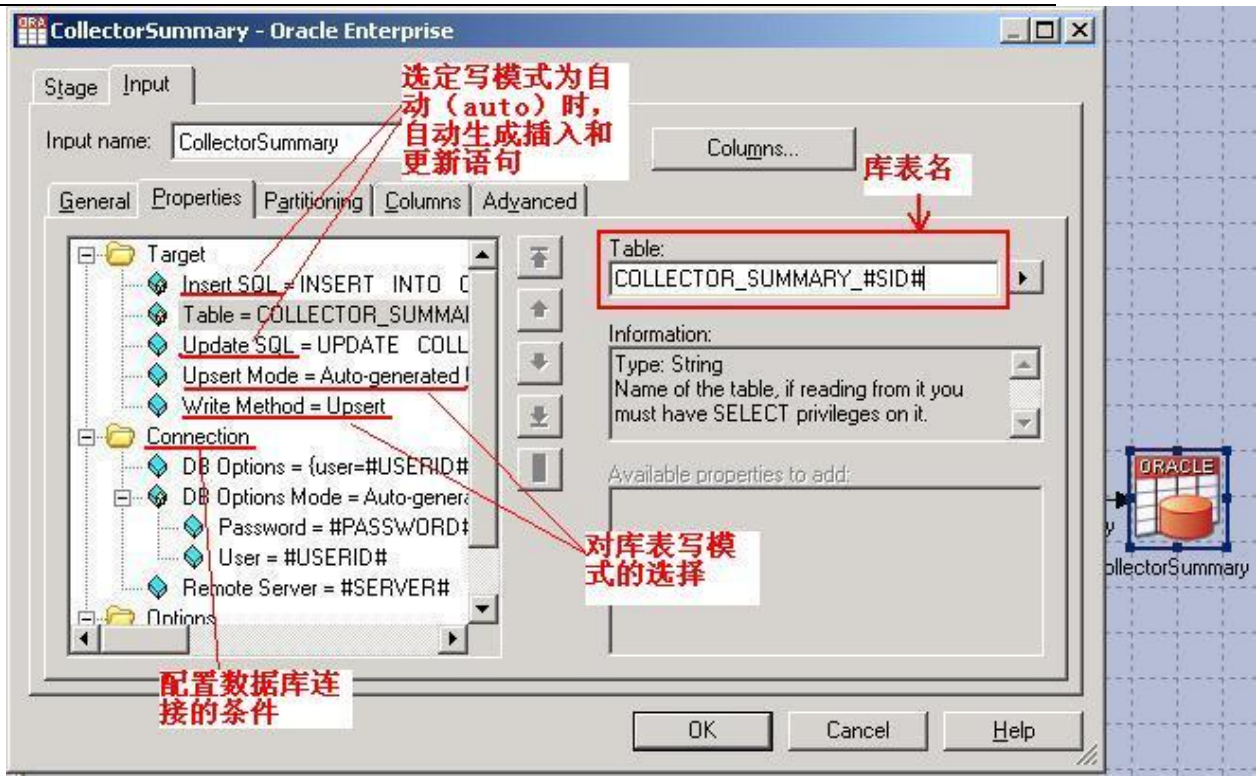






- 具体用法：
  - ✓ Inputs Page

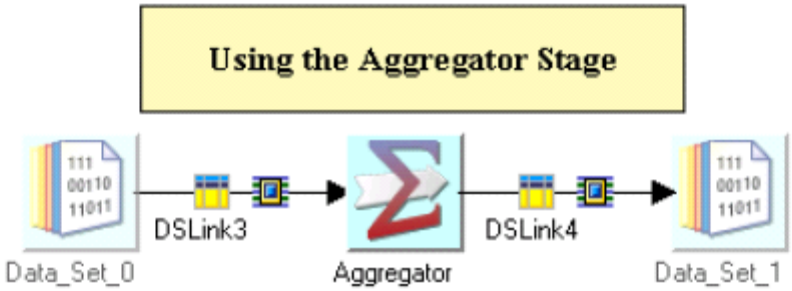
  向数据库中写数据，关键是对Properties的配置



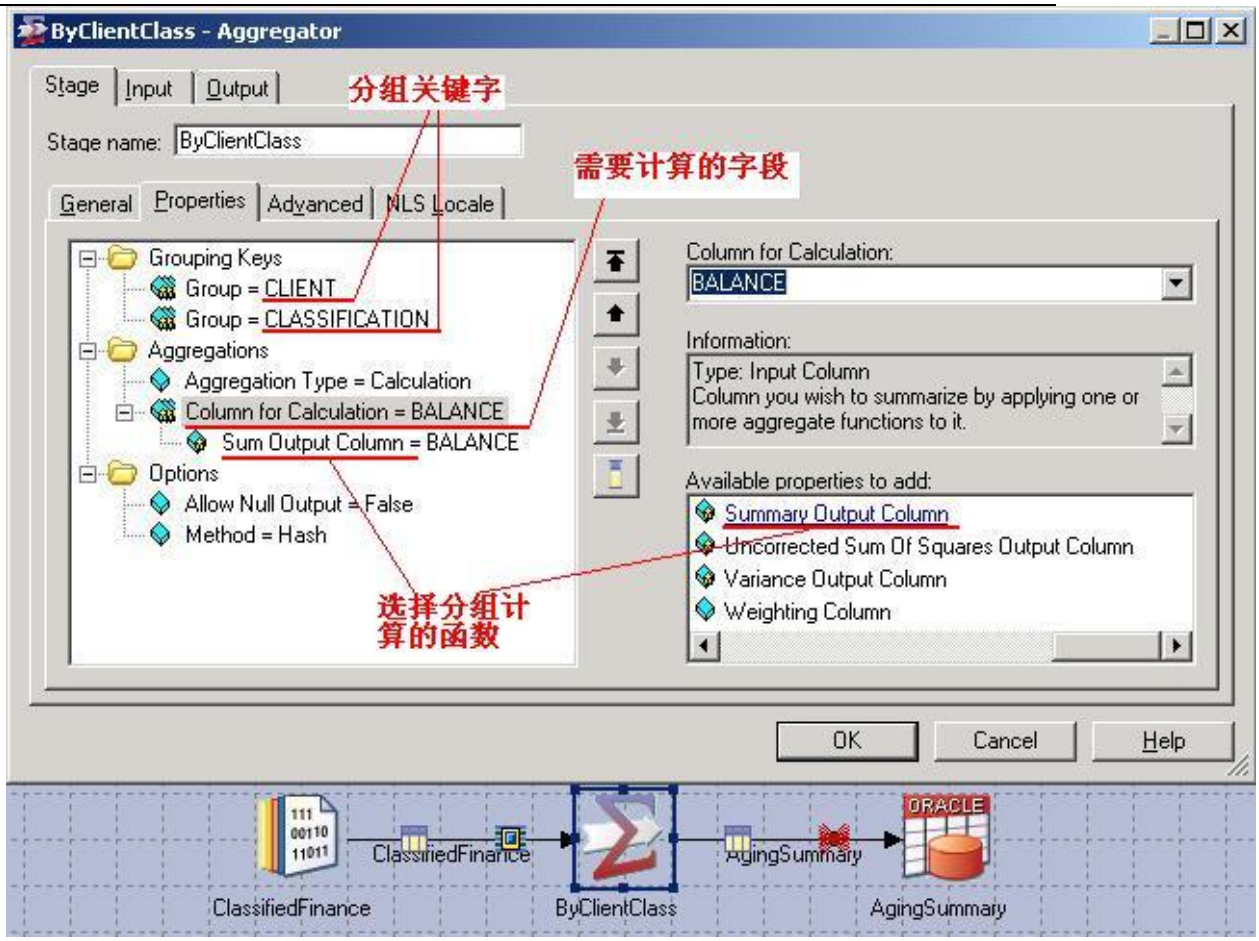
✓ Outputs Page：与Inputs Page类似，只是完成的是从数据库中读取数据。

4.1.18. Aggregator Stage

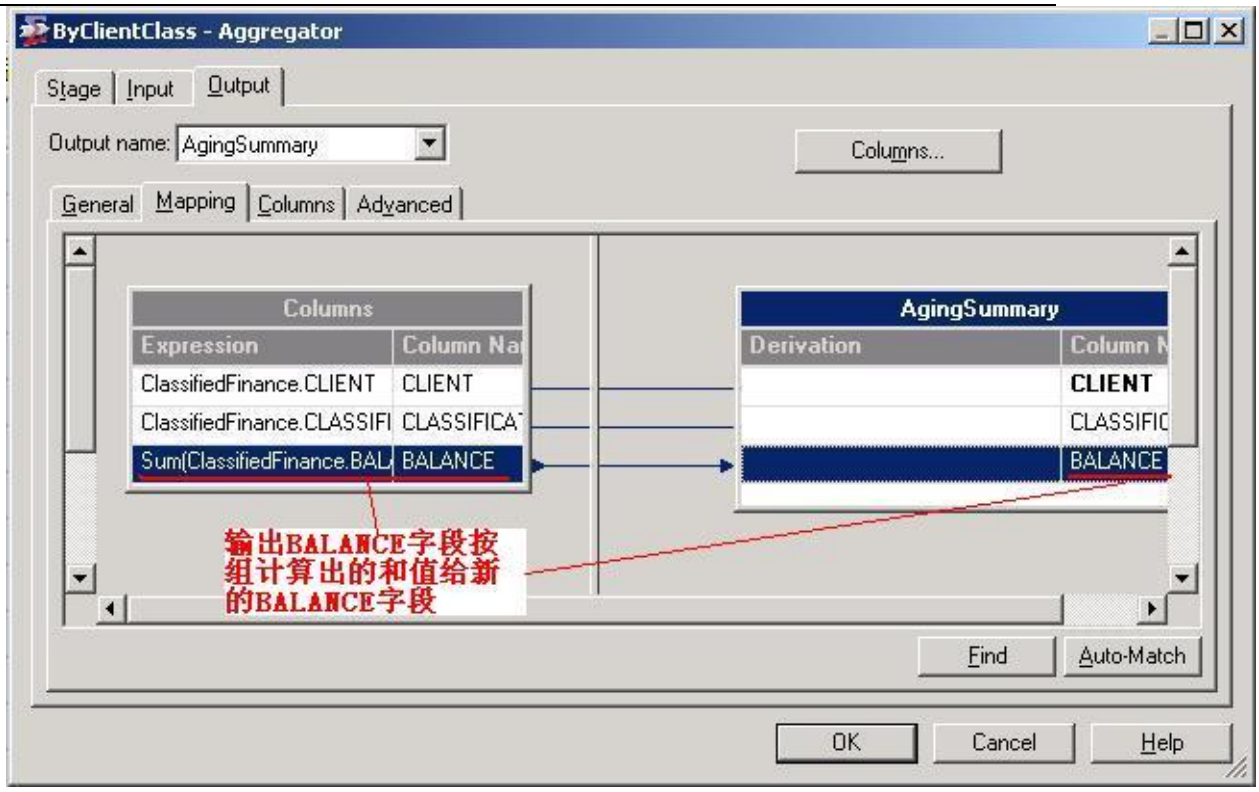
- Stage类型：Processing Stage
- 功能说明： 将输入的数据分组，计算各组数据的总和或者按组进行其他的操作，最后将结果数据输出到其他的stage。



- 具体用法：
  - ✓ Stage Page：描述stage的一般信息以及字段的分组信息和选择分组计算函数

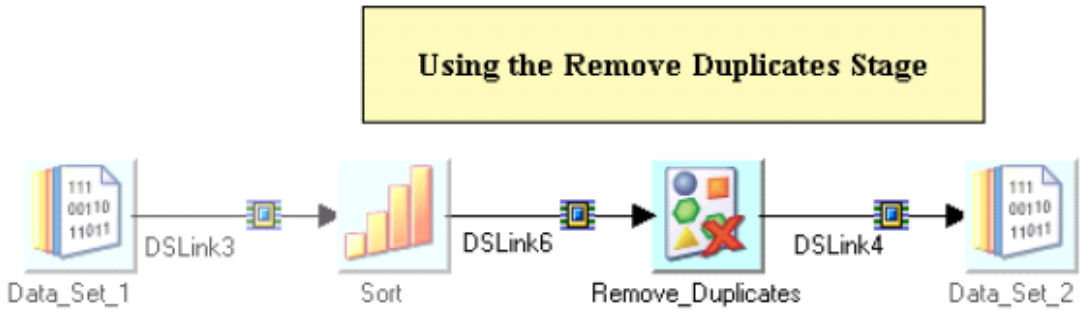


- ✓ Inputs Page：详细描述输入数据信息，一般直接反映输入数据字段信息
- ✓ Oupputs Page：详细描述输出数据信息，即经过分组计算后的数据字段信息

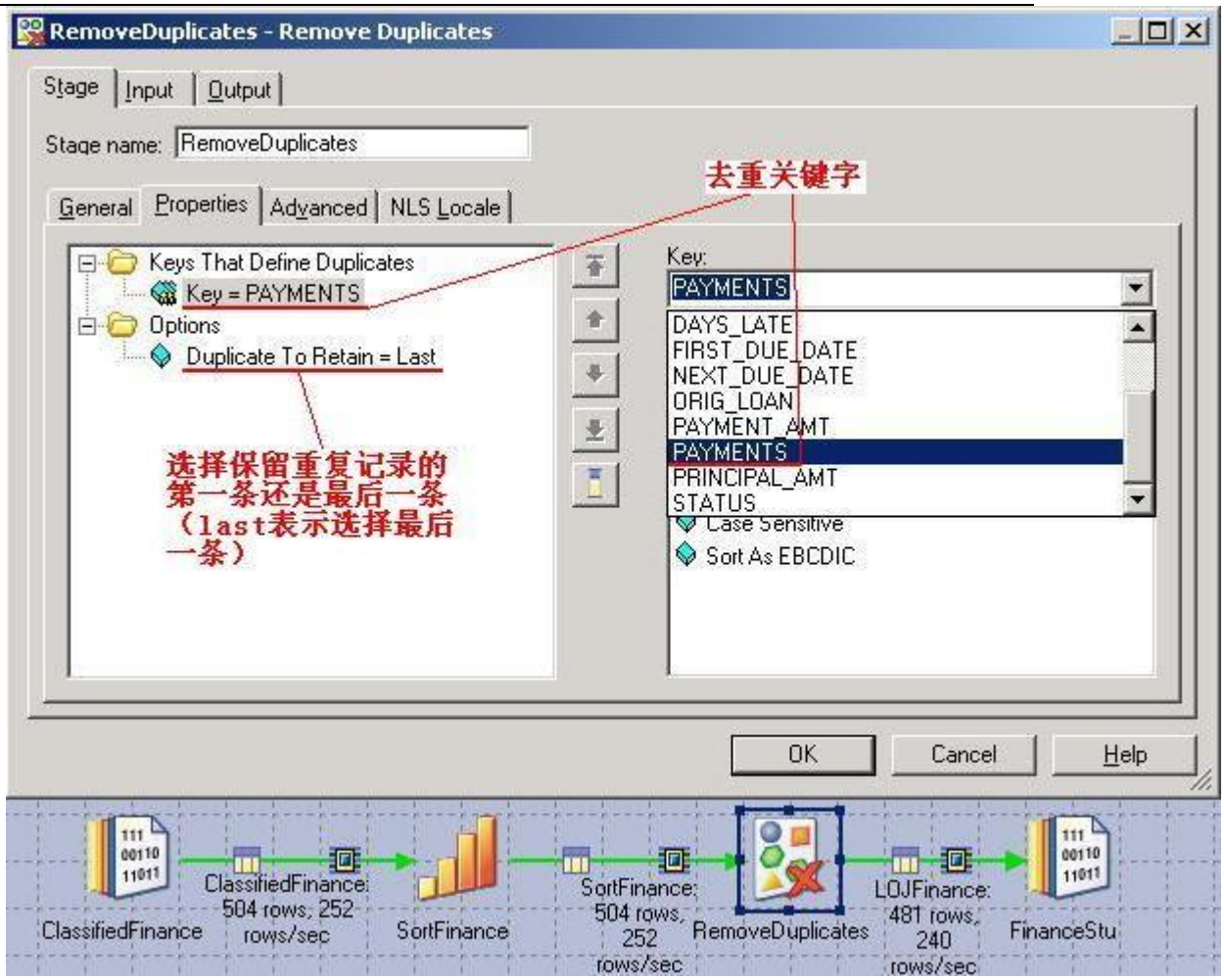


4.1.19. Remove Duplicates Stage

- Stage类型：Processing Stage
- 功能说明： 输入根据关键字分好类的有序数据，去除所有记录中关键字重复的记录，通常与sort stage配合使用



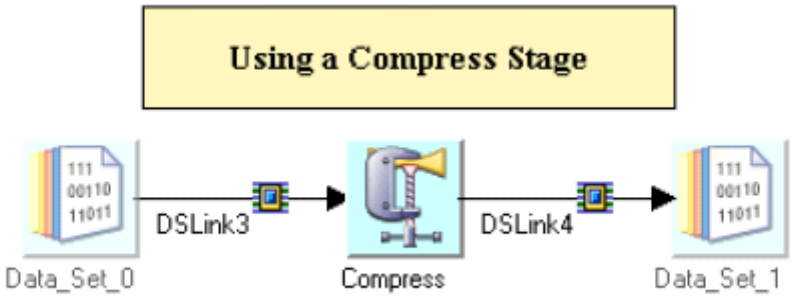
- 具体用法：
  - ✓ Stage Page：Properties中的key值与之前sort stage的分类key值相同



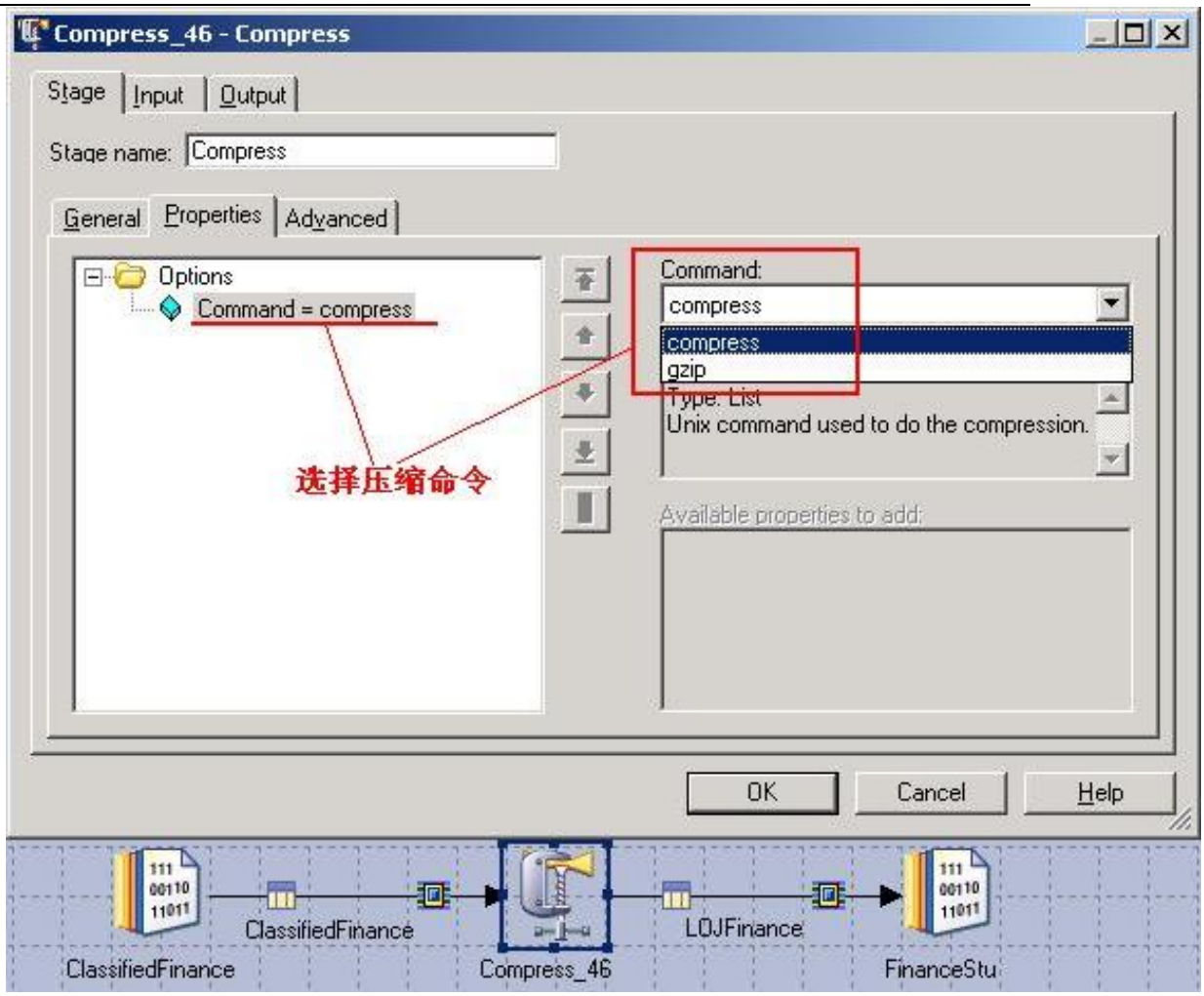
- ✓ Inputs Page：对输入数据字段的描述
- ✓ Oupputs Page：对输出数据字段的描述

4.1.20. Compress Stage

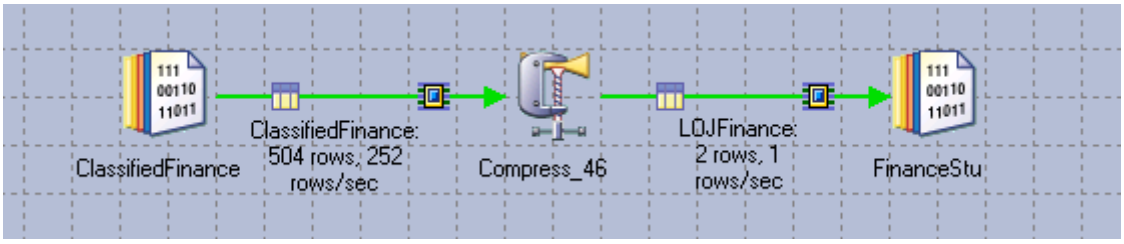
- ➢ Stage类型：Processing Stage
- ➢ 功能说明： 将data set文件压缩成二进制文件（与expend datastage相对应）



- ➢ 具体用法：
  - ✓ Stage Page



运行结果

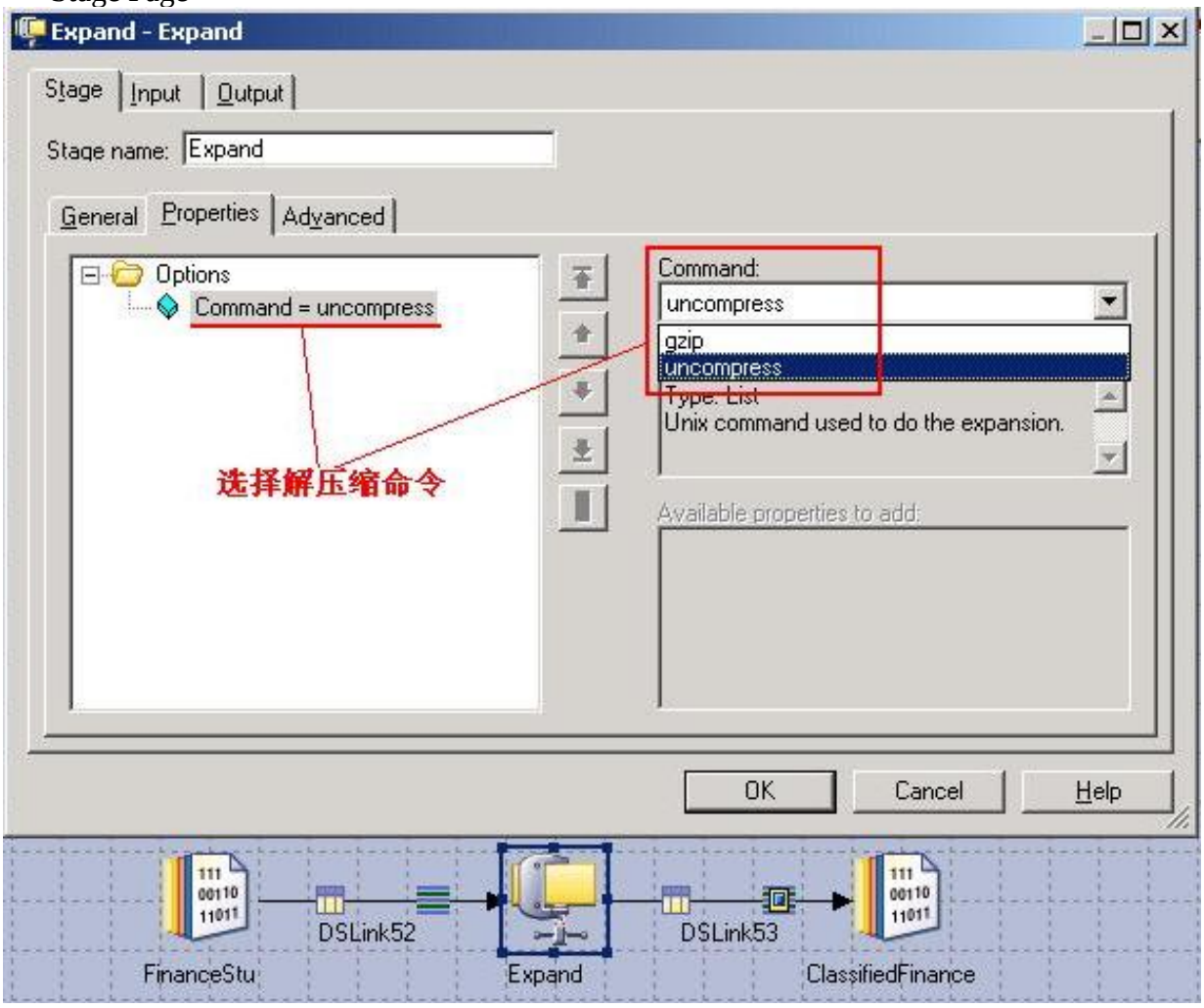


- ✓ Inputs Page：对输入数据字段的描述
- ✓ Oupputs Page：对输出数据字段的描述

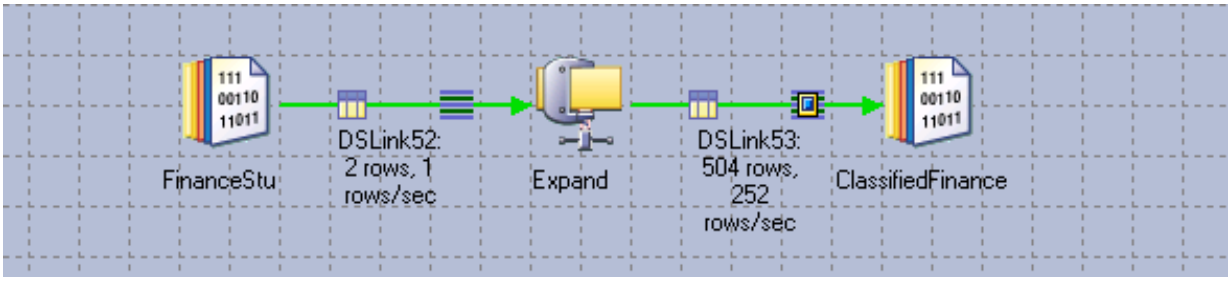
4.1.21. Expand Stage

- ➢ Stage类型：Processing Stage
- ➢ 功能说明：将压缩的二进制文件解压缩（解压缩compress stage生成的压缩文件）
- ➢ 具体用法：

✓ Stage Page



运行结果：



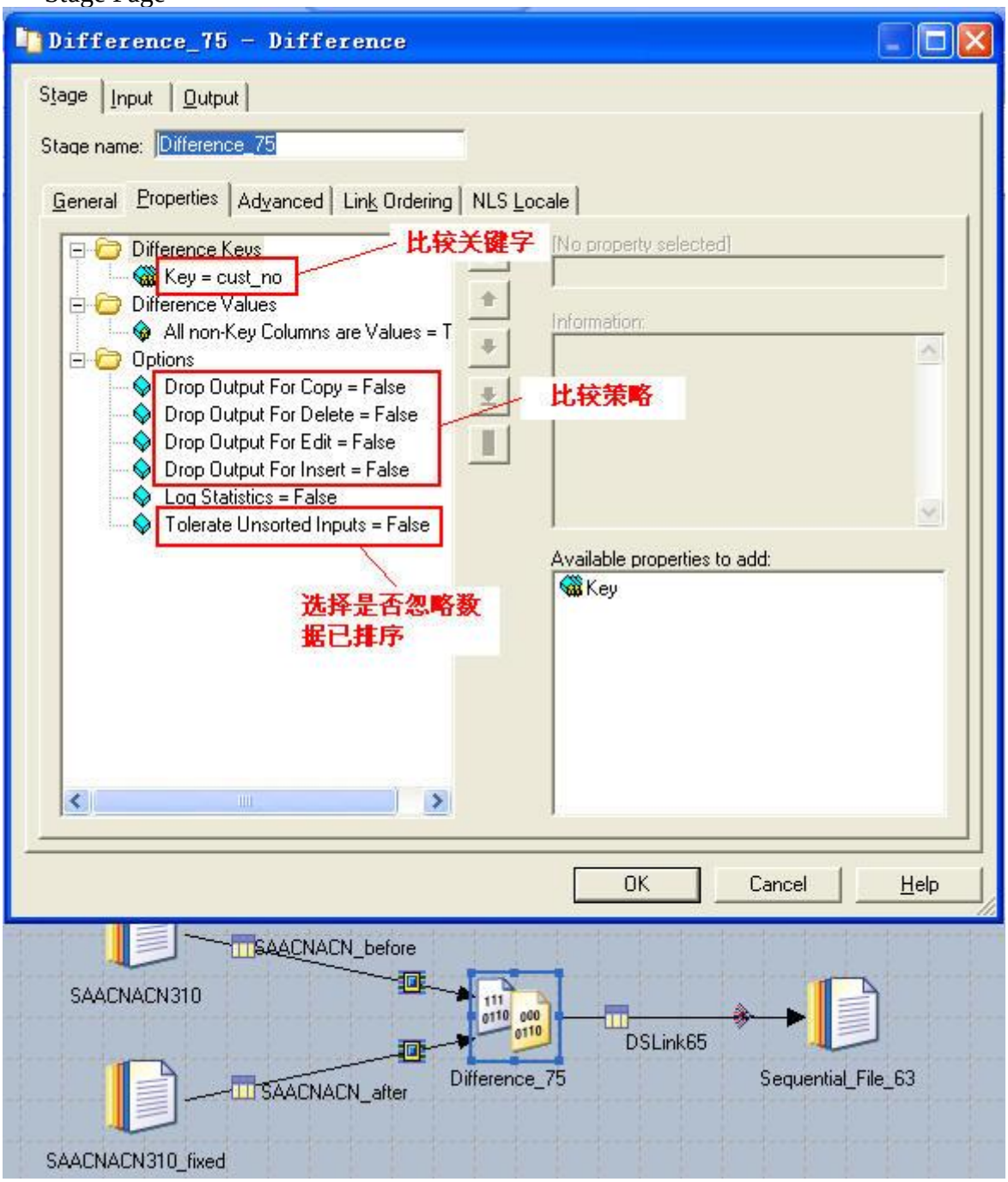
✓ Inputs Page：对输入数据字段的描述
✓ Oupputs Page：对输出数据字段的描述

4.1.22. Difference Stage

➢ Stage类型：Processing Stage
➢ 功能说明： 按字段比较两个文件，找出不同的记录。（两个文件before和after，

以before为准，与after文件中的记录进行比较，找出before在after文件中没有或者有的记录）

- 具体用法：
  - ✓ Stage Page



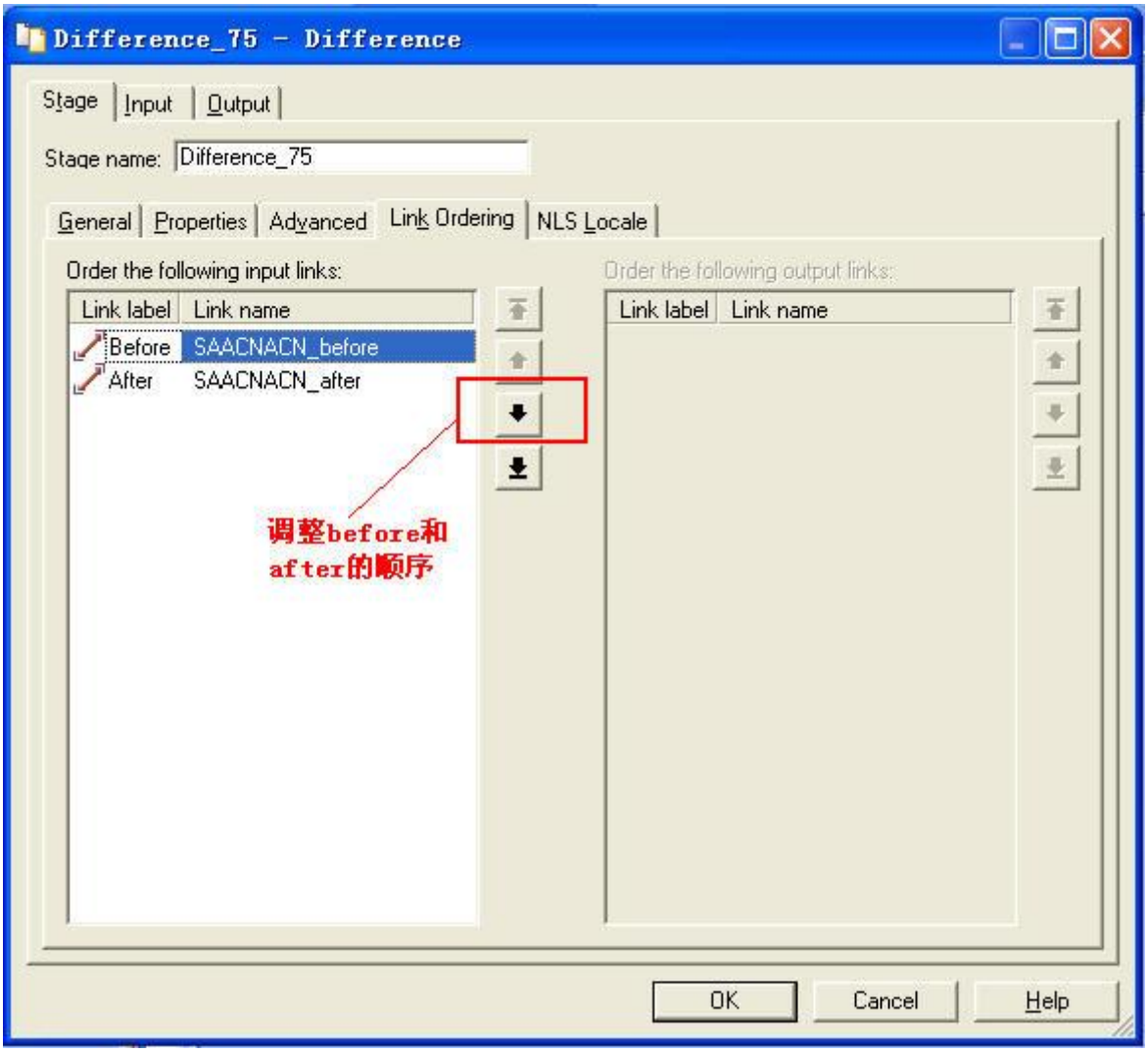
  - ➢ 比较策略说明

Drop Output For Copy	False：保留before及afte link中key值相同的行
	True：删除before及afte link中key值相同的行
Drop Output For Delete	False：保留before link中有但是afte link中没有的key值所在的行
	True：删除before link中有但是afte link中没有的

key值所在的行

Drop Output For Edit	False：保留key值相同,value不同的行 True：删除key值相同,value不同的行
Drop Output For Insert	False：保留before link中没有但afte link中有的key值所在的行 True：删除before link中没有但afte link中有的key值所在的行

调整before和after的顺序：

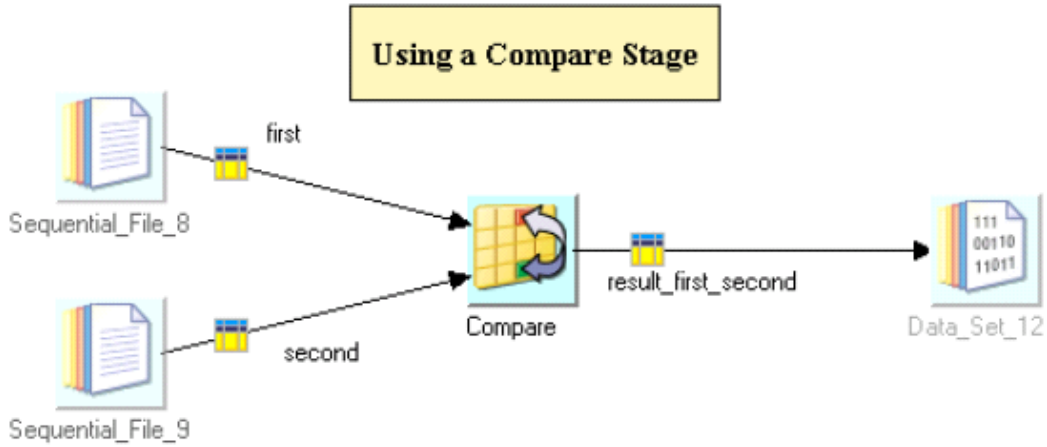


- ✓ Inputs Page：对输入数据字段的描述
- ✓ Oupputs Page：对输出数据字段的描述

4.1.23. Compare Stage

- ➢ Stage类型：Processing Stage

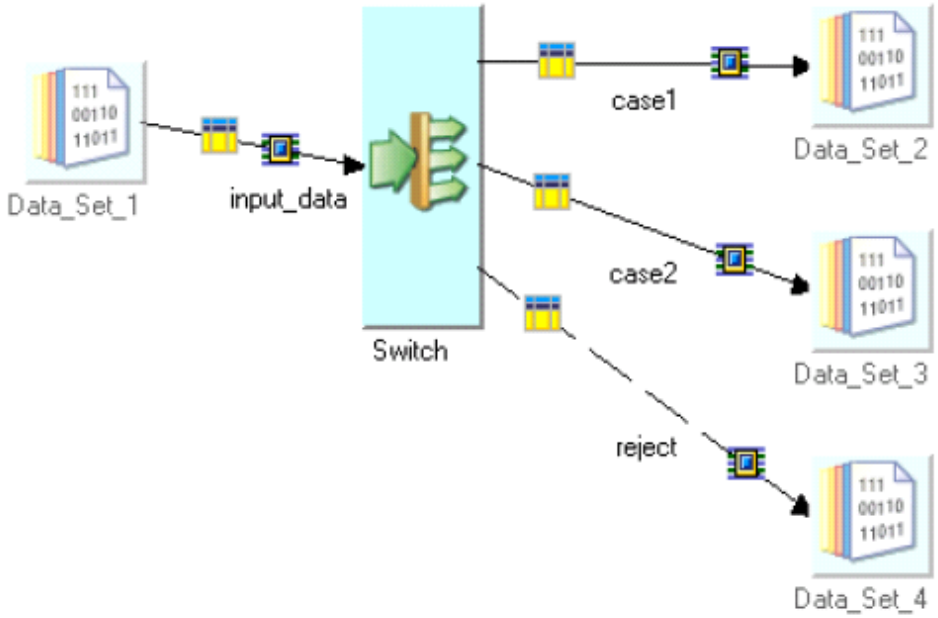
- 功能说明：按字段对比两个已经分类的有序的文件



- 具体用法：
  - ✓ Stage Page
  - ✓ Inputs Page：对输入数据字段的描述
  - ✓ Oupputs Page：对输出数据字段的描述

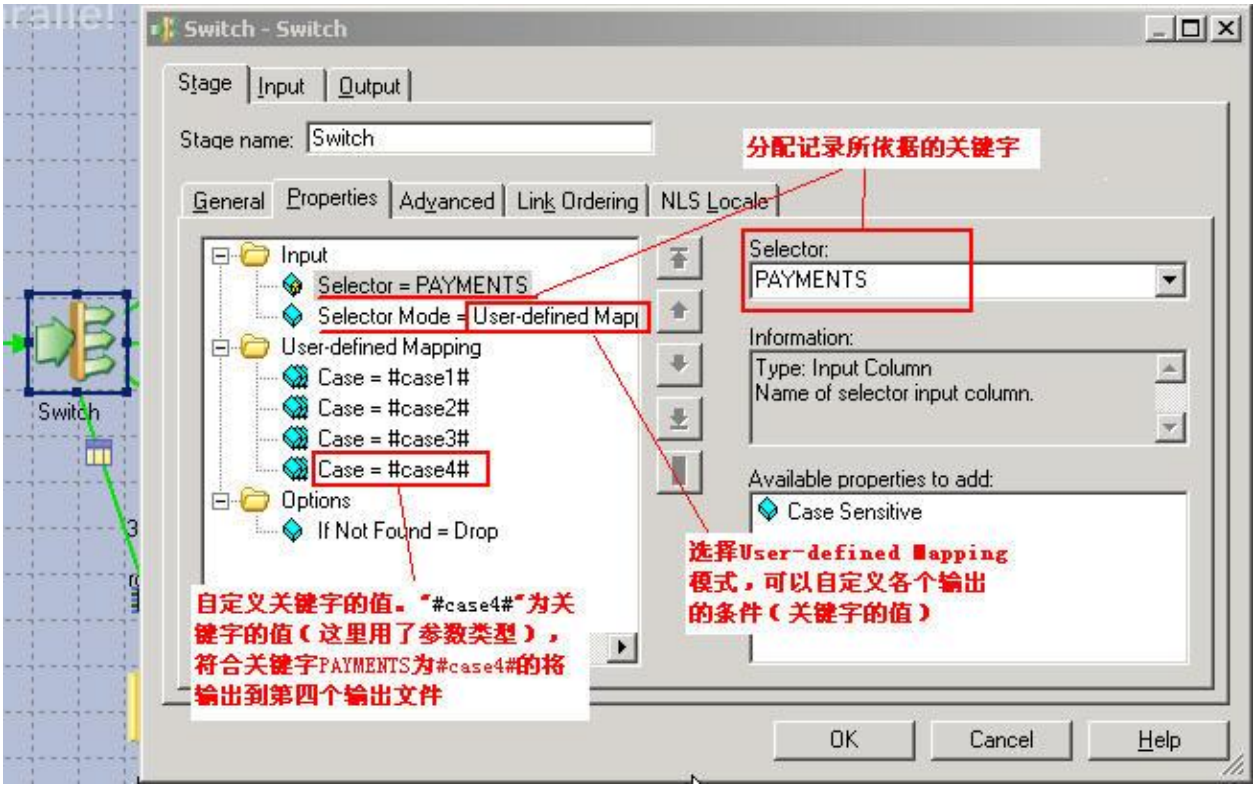
4.1.24. Switch Stage

- Stage类型：Processing Stage
- 功能说明：将文件按照一定的条件（一般为字段的值）分割成多个子文件。具体是将输入的每一条记录按照各自符合的条件（关键字的值）分配到不同的输出（Switch Stage 有一个input link 和多个output link，一个 reject link，output link最多可达128个；此功能很类似与C函数中的switch函数）。



- 具体用法：
  - ✓ Stage Page

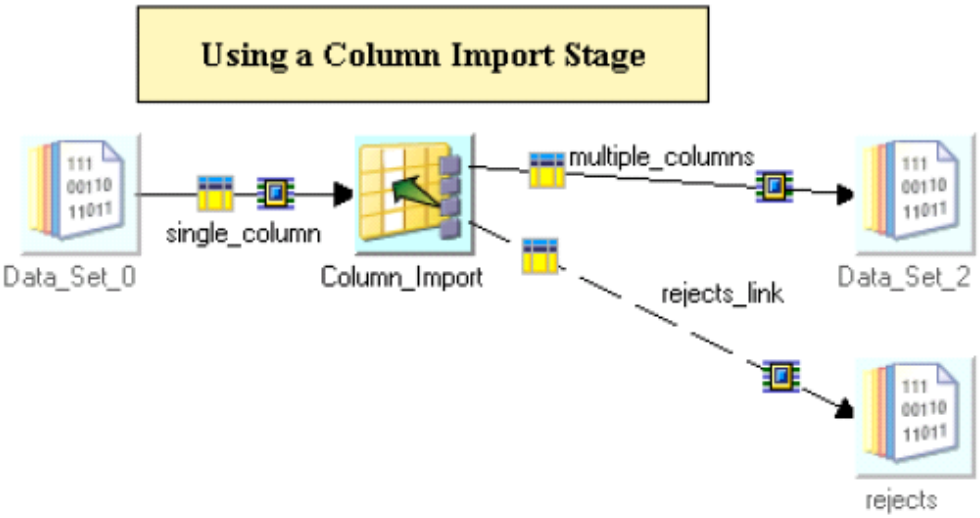
定义分配记录的关键字及其值



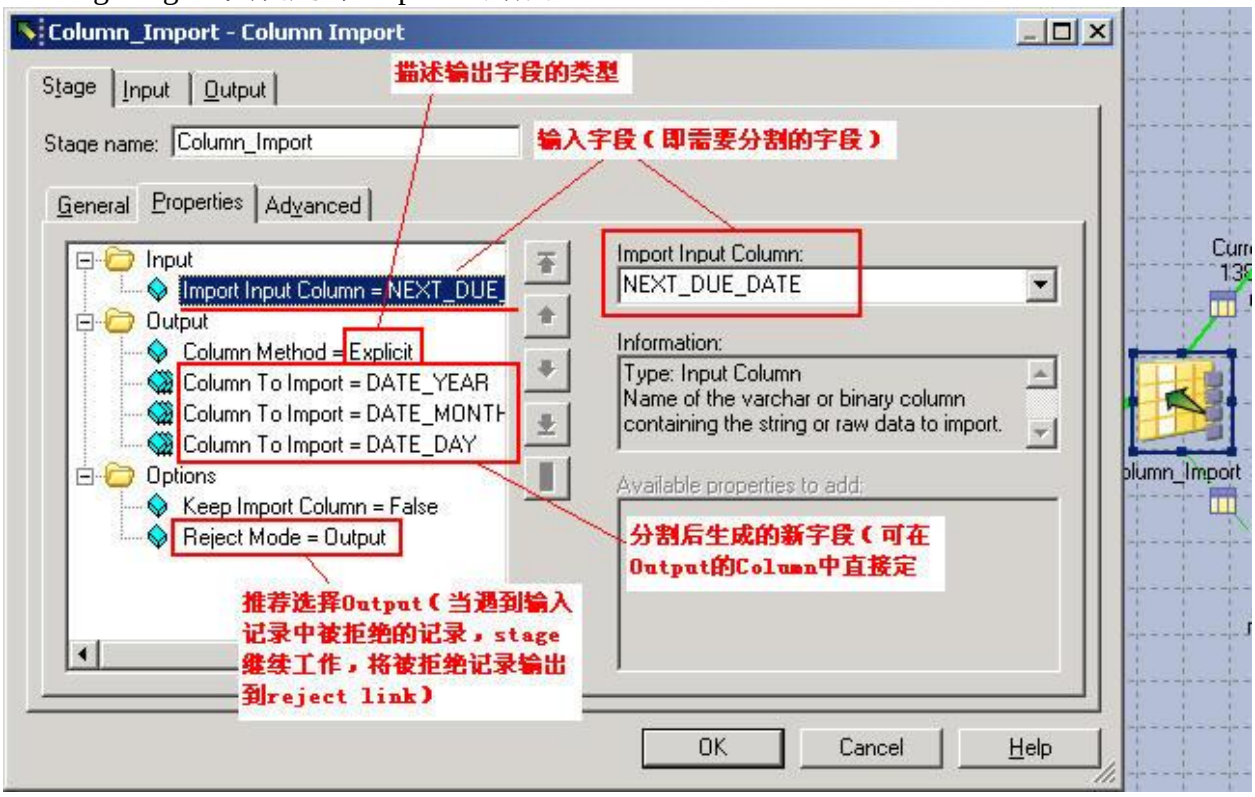
- ✓ Inputs Page：对输入数据字段的描述
- ✓ Oupputs Page：对输出数据字段的描述，通过Mapping自定义各个输出文件中的字段

4.1.25. Column Import Stage

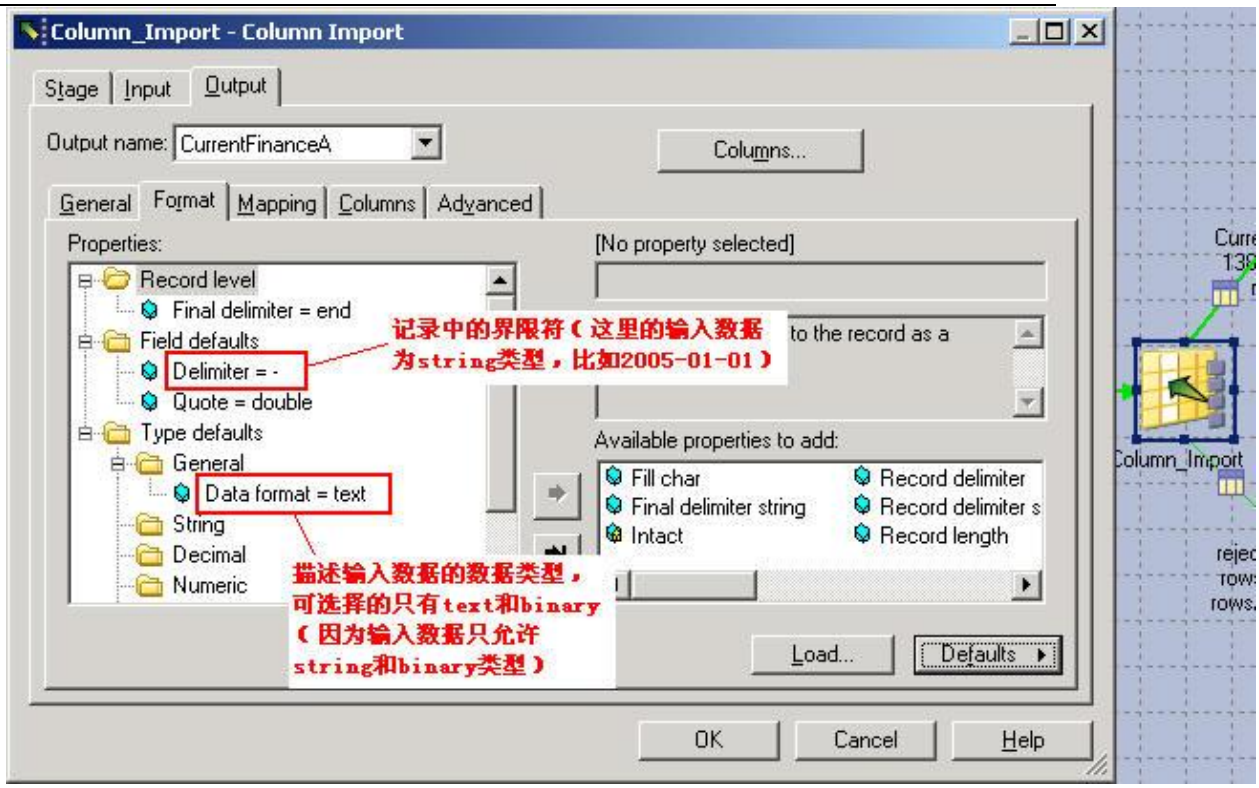
- ➢ Stage类型：Restructure Stage
- ➢ 功能说明： 将一个字段中的数据输出到多个字段中。（也可以用这个stage完成分割单个字段数据到多个字段的目的，此时，输入数据应为定长或者有可以被识别的可分割的界限，必须是String或者Binary类型的，输出数据可以是任何数据类型）



- 具体用法：
  - ✓ Stage Page：关键是对Properties的配置



  - ✓ Inputs Page：对输入数据字段的描述
  - ✓ Oupputs Page：对输出数据字段的描述，在Column自定义输出字段



✓ 经过stage的前后数据的对比

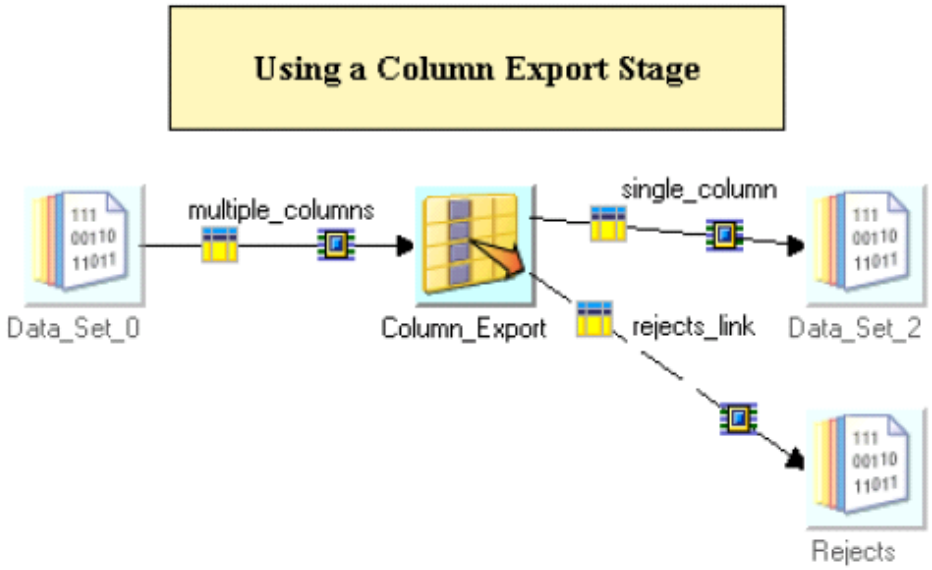
ACCOUNT	NEXT_DUE_DATE
298253	2003-04-01
298257	2003-05-20
298261	2003-03-10
298265	2003-04-01
298269	2003-05-20

字段分割后：

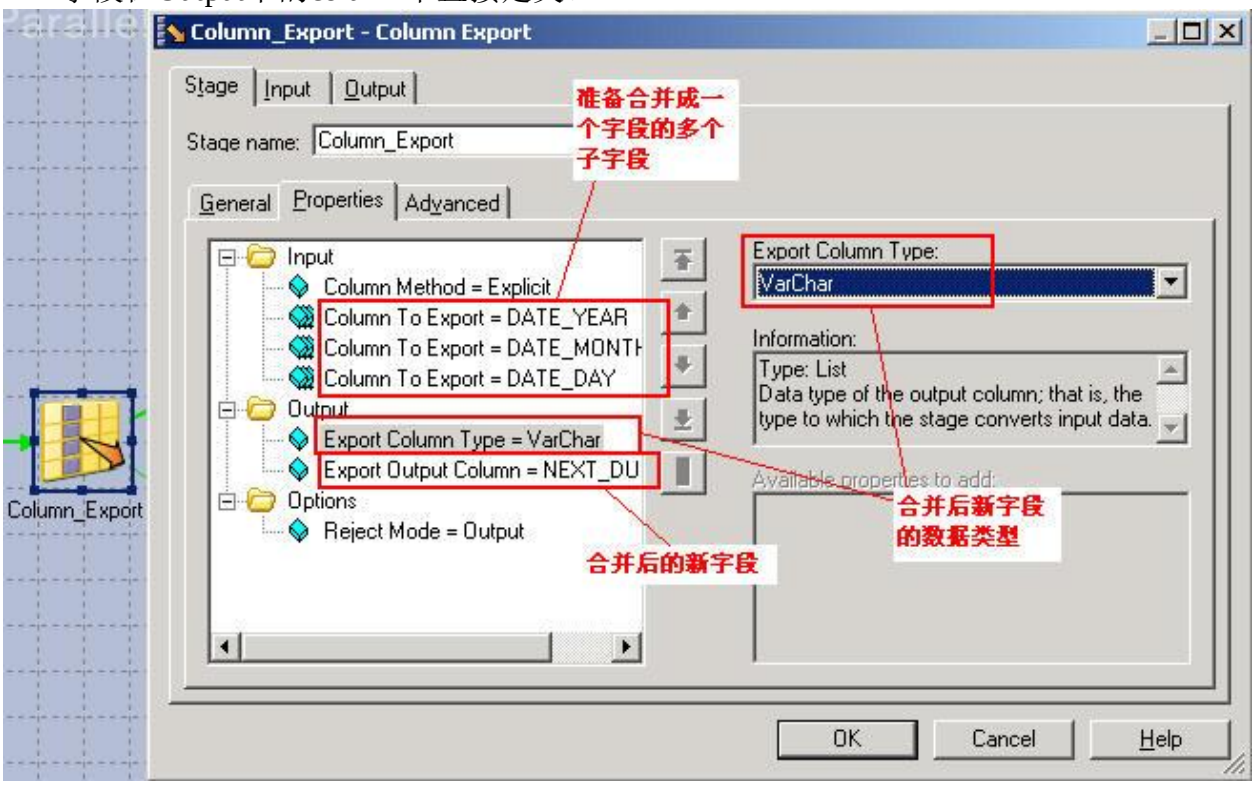
ACCOUNT	DATE_YEAR	DATE_MONTH	DATE_DAY
298253	2003	04	01
298257	2003	05	20
298261	2003	03	10
298265	2003	04	01
298269	2003	05	20
298273	2003	03	10

4.1.26. Column Export Stage

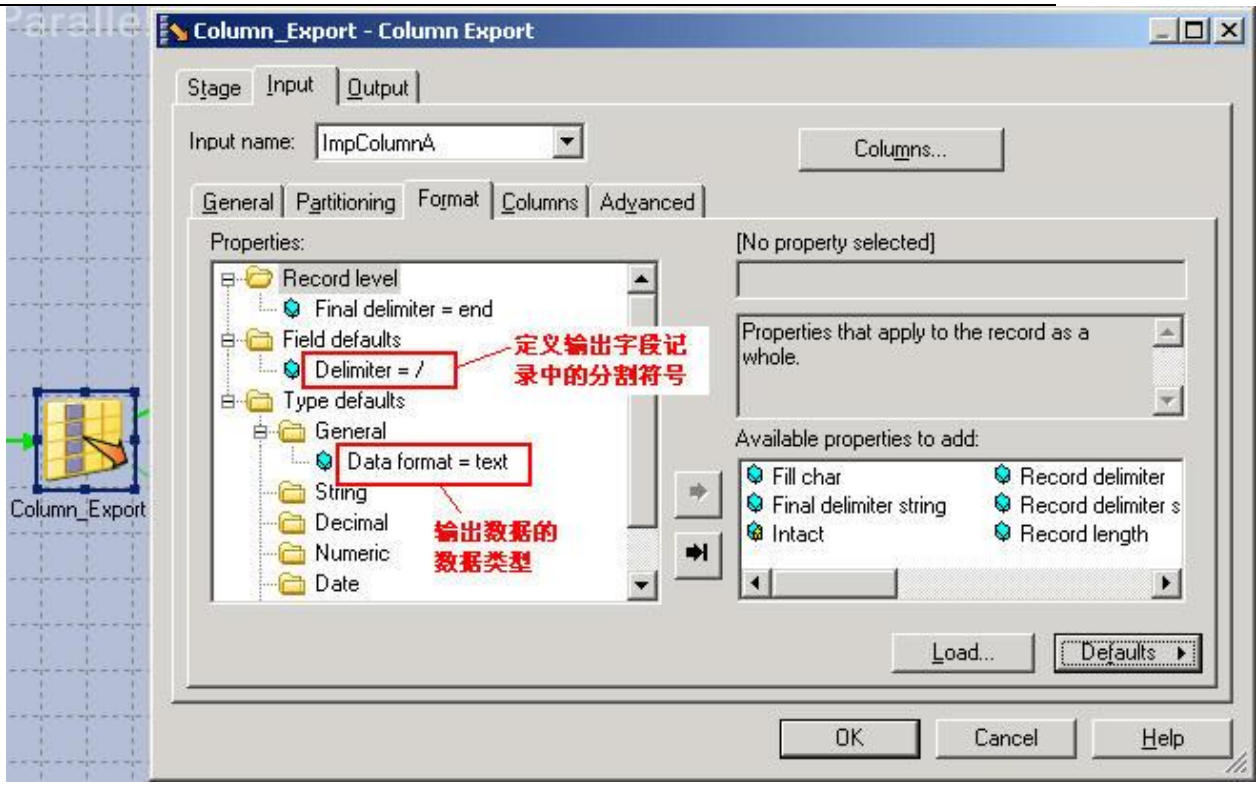
- Stage类型：Restructure Stage
- 功能说明： 与Column Import Stage相反，将多个类型不同的字段合并成一个string或者binary类型的字段。



- 具体用法：
  - ✓ Stage Page：关键是properties的配置，选择将哪些字段合并，合并后的新字段在Output下的column中直接定义。



  - ✓ Inputs Page：对输入数据字段的描述，这里关键是Format的配置，决定合并后的字段的格式



✓ Oupputs Page：对最终合并后的输出数据字段的描述
✓ 经过stage的前后数据的对比

ACCOUNT	DATE_YEAR	DATE_MONTH	DATE_DAY
298253	2003	04	01
298257	2003	05	20
298261	2003	03	10

合并字段后：

ACCOUNT	NEXT_DUE_DATE
298251	2003/05/20
298255	2003/03/10
298259	2003/04/01

5. 高级应用

5.1. DataStage BASIC接口

该功能主要应用于Job属性定义中的Job control中，该功能可用于在本Job中调用其它的Job，给Job赋参数、得到Job执行状态等。

5.2. 自定义Stage Type

为了扩大并行Stage type的应用范围，DataStage允许开发者定义自己的Stage type，并将其应用于Parallel Job中。目前有三种形式的Stage可以自己定义。

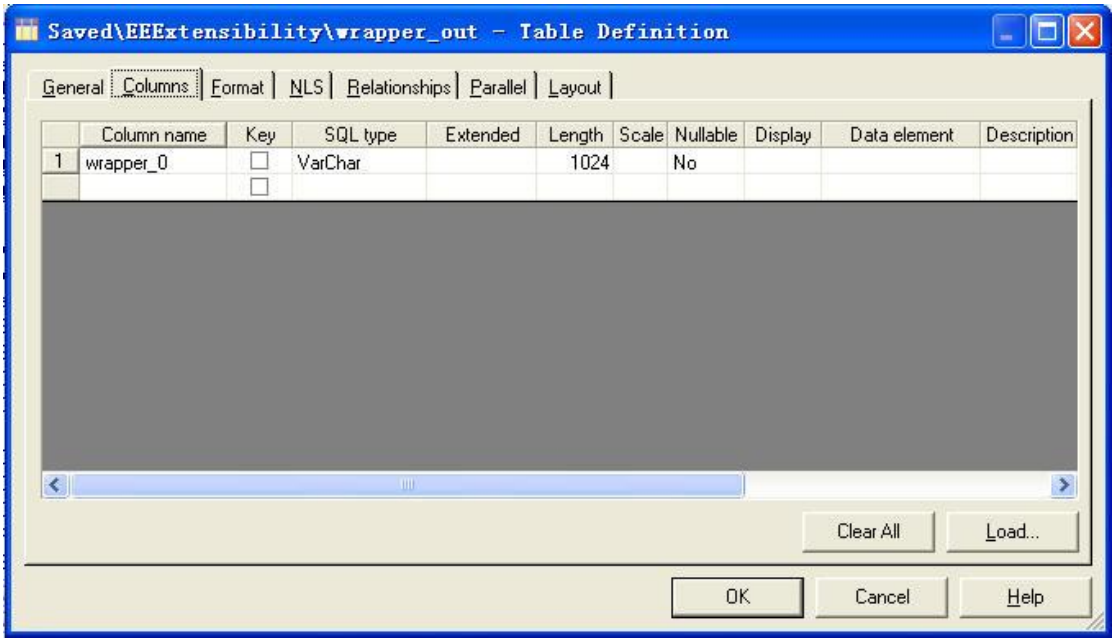
5.2.1. Wrapped Stage

适用于对应用本身不做处理。而且对系统的性能要求不高。

可以通过定义Wrapped stage来实现Unix命令，并且可以定义相应参数和输入输出。

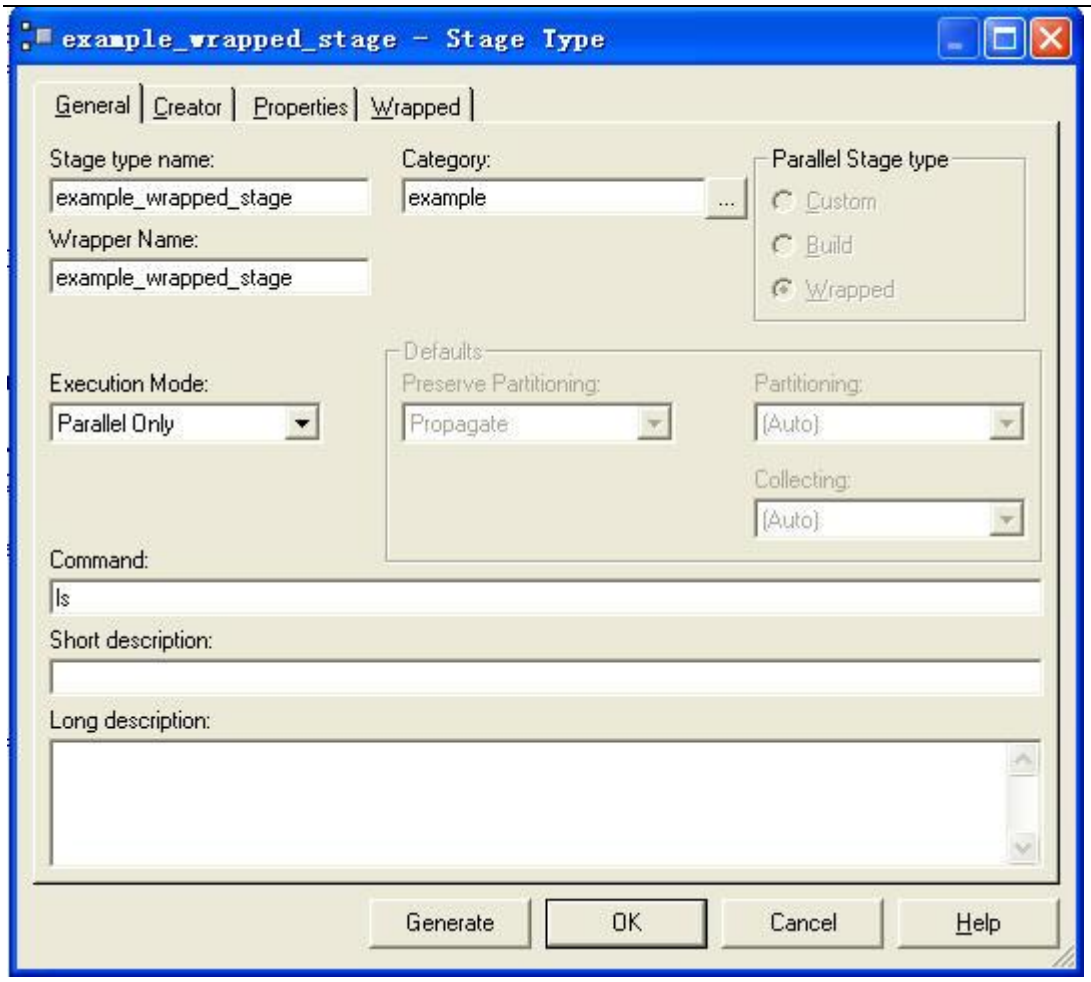
以实现unix的命令ls为例来说明。实现ls根目录下的文件。

首先需要为自定义的stage定义输入和输出表格式。本例中ls不需要定义Input项，故不用定义输入表格式。为Output项定义一个表，即为ls的输出定义表wrapper_out：

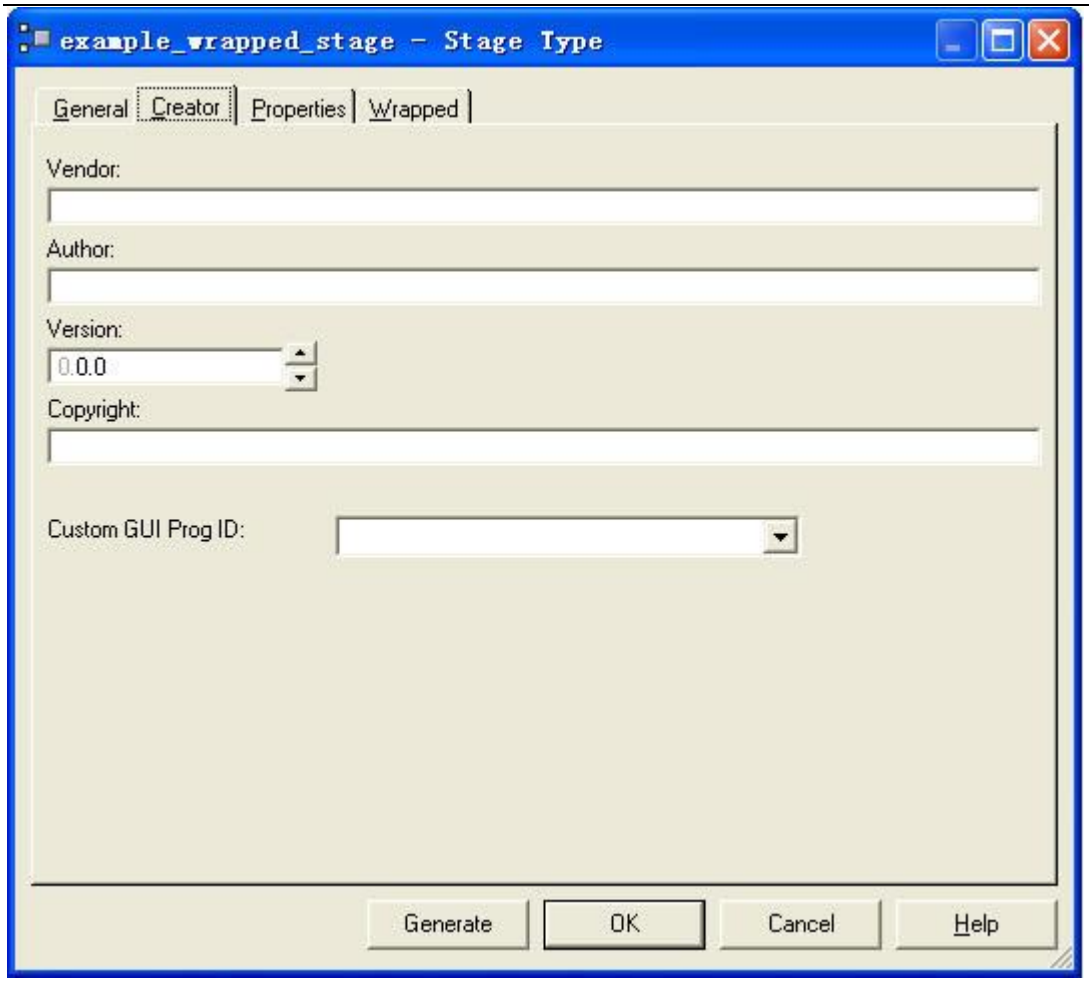


定义好输入输出后，开始定义wrapped stage。步骤如下：

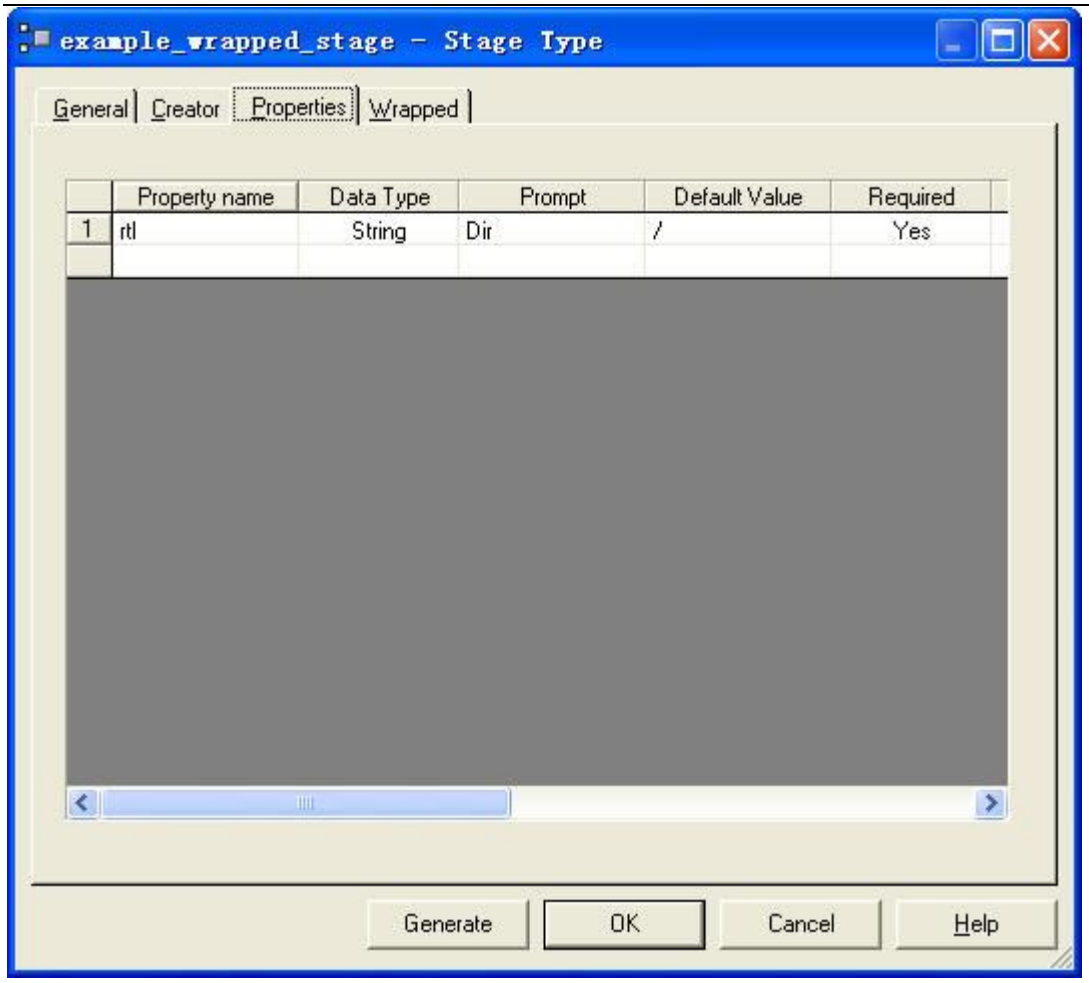
General页面定义stage的基本信息，如名称、存放目录、执行模式、Unix命令以及描述等：



Creator页面是描述创建者的信息以及定义版本号：



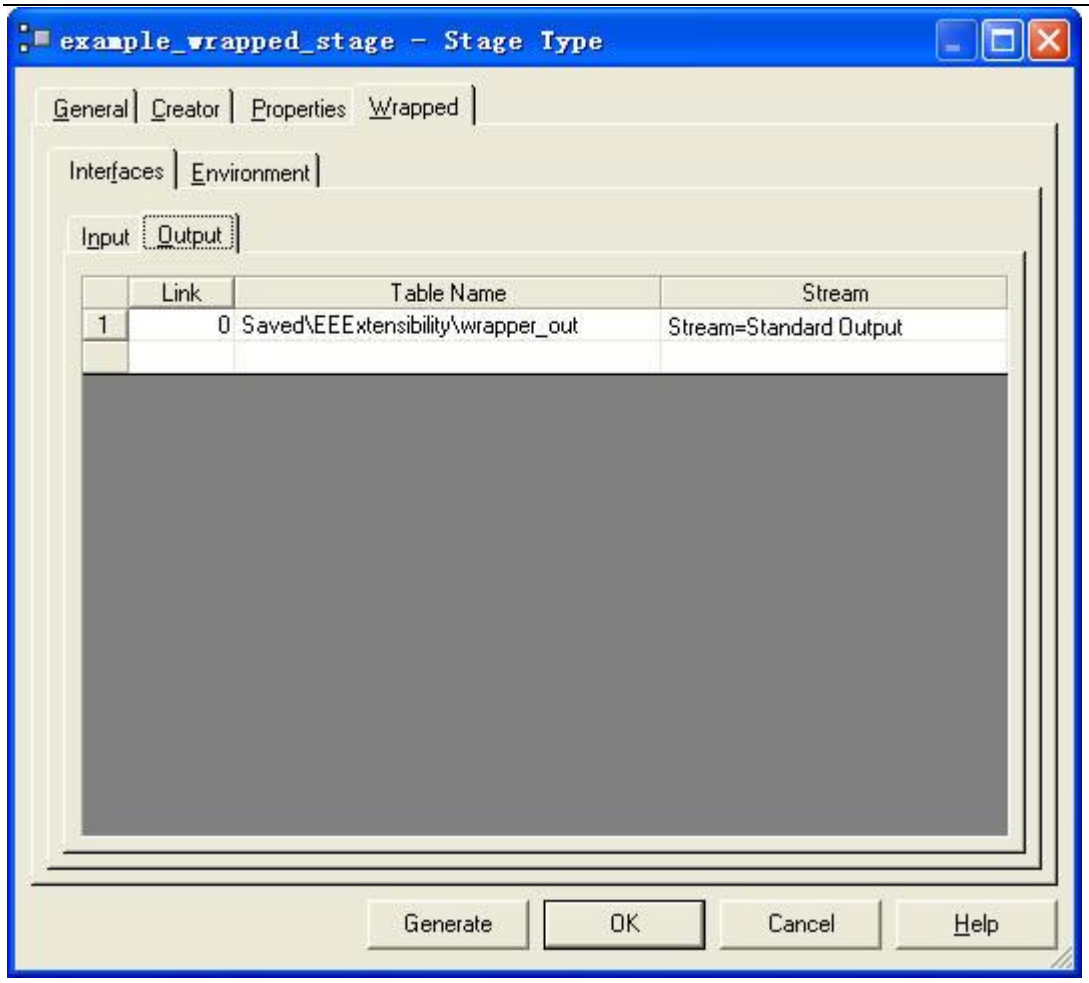
Properties页面定义stage所需的propertiy 组建：为命令ls定义一个目录



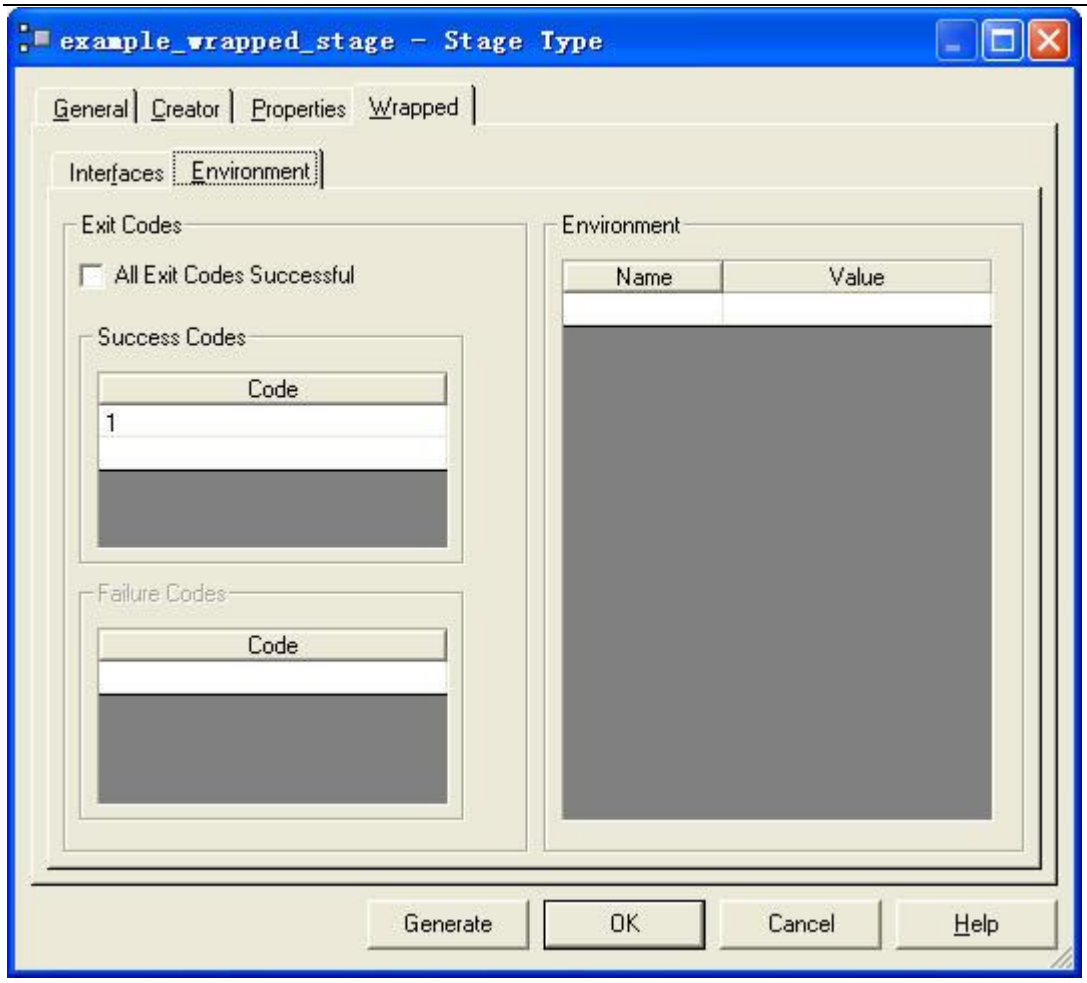
Wrapped页面定义stage的输入输出格式以及环境变量；

定义stage的Input：由于命令ls不需要输入格式定义，故这里不做定义；

定义stage的Output：在Table Name中选择已经定义好的表wrapper_out，Stream选择Standard Output。



定义环境变量，可以自定义stage执行成功或者失败后的返回码，以及定义stage相关的环境变量：



最后点击按钮“Generate”使自定义的wrapped stage生效。生成的自定义stage会出现在desinger的工具面板中，当定义成功后，就可以向其他stage一样运用在job中了。

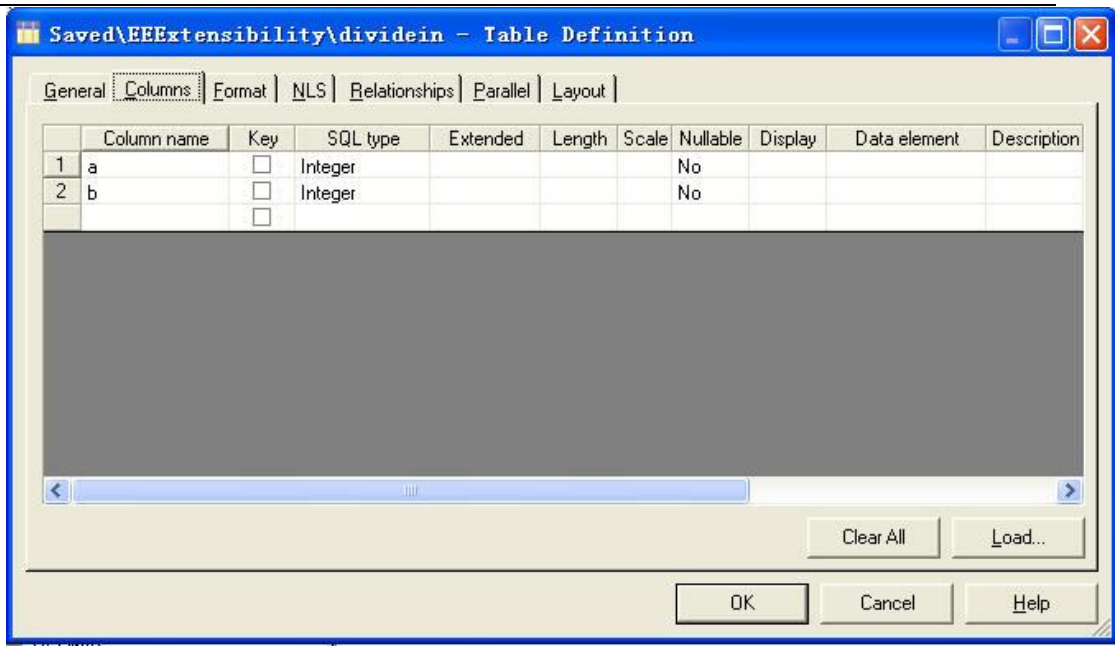
5.2.2. Build Stage

希望自己编码，但不需要动态的变化输入和输出参数。

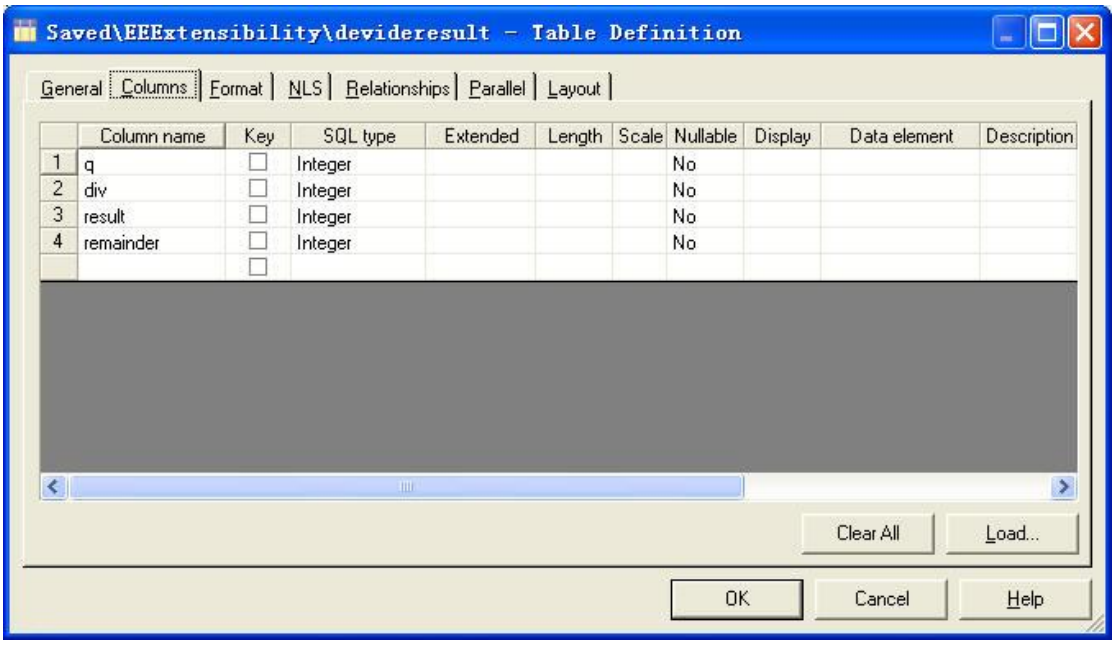
下面举例示范如何定义一个叫做Divide的Build stage。它的功能是：1、两个整型数据相除，输出两数相除的商和余数；2、检查除数是否为 0，如果为 0 则将记录输出到reject link。在这个例子中，还定义了一个最小除数，当输入数据的除数小于这个最小除数时，数据也会被拒绝，输出到reject link。

首先需要为自定义stage的输入输出定义相应的表。本例子中，为表dividein和表devideresult。

输入数据为两个准备相除的整型数据，表dividein定义如下：

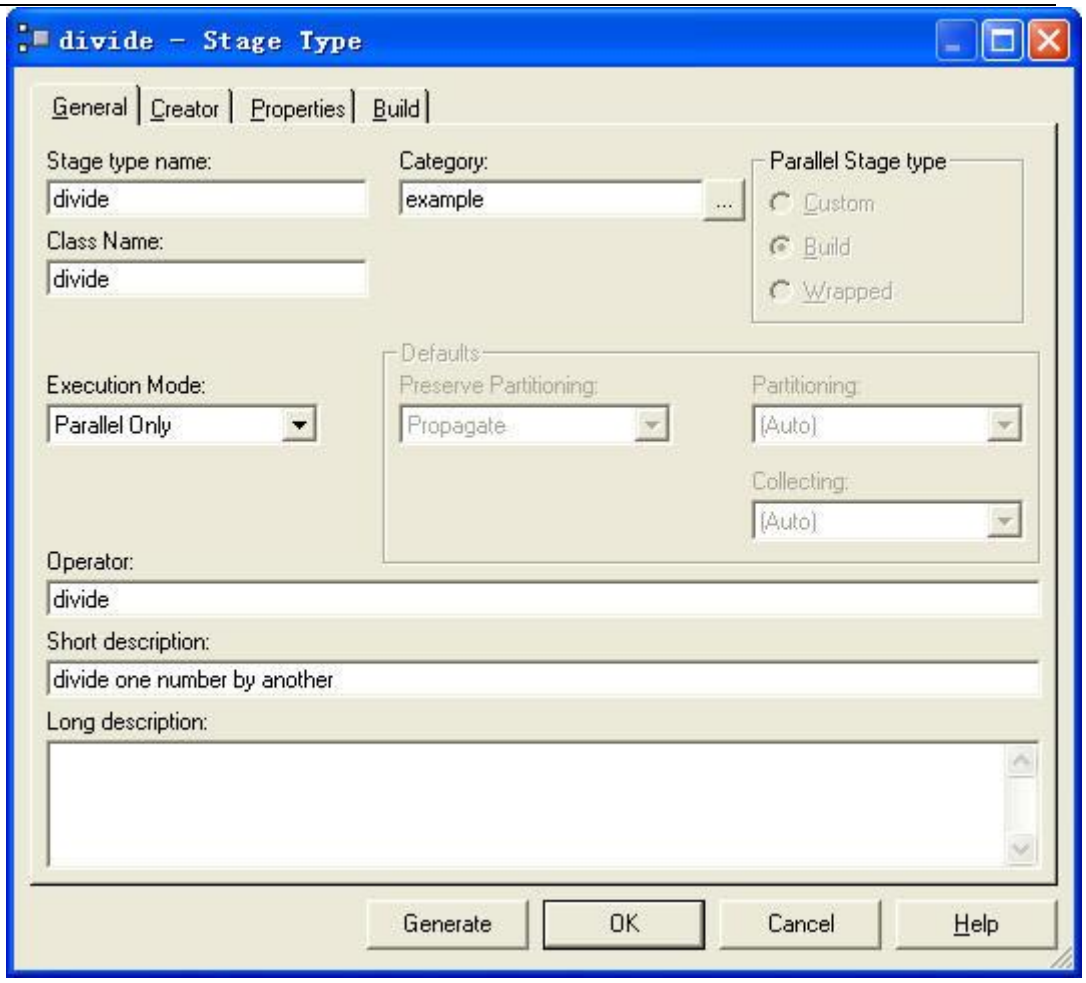


输出数据为原输出数据（被除数和除数）、商和余数，表devideresult定义如下：



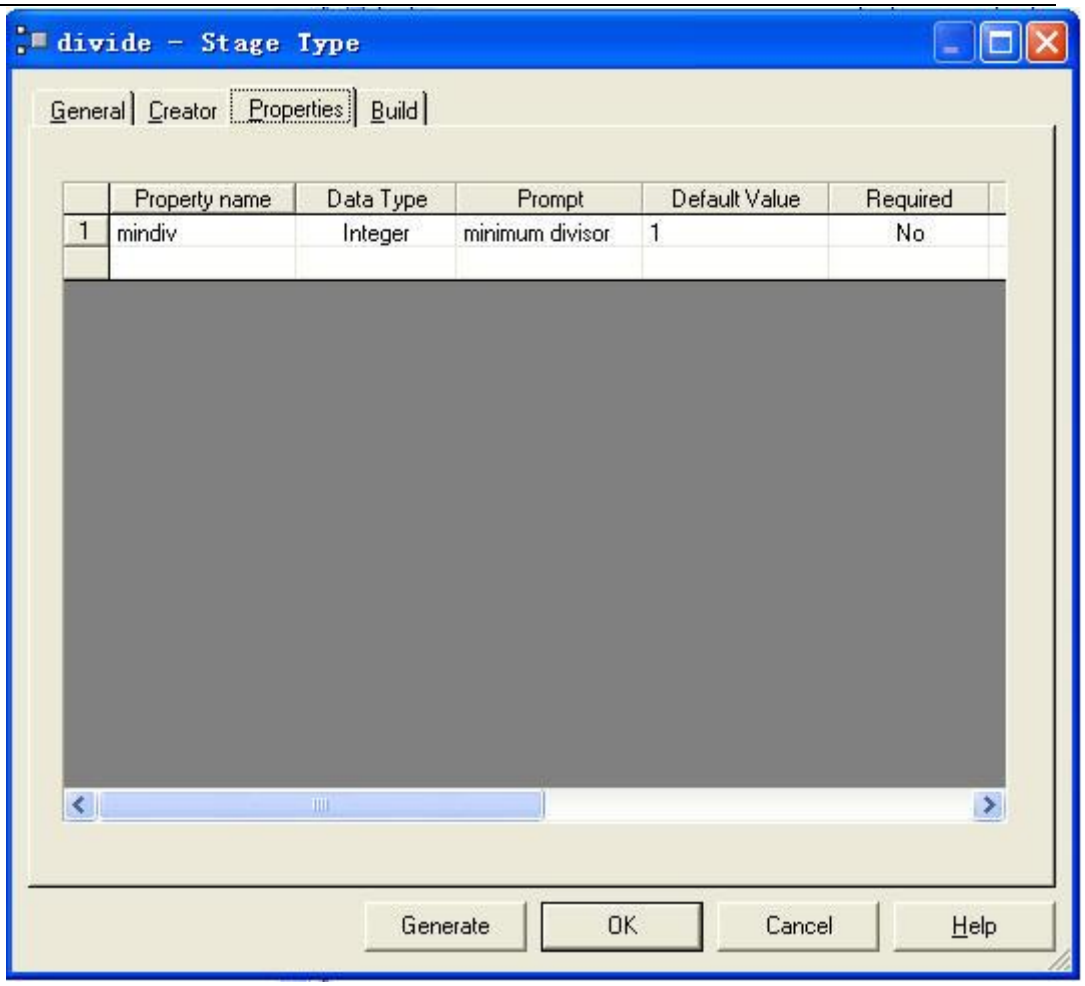
做完表定义的准备工作后，开始定义Build stage，名称为Divide。步骤如下：

定义General页面，主要定义stage的名称、目录、执行模式、算子以及简单的描述：



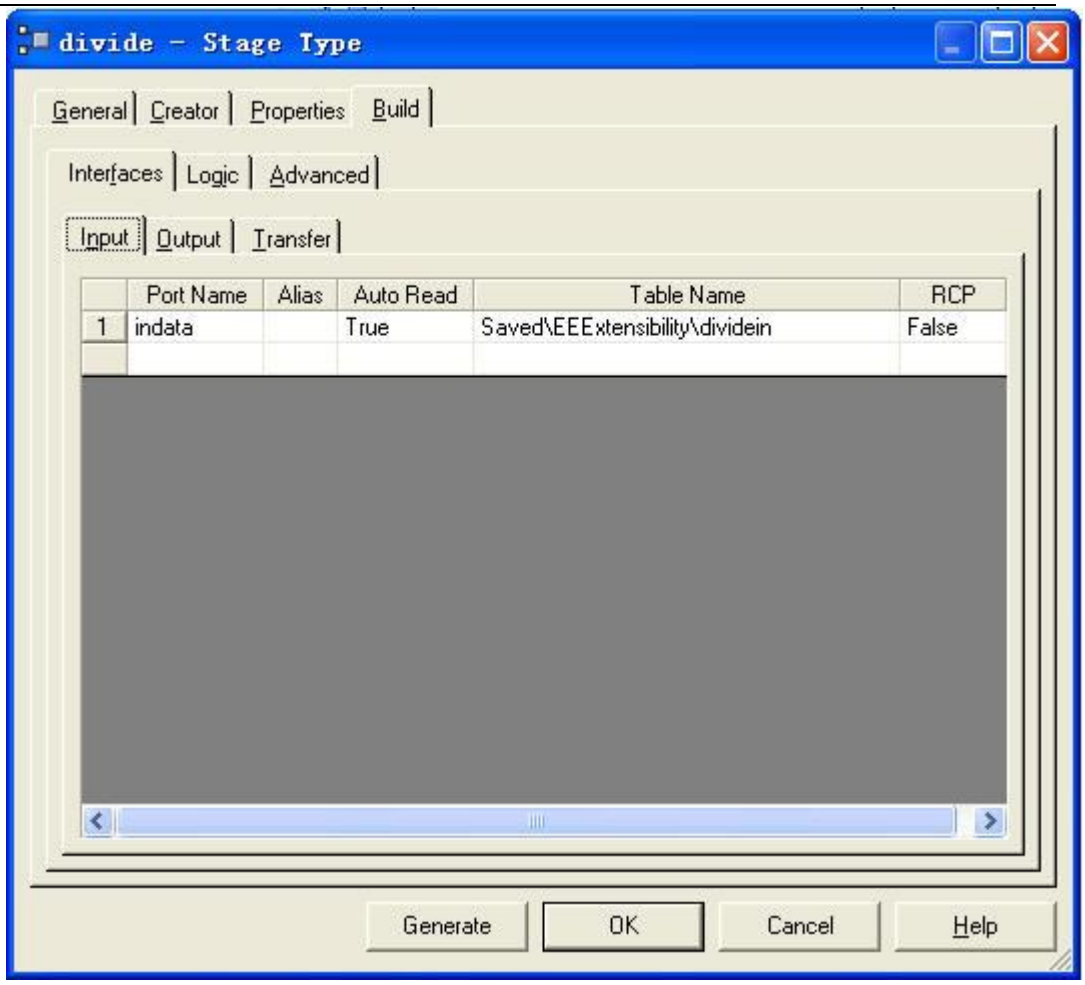
Creator页面主要描述创建者和版本信息；

Properties页面，定义一个变量字段，在stage中接收最小除数数值：

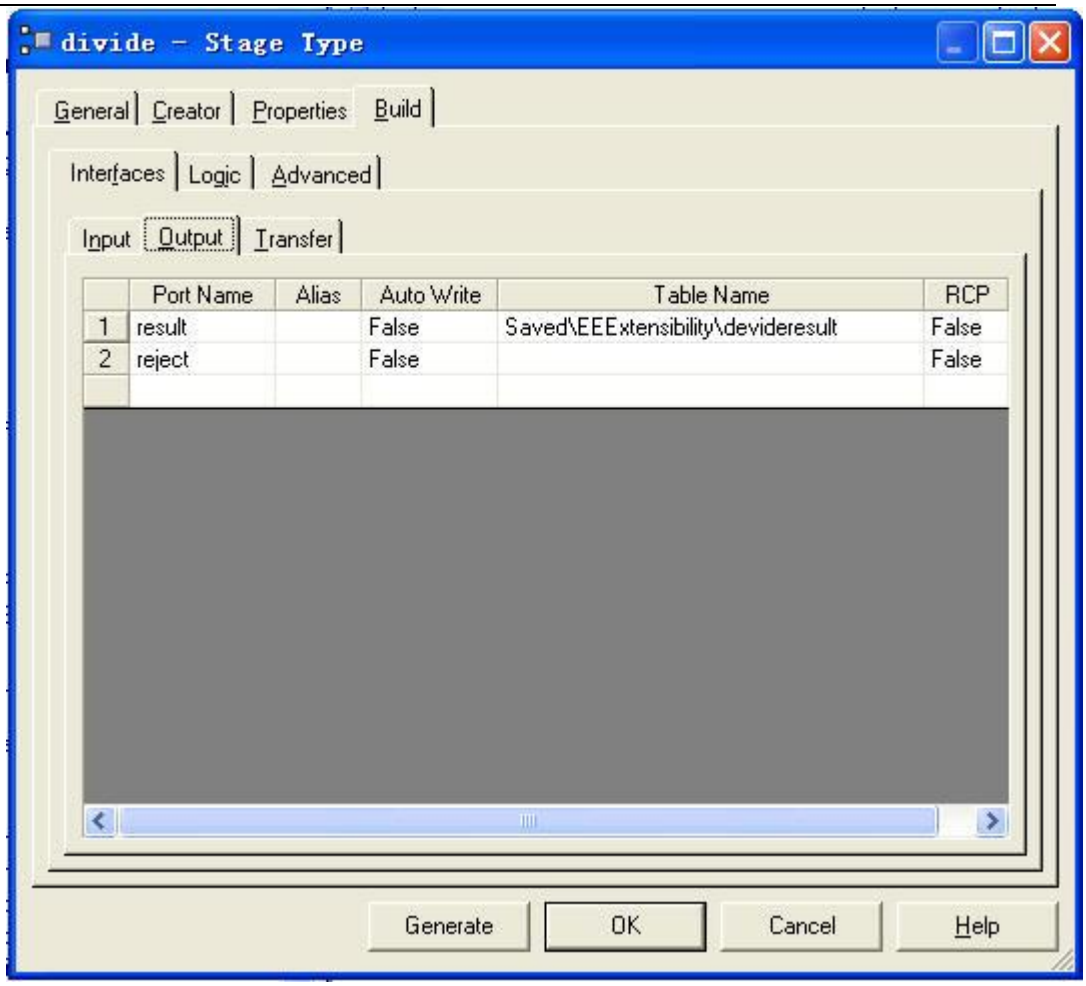


Build页面中定义stage相关的输入、输出、以及自定义执行的运算等。在本例中，需要定义一个输入和两个输出（result link和reject link）。输入定义为自动读入，输出定义为非自动输出。

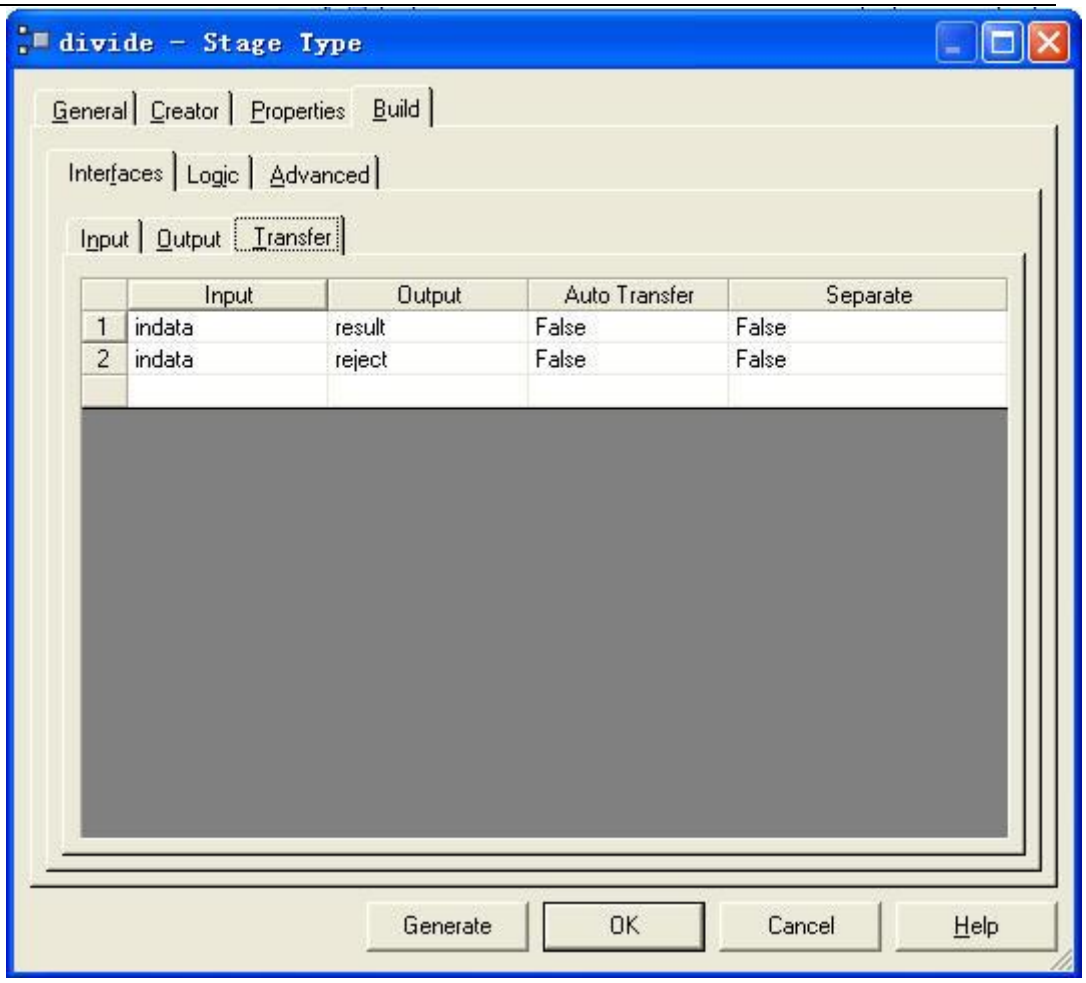
Input定义如下：Table Name选择预先定义好的表dividein，Auto Read置为Ture；



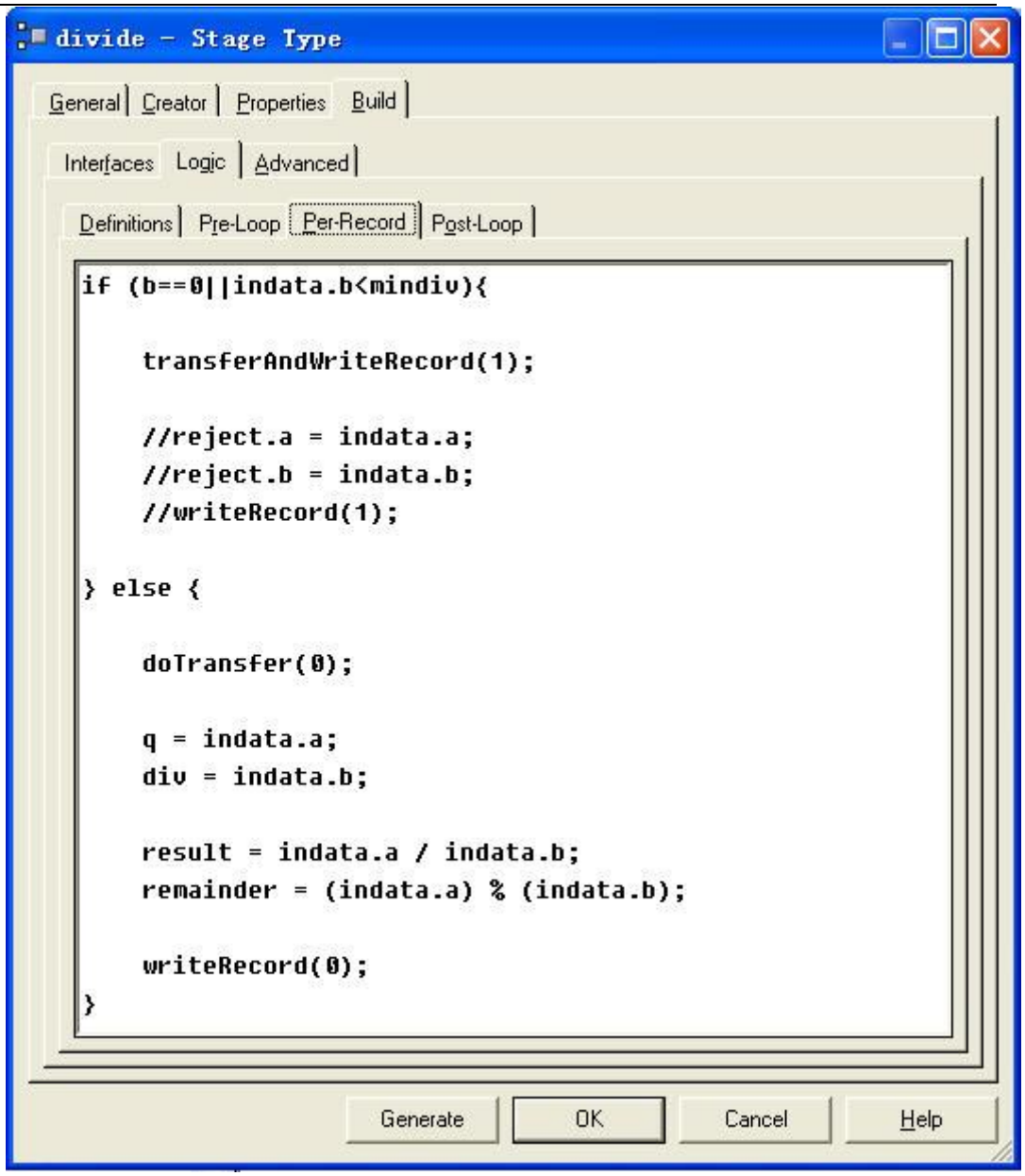
Output定义：需要定义正常的结果输出和reject 输出。如下图示：正常结果输出的Table Name选择预先定义好的表devideresult，Auto Read置为False；reject输出Table Name不需定义；



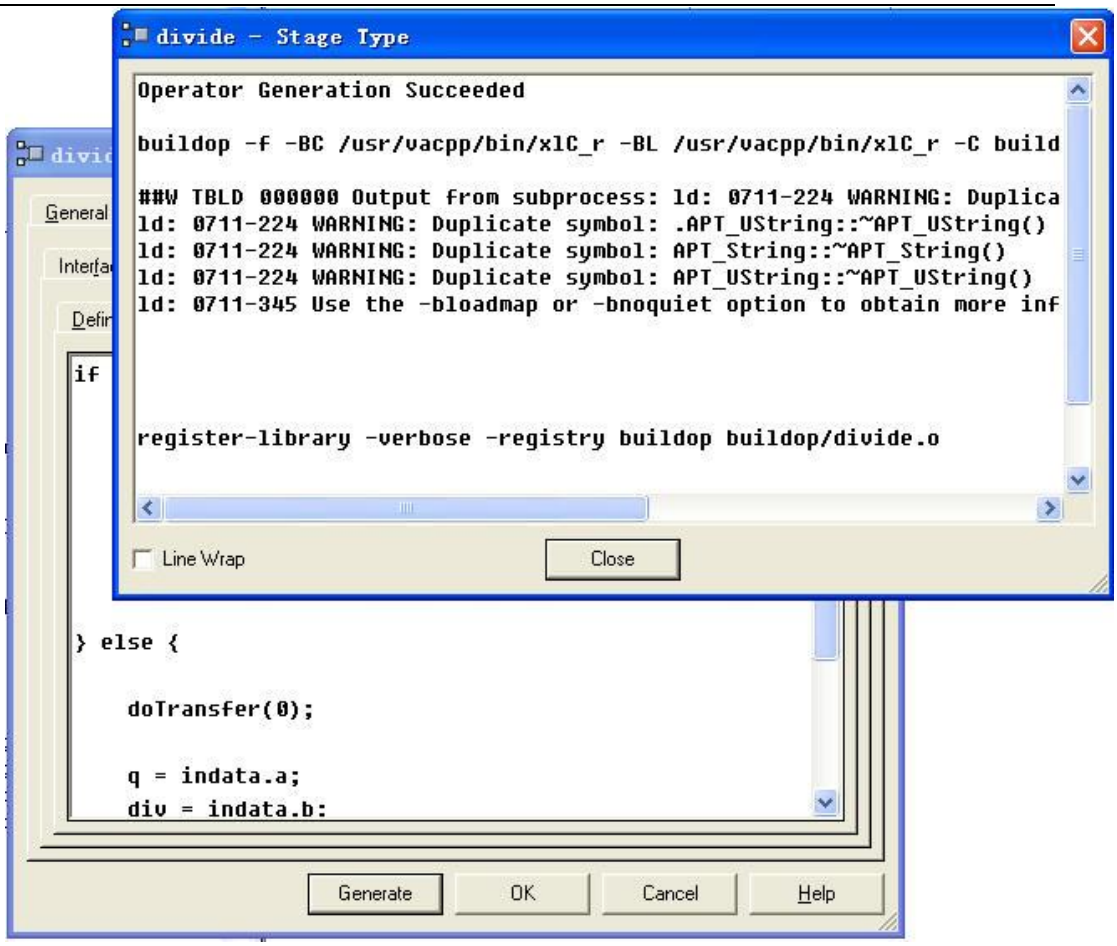
Transfer定义数据的流向：



Logic页面定义stage所要做的运算：



对于此例子中的stage，到此就定义好了一个Build Stage，最后，点击按钮“Generate”使自定义build stage生效。当定义正确时，会提示“Operator Generation Succeeded”，如下图：



生成的自定义stage会出现在desinger的工具面板中，当定义成功后，就可以向其他stage一样运用在job中了。

5.2.3. Custom Stage

针对熟悉Orchestrate的资深开发者，通过指定一个Orchestrate Operator来作为一个新的Stage type。使其能够在Parallel Job中得到使用。

5.3. 性能调优

对于大数据量的处理，DataStage的运行效率是一个十分重要的环节。

5.3.1. 优化策略

5.3.1.1. 提高读写效率

在处理大数据量的情况下，文件的读写速度往往对整个 Job 的处理时间，有很大的影响。所以，提高读写效率是 Job 优化的最基本也是最关键的一环。

由于主要是测试 Job 的读写效率，所以我们的测试案例就是读一个数据文件，然后取其

中的部分字段写到文件中去。

优化方案	优化前	优化后	效率提升（变化量/优化前）
Unix外部命令	30918 条/秒		
调整数据分部（I/O通道）	30918 条/秒		
通过组件过滤字段	30918 条/秒	88206 条/秒	185%
用RowEdit过滤字段	30918 条/秒	96742 条/秒	213%
用Complexfile读入	30918 条/秒	149950 条/秒	385%
增加读如readers	14488 条/秒	30918 条/秒	113%
改变源数据格式（定长带分隔符-> 定长不带分隔符）	40000 条/秒	46875 条/秒	17%

1. 引入 unix 外部命令（有效度：☆）

为了提高读取的速度，我们需要尽量避免读入多余的字段，我们选择了在 Sequential Stage 中加入了 Filter 选项，用 unix shell 的 cut 命令指定只读所需的字段。这样会明显提高读入的速度，但数据量增大后（估计 300MB 以上），cut 的处理速度将会明显下降，不能起到提高读取的目的。所以我们没有进行大数据量的 Cut 测试。

2. 调整数据在磁盘上的分布（有效度：★★★★）

由于数据的读写对磁盘的 I/O 依赖比较强，我们尽量将不同的数据分配到不同的文件系统，这些文件系统被 mount 到不同的磁盘上，如增量文件、全量文件、Scratch 目录、Datasets 目录，以及目标文件的存放。对于同样的数据存放到了一组磁盘上面，我们也尽量让这些数据是以分散的跨盘存储的方式读取。结果证明，数据的读取效果将有比较明显的改善。大概是在一个磁盘上读取的 2~3 倍。由于 I/O 通道的多少和在磁盘上的分布，需要有系统管理员的支持，我们没有提供有关这种优化的测试结果数据。

3. 通过组件过滤多余字段（有效度：★☆）

由于引入了 unix shell 命令以后，对于较大数据量的读取不能有较好的效果。所以，我们采取将数据先用 Sequential File Stage 读入，然后再通过 Copy Stage 对不要处理的字段进行过滤，然后再做下一步处理。这种优化虽然增加了一个组件，但 Copy Stage 的速度是很快的，所以读取的速度比过滤字段要快。

4. 通过 drop on input 过滤多余字段（有效度：★☆）

在增加了一个 Copy Stage 的基础上，对于读入文件 sequential file 中不需要抽取的字段做如下设置：edit row -> filed level -> drop on input。这样将只读入需要抽取的字段。

5. 引入 Complex File Stage 读入数据（有效度：★★★☆）

由于 Complex File Stage 能够并行读取数据，而且能够选择性的读入指定的字段，所以是我们一直希望引入的一个读入数据的组件，但由于该组件的配置比较复杂，而且对源数据的格式有一定的要求。而且其中有些功能我们目前还没有找到一些有效的方法解决。但我们在测试用该组件读取数据时，文件的读入速度有了大幅的提升。基本上可以达到正常的 Sequential File Stage 读取数据的 4~5 倍。如果其配置问题能够得到解决，那么这将是我们今后可以重点考虑的读入数据的组件。

6. 在读入文件是采用多 readers（有效度：★★★★）

无论是用 Sequential File Stage 还是用 Comples File Stage，都有一个可调的选项：Number of Readers Per Node，其缺省值为 1，它的功能是在每一个 Node 上，开 N 个读取数据的 Instances。适当调整这个参数，提高读取数据的并行度，可以明显提高读取的效率。通常，对于每一个结点，将该选项的值设置为 8~12 个为宜，如果同时有几个文件同时读入，则希望总计不要超过 12 个，这可以依据实际的情况进行调试。但该参数好像也要求数据文件是定长的。

5.3.1.2. 提高运行效率

加快Job除了数据的输入输出以外的逻辑的处理速度，灵活应用不同的Stage，或者走不同的处理流程，对整个Job的处理速度的提升，也会有比较显著的效果。

本次测试的案例为将SAACNACN档与VIP客户清单档做Lookup，找出其中属于VIP客户的账户信息。

最终测试的效率变化结果如下：

优化方案	优化前（秒）	优化后（秒）	效率提升（优化前/优化后）
改用Join方式	159	119	134%
用LookUp File Set	159	57	279%
将字段合并	57	24	237%

1. 对于数据较大的参数表，改用 Join Stage 方式（有效度：★★）

我们通常会用 Lookup Stage 做有关数据的匹配查找工作。对于 Lookup 的 Reference Link，由于其是将数据放到内存中，所以当其数据量较大的时候（超过了内存的大小），我们决定用 Join Stage 的内连接来实现相同的功能。由于我们的源数据是排序过的，所以会减少 Join Stage 的处理时间。

2. 用 LookUp File Set 代替 Sequential file（有效度：★★）

我们通常用 Sequential File Stage 来作为 Lookup Stage 的 reference link 的输入，在数据量很小的情况下，好像这种方式不会影响效率。但如果数据量比较大，接近了内存的大小，那么 DataStage 会推荐使用 Lookup file set 作为 Lookup reference 的输入。虽然生成 Lookup file set 需要单独的 Job 生成，但这对于 Lookup 效率的提升，还是很值得的。在多个 Job 需要使用相同的 Lookup file set 的情况下，这种方式应该会更加有效。

3. 将字段合并处理（有效度：★★）

我们会在数据读入的时候，过滤掉没有用的字段。而对于某些字段，在输出的结果中需要包含，但它们在处理的全程中，不会参加运算等单独的处理。我们考虑将这些连续的字段进行合并处理。这样可以减少数据的转换量。让工具只需要处理较少的需要转换的数据，对于有很多这种字段连续在一起的情况下，这种处理方式也有很好的效果。

5.3.1.3. 加强并行处理

1. 用 Cluster 方式运行（有效度：★★★★★）

在本次测试有两台机器可供测试，由于 DataStage Enterprise Parallel 的并行功能，我们将另外的一台机器也配置到了 Configure File 中。这也是它区别与其它工具的一个特点。在进行大数据量，多个 Job 的并行处理时，这将是一个最有效的解决方法。

2. 同时读取处理多个文件

以通配符的方式读取多个文件。由于测试要求对于输出的文件，需要每个分行一个单独的文件，所以我们没有对这种方式进行深度的测试。

3. 在一个 Job 中包含几个相同流程

这种处理方式就是将 Job 的内容进行拷贝，放在同一个 Job 之中。运行这个 Job 的时候会同时运行有不同输入的几个流程。但这种方式好像效果并不如几个 Job 单独运行有效。

4. 几个 Job 并行调用

将处理逻辑完全一样的几个 Job 同时运行，这是目前我们最终决定采用的并行调度方式。但每次并行的个数，还需要依据不同的 Job，做不同的分析。因为如果一次运行的 Job 过多，由于系统的资源有限，会造成并行处理的时间要远远大于同样数量的 Job 串行处理的时间。而且如果某个 Job 在一定的时间以内，没有等待到资源，那么它会因为超时而被 abort 掉。造成运行的 Job 不能全部成功的完成。每个 Job 运行起来以后，工具会在系统启动很多进程，我们分析，是否可以通过分析 process 的总量来确定不同 Job 所需要启动的最大数。但这在不同的硬件环境下，会有不同的结果。由于时间有限，我们还没有能够对每个 Job 的最大并行数进行测试。

5. 对并行的 Job 分配 Node map

为了避免出现有关几个 job 同时并行而造成进程等待超时，我们将每个 Job 都固定在几个 Node 上运行，如对于 8 个 node 的 configure file，我们将两个并行的 Job，在其中的 Stage 的 Node map 中，第一个 Job 用 1、2、3、4 个 node，第二个 Job 用剩下的 Node，这样做的效果会好很多，而且由于生成的进程少，也不容易出现 Abort 掉的结果。

5.3.1.4. 其它未进行的优化策略

1. 调整 Partition 方式

我们曾经尝试对某些 Stage 的 Partition 方式进行手工设置，但好像效果并不明显，选择 Auto 方式，让工具依据前面的 Stage 运行的 Partition 和本 Stage 的类型进行自动设定。这样效果还要相对好一些。不过对于某些特殊的，或者依据不同环境进行可以有不同选择的 Stage，也可以通过手工设定 Partition 方式来提高运行的效率。

2. 引入 DataSets

由于本次的测试案例都是相对独立的，没有必然的联系，所以本次没有采用 DataSets 存放中间结果。

3. 调整输入文件格式

我们在单项测试时，曾经对文件的格式对效率的影响进行了测试，发现变长带分隔符和定长不带分隔符的文件，在读取效率上最高。但我们本次的数据为定长带分隔符。

4. 调整环境变量

工具本身有许多的环境变量，可以调整工具运行的效率，但由于变量数目众多，而且时间有限，我们只对其中很少一部分进行了测试。

5. 使用 RCP

由于使用 RCP（Runtime Column Propagation）可以减少一些数据的转换，但同时它可能会影响一些 Stage 运行的准确性。所以我们在这次测试中关闭了 RCP 功能。

6. 针对 Transformer

如果在 Transformer 中有许多在 Input Link 中的字段，在 Output Link 中不需要输出，那么在进入 Transformer 以前，用 copy 组件将多余字段先去掉，让进入 Transformer 的数据都是有效的。这样可以节省一定的处理时间。

由于与去掉的字段的多少有关，所以还不能得出效率提升的百分比。

5.3.2. 关键问题分析

5.3.2.1. 文件读写

文件的读写，会在很大程度上，影响一个 Job 的运行效率。对于那种数据量大而且处理的逻辑相对简单的 Job，更是至关重要。因为有可能整个 Job 运行时间的 50%以上都在数据的读写上面。对于同时有多个文件读入的 Job，读入速度最慢的文件会影响拉长整个 Job 的运行时间。所以，数据以何种格式存放；存放的在什么地方；工具以何种方式读入等，都是我们需要认真考虑的地方。

5.3.2.2. 磁盘I/O

如前面所述，工具运行的效率对此依赖也是十分明显。因为当数据量比较大时，工具会十分频繁的读写磁盘，会在磁盘上存放和读取数以千计的临时文件。因此如果尽可能的增加 I/O 通道，提高对数据的读写速度，也时关键一环。

5.3.2.3. 内存限制

DataStage 会最大限度的使用系统资源，这就包括系统内存，因为工具会尽可能把数据放在内存中进行处理，在内存不够的情况下，就需要通过页面调度来存放数据。这样就会大大降低整个的处理速度。工具推荐每个 CPU 至少配备 2～3GB 的内存空间，至少这种比例，才能较好的支持工具对系统资源的消耗，在我们进行多个 Job 的并行调度的测试时，对内存的依赖就更加明显。

5.3.3. 并行度

Job 与 Job 的并行调度，一直是我们很关注的问题，经过我们反复的测试，发在 Job 能够并行的个数的多少，没有一个可以量化的标准，这与系统的软硬件环境也很有关系。大概的规则是：如果运行的 Job 逻辑简单，需要进行的运算（如排序、类型转换等）较少。则能够并行调度的 Job 数量会比较多，而且并行的处理时间会小于同样数目的 Job 串行处理的时间。

并行度最终也依赖于当时整个 DataStage 派生出来的进程数量，依据不同的硬件配置，观察当时硬件环境能够承受的进程数的一个阀值，只要保证进程的数据量不超过该阀值，则可以依据 Job 的不同，同时启用不同的并发数。

5.3.4. 处理建议

5.3.4.1. 提高读取速度

1. 对于只有少数字段参加处理流程的文件可以考虑采用 Complex 进行数据读取。但需要对 Table Definition 做改造。
2. 如果采用定长不带分隔符的数据格式，既可以避免字段中带分隔符的情况不能处理，也可以提供数据的读取速度。

5.3.4.2. 合理分配数据存储空间

1. 按照数据量和数据的种类进行分类，避免频繁读取的数据被放在同一个磁盘空间上。为频繁读取的数据分配更多的 I/O 资源。
2. 对于工具 Scratch 目录，尽量分配较多的 I/O 通道。

5.3.4.3. 加强Stage的调优

1. 需要在以后的 Job 设计、测试过程中尽可能多的对每一个 Stage 的各项参数，如 Partition 方式，Buffer 类型，运行的 Node 数等，进行调优。

5.3.4.4. 合理工具参数设置

1. 对于工具的众多环境变量，最好能够在理解其含义的情况下，对效率影响比较大的一些，依据系统的实际情况进行设置。

5.3.4.5. 调整操作系统参数

1. 工具不能控制一些操作系统级的参数，所以需要我们依据工具对系统的内存、CPU、I/O 通道的依赖。调整一些系统的参数，对操作系统本身进行优化。

5.3.5. 其它

5.3.5.1. 对输出结果排序

1. 由于本次的测试没有对输出文件的排序做要求，所以我们在测试过程中没有考虑最后的输出文件的排序。
2. 我们仅仅对一个大小约为 80 万条记录的输出数据进行了排序处理，发现这种规模的排序，对整个 Job 的处理时间（约 5 分钟）没有太大影响。

5.3.5.2. Complex file的配置

由于 Complex File Stage 主要是对于 EBCDIC 码等比较复杂的文件进行读取，所以对于普通的 Sequential file，我们如果用 Complex File Stage 读取，还存在一些问题。如果决定要采取这种方式读入数据，还需要对这个 Stage 的功能进行进一步的理解。

参考厂商的建议，我们也不推荐用该 Stage 作为对 Sequential file 的读取组件。

5.3.6. 机器的对称性

我们在运行并行的 Job 调度时，发现有这样一种情况，我们每次调用三个，甚至两个 Job 时，都有可能会出现一个 Job 因为无法在规定时间内等到资源而被 Abort 掉，查看日志发现问题出现在从机 xmkf_A22 上，和 Support 联系后，他们认为我们的两台机器在配置上不对称也是我们出现这个问题的原因。

5.3.7. 并行调度测试说明：

按照Ascential的工程师给我们的建议，当一个Job能够比较彻底的消耗系统的资源的时候，我们通常会串行的调度我们的Job。而如果盲目的对几个同样的Job进行并行的调度，那么从时间上说，可能不会节省多少，而且有可能会造成整个Job运行的稳定性降低。

但由于部分Job在运行过程中的资源的消耗不是很彻底。于是我们想测试一下多少个Job的运行会消耗掉所有的系统资源。而且我们在Job能够顺利执行成功的情况下能够做多少Job的并行。

5.3.7.1. 测试结果

由于我们在测试中发现如果我们对于每一个Job都同时在全部的 8 个结点上运行，当我们的并行度增加到 3 个的时候（此时的Job没有进行流程的改造），会有一个Job因为超时而

失败。所以我们准备用对不同的Job指定不同的Node来运行的方法来进行并行的测试，结果是虽然运行的时间没有得到提升，但稳定行得到了提高，最大可以达到同时运行 16 个Job。

测试	每个Job的Node数	并行的Job数	运行时间
1	有 4 个Job为 3 个node；有 2 个Job为 2 个Node	6	8′ 20″
2	每个Job为 2 个Node	8	12′
3	每个Job为 1 个Node	16	21′ 8″
4	每个Job为 8 个Node	1	35″

6. 开发经验技巧汇总

下面就说一说使用中困惑过的一些问题：

1．Job 的粒度。一套 ETL 过程中，含有多个步骤，在设计过程中，到底是粗化一些,用少而复杂的 job 实现,还是细化一些,用多而简单的 job 实现更好呢？个人认为，比较细的粒度更有利于程序的开发。在开发初期，表面看来细化的 job 比较繁琐，但在项目后期的测试阶段，细化的 job 可以更准确的定位错误并易于修改。

2．并行和串行。当到了开发后期，准备把多个 job 连接起来，我们就会发现，能否将多个 job 并行成为 ETL 效率的关键，而这个因素在设计初期往往被忽略。ETL 中可能会涉及多个数据源的多个表，而多个 job 也可能会形成对某个数据源以及其中的某个表的争用。当数据源争用时，会影响 ETL 的执行效率。当表争用无法解决的时候，就只能使用串行。而一个好的结构流程设计，可以极大的减少这种争用，从而提高 ETL 的效率。

3．要将 Datastage 与外部 code 相结合。Datastage 并不是独立运行的开发工具，它需要外部控制程序为载体，才可以进行良好的客户操作。而 Datastage 也不是万能的，简单的说，它只是 sql 语言的一个可视化载体。因此，有一些功能，并不一定要在 Datastage 中实现，而应该放到外部程序中，以 sql code 的形式完成，以保证整个程序的稳定性，安全性。

上面是一些大方向的问题，在实际中会有很多烦琐的小问题，我也尽量的列举一些：

1．字符集：output 和 input 中的字符集都设置为 none,是一个不错的选择。至少可以保证程序运行不会因为乱码 abort。

2．文本中的列分隔符无法设置为三位，从理论上讲，只有三位分割符才可以保证程序不会将乱码辨认为分隔符，这是 Datastage 的一个缺陷。

3．在使用自定义 sql 前，需要使用非自定义形式手工配置好所需要的表，然后再切回自定义格式，如果直接写自定义 sql,将导致 Datastage 无法辨别表名，从而导致错误，这应该是一个 bug。

4．保持配置一个 input 或 output，就 view data 一下的习惯，不要等到 run 时再回头找 error.

5．Input 中尽量不要使用 insert or update 之类的选项，它和 insert only 的差别是巨大的。使用 insert or update 等选项，相当于使用游标，逐条进行对比，每 insert 一条，都要先做一次全表扫描，其速度是可想而知的。如果必须要实现这种功能，应使用其他方法，如先 delete 目标表中所有与源表重复的记录，然后再从源表中 insert 数据.

6．Date 型数据是比较麻烦的，因为 Datastage 中的日期格式为 timestamp,当然你也可以把它的日期格式更改为 date 型，但经常会出现错误。对于 oracle 数据库源表和目标表，不需要对 date 型数据做任何转换，直接使用默认即可，但对于 informix 等一些数据库，则需要使用 oconv，iconv 函数进行转换，并在 output 中相应的修改 output sql 中的日期格式。具体用法可以去网上或查 datastage 帮助。

7．只要你保证 input 和 output 时数据类型和长度不会有问题，在两者之间的这一段过程中，Datastage 中的数据类型和长度是可以随意更改的，也可以随意增加自定义列。

8.字符串中的半角空格需要用 trimb,而不是 trim 函数,但这点往往被忽略。其他的情况还可能有半角中文等，所以字符串，长度，字符集，这几者之间经常会导致 Datastage 产生错误，所以应尽量保证 insert 前的字符串长度要小于 insert 后的字符串长度，而你看到的 insert 前的字符串长度并不一定就是它在 Datastage 中真正的长度，所以使用 trimb 函数在 input sql 中做一下限制，才是最稳妥的方法。