

---



---

## How Do I Load Transaction Data?

### Scenario

Your company records all its transactions as they occur, resulting in inserts, updates, and deletes, in a flat file called `record.csv`. These transactions need to be processed in the exact order they were stored. For example, if an order was first placed, then updated, then cancelled and re-entered, this transaction must be processed exactly in the same order.

An example data set of the source file `record.csv` is defined as:

Action,DateTime,Key,Name,Desc

```
I,71520031200,ABC,ProdABC,Product ABC
I,71520031201,CDE,ProdCDE,Product CDE
I,71520031202,XYZ,ProdXYZ,Product XYZ
U,71620031200,ABC,ProdABC,Product ABC with option
D,71620032300,ABC,ProdABC,Product ABC with option
I,71720031200,ABC,ProdABC,Former ProdABC reintroduced
U,71720031201,XYZ,ProdXYZ,Rename XYZ
```

You want to load the data into a target table such as the following:

```
SRC_TIMESTA KEY NAME DESCRIPTION
-----
71520031201 CDE ProdCDE Product CDE
71720031201 XYZ ProdXYZ Rename XYZ
71720031200 ABC ProdABC Former ProdABC reintroduced
```

### Solution

Warehouse Builder enables you to design ETL logic and load the data in the exact temporal order in which the transactions were stored at source. To achieve this result, you design a mapping that orders and conditionally splits the data before loading it into the target. Then, you configure the mapping to generate code in row based operating mode. In row based operating mode Warehouse Builder generates code to process the data row by row using if-then-else constructions, as shown in the example below:

```
CURSOR
  SELECT
    "DATETIME$1"
  FROM
    "JOURNAL_EXT"
  ORDER BY "JOURNAL_EXT"."DATETIME" ASC
LOOP
  IF "ACTION" = 'I' THEN
    INSERT this row
```

```

ELSE
  IF "ACTION" = 'U' THEN
    UPDATE this row
  ELSE
    BEGIN
      DELETE FROM
        "TARGET_FOR_JOURNAL_EXT"
    END LOOP;

```

This ensures that all consecutive actions are implemented in sequential order and the data is loaded in the order in which the transaction was recorded.

### Case Study

This case study shows you how to create ETL logic to load transaction data in a particular order using Warehouse Builder.

**Step 1: Import and Sample the Source Flat File, *record.csv*** In this example, the flat file *record.csv* stores all transaction records and a timestamp. Import this flat file from your source system using the Warehouse Builder Import Metadata Wizard. Proceed to define the metadata for the flat file in Warehouse Builder using the Flat File Sample Wizard.

**Note:** You can replace this flat file with a regular table if your system is sourced from a table. In this case, skip to Step 3.

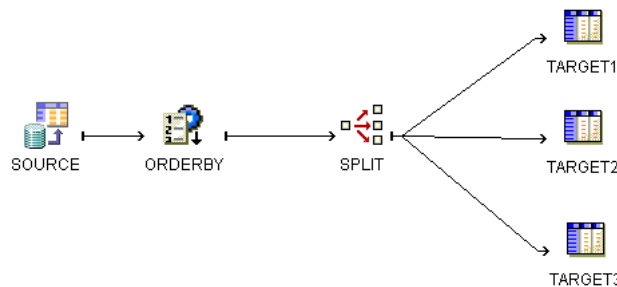
**Step 2: Create an External Table** To simplify using the sampled flat file object in a mapping in Warehouse Builder, create an external table (JOURNAL\_EXT) using the External Table wizard, based on the flat file imported and sampled in Step 1.

The advantage of using an external table instead of a flat file is that it provides you direct SQL access to the data in your flat file. Hence, there is no need to stage the data.

### Step 3: Design the Mapping

In this mapping, you move the transaction data from an external source, through an operator that orders the data, followed by an operator that conditionally splits the data before loading it into the target table.

**Figure 5–1 ETL Design**



The sorter operator enables you to order the data and process the transactions in the exact order in which they were recorded at source. The splitter operator enables you to conditionally handle all the inserts, updates, and deletes recorded in the source data by defining a split condition that acts as the if-then-else constraint in the generated code. The data is conditionally split and loaded into the target table. In this mapping, the same target table is used three times to demonstrate this conditional loading. The

mapping tables TARGET 1, TARGET 2, and TARGET 3 are all bound to the same repository table TARGET. All the data goes into a single target table.

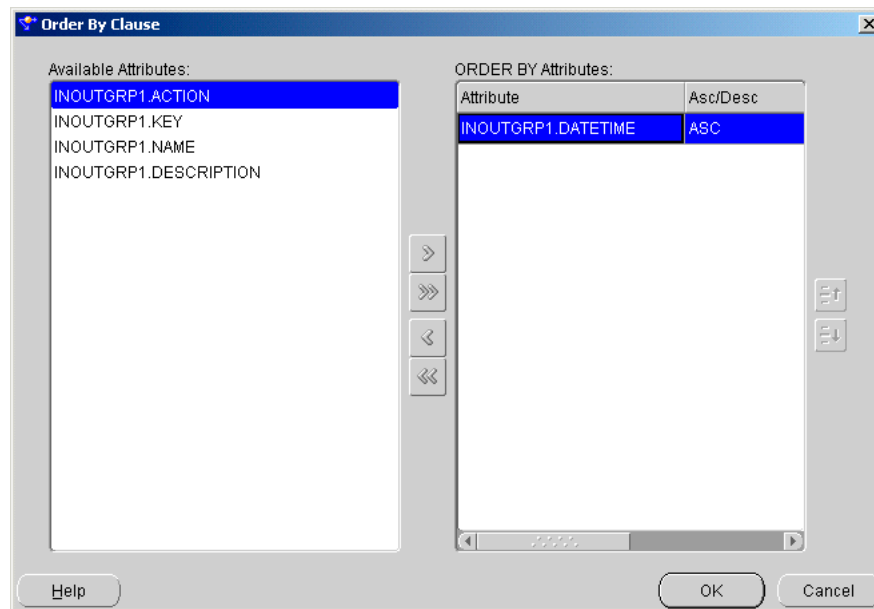
The following steps show you how to build this mapping.

**Step 4: Create the Mapping** Create a mapping called LOAD\_JOURNAL\_EXT using the New Mapping Wizard. Warehouse Builder then opens the Mapping Editor where you can build your mapping.

**Step 5: Add an External Table Operator** Drag and drop a mapping external table operator onto the mapping editor and bind it to the external table JOURNAL\_EXT.

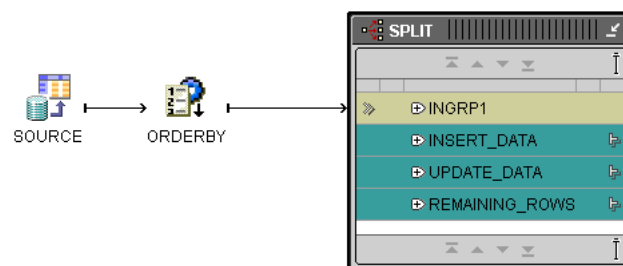
**Step 6: Order the Data** Add the Sorter operator to define an order-by clause that specifies the order in which the transaction data must be loaded into the target. [Figure 5-2](#) shows you how to order the table based on the timestamp of the transaction data in ascending order.

**Figure 5-2 Order By Clause Dialog**



**Step 7: Define a Split Condition** Add the Splitter operator to conditionally split the inserts, updates, and deletes stored in the transaction data. This split condition acts as the if-then-else constraint in the generated code.

**Figure 5-3 Adding the Splitter Operator**



Define the split condition for each type of transaction. For outgroup INSERT\_DATA define the split condition as `INGRP1.ACTION = 'I'`. For UPDATE\_DATA, define

the split condition as `INGRP1.ACTION = 'U'`. In Warehouse Builder, the splitter operator contains a default group called `REMAINING_ROWS` that automatically handles all Delete ('D') records.

**Step 8: Define the Target Tables** Use the same repository target table three times for each type of transaction: one for `INSERT_DATA`, one for `UPDATE_DATA`, and one for the `REMAINING_ROWS`.

**Step 9: The Solution: Configure the Mapping `LOAD_JOURNAL_EXT`** After you define the mapping, you need to configure the mapping to generate code. Because the objective of this example is to process the data strictly in the order it was stored, you must select row based as the default operating mode. In this mode, the data is processed row by row and the insert, update, and delete actions on the target tables take place in the exact order in which the transaction was recorded at source.

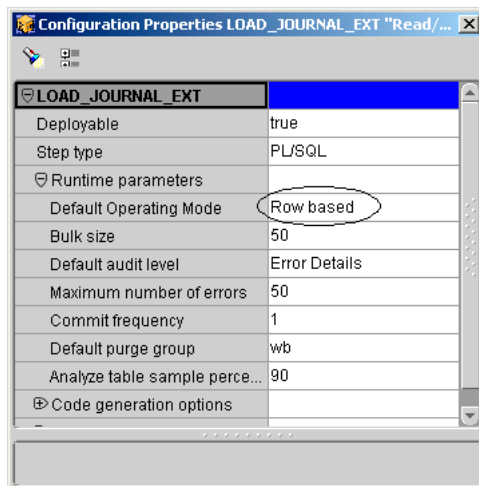
Do not select set-based mode as Warehouse Builder then generates code that creates one statement for all insert transactions, one statement for all update transactions, and a third one for all delete operations. The code then calls these procedures one after the other, completing one action completely before following up with the next action. For example, it first handles all inserts, then all updates, and then all deletes.

To configure the mapping for loading transaction data:

From the Warehouse Builder navigation tree, right-click the `LOAD_JOURNAL_EXT` mapping and select **Configure**.

Expand the Runtime Parameters node and locate the Default Operating Mode parameter.

**Figure 5-4 Configuration Properties for mapping `Load_Journal_Ext`**



Set this parameter to **Row based** as the default operating mode as shown.

In this example, accept the default value for all other parameters. Validate the mapping before generating the code.

**Step 10: Generate Code** After you generate a mapping, Warehouse Builder displays the results in the Generation Results dialog.

When you inspect the code, you will see that Warehouse Builder implements all consecutive actions in row based mode. This means that the data is processed row by row and Warehouse Builder evaluates all conditions in sequential order using

---

if-then-else constructions, as shown on page 5-2. The resulting target table thus maintains the sequential integrity of the transactions recorded at source.

