HITACHI
Inspire the Next

Pentaho Data Integration

Database Operations

James O’Reilly
Hitachi Vantara Global Learning
Date

Module Objectives

When you complete this module, you should be able to:

- Conduct various database operations that include:
  - Configure a Database Connection
  - Conduct Standard database operations

    - Create / Read

    - Update

    - Insert

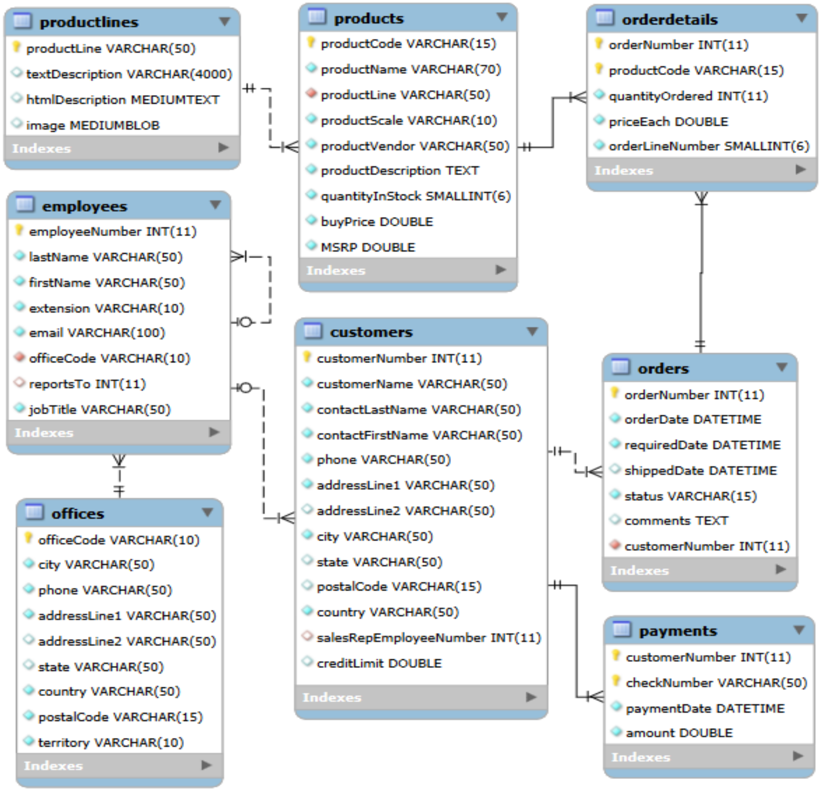
    - Delete
  - Implement a Type II Slowly Changing Dimension

Steel Wheels Inc

Steel Wheels buys collectable model cars, trains, trucks, etc, from manufacturers and sells to distributors across the globe.

Overview of Steel Wheels Database

HITACHI
Inspire the Next



productlines
- productLine VARCHAR(50)
- textDescription VARCHAR(4000)
- htmlDescription MEDIUMTEXT
- image MEDIUMBLOB
- Indexes

products
- productCode VARCHAR(15)
- productName VARCHAR(70)
- productLine VARCHAR(50)
- productScale VARCHAR(10)
- productVendor VARCHAR(50)
- productDescription TEXT
- quantityInStock SMALLINT(6)
- buyPrice DOUBLE
- MSRP DOUBLE
- Indexes

orderdetails
- orderNumber INT(11)
- productCode VARCHAR(15)
- quantityOrdered INT(11)
- priceEach DOUBLE
- orderLineNumber SMALLINT(6)
- Indexes

employees
- employeeNumber INT(11)
- lastName VARCHAR(50)
- firstName VARCHAR(50)
- extension VARCHAR(10)
- email VARCHAR(100)
- officeCode VARCHAR(10)
- reportsTo INT(11)
- jobTitle VARCHAR(50)
- Indexes

customers
- customerNumber INT(11)
- customerName VARCHAR(50)
- contactLastName VARCHAR(50)
- contactFirstName VARCHAR(50)
- phone VARCHAR(50)
- addressLine1 VARCHAR(50)
- addressLine2 VARCHAR(50)
- city VARCHAR(50)
- state VARCHAR(50)
- postalCode VARCHAR(15)
- country VARCHAR(50)
- salesRepEmployeeNumber INT(11)
- creditLimit DOUBLE
- Indexes

orders
- orderNumber INT(11)
- orderDate DATETIME
- requiredDate DATETIME
- shippedDate DATETIME
- status VARCHAR(15)
- comments TEXT
- customerNumber INT(11)
- Indexes

offices
- officeCode VARCHAR(10)
- city VARCHAR(50)
- phone VARCHAR(50)
- addressLine1 VARCHAR(50)
- addressLine2 VARCHAR(50)
- state VARCHAR(50)
- country VARCHAR(50)
- postalCode VARCHAR(15)
- territory VARCHAR(10)
- Indexes

payments
- customerNumber INT(11)
- checkNumber VARCHAR(50)
- paymentDate DATETIME
- amount DOUBLE
- Indexes

Table	Description
CUSTOMERS	Steel Wheels' customers
EMPLOYEES	All employee information, organization structure such as who reports to whom
PRODUCTS	Products sold by Steel Wheels
PRODUCTLINES	List of product line categories.
OFFICES	Steel Wheels' offices
ORDERS	Information about sales orders
ORDERDETAILS	Sales order line items for each sales order.
PAYMENTS	Payments made by customers based on their accounts.

Topics

Database Connections

Write / Read to / from a Table

Insert / Update

Delete

Slowly Changing Dimensions

Database Connections

- Working file-based
  - DB connections are specific to job or transformation

- Working repository-based
  - DB connections are stored centrally in repository
  - Defined connections are readily available to transformations and jobs
  - DB connections can be secured

Database Connections

- **JDBC (Native) Access**
  - Database drivers must be added to

    - Spoon: data-integration/lib

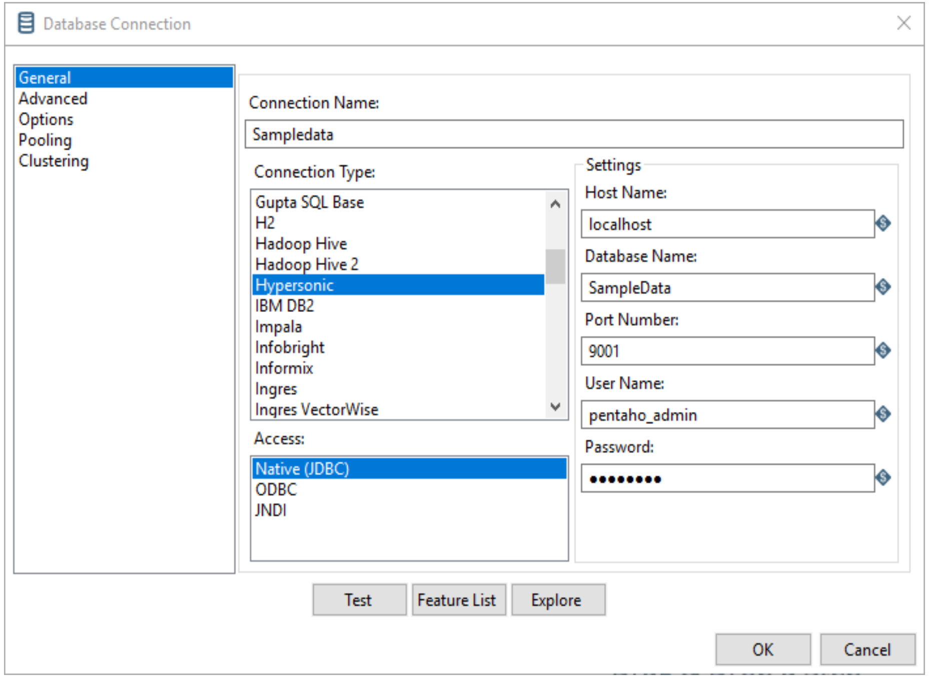
    - Pentaho Server: /pentaho/server/data-integration-server/tomcat/webapps/pentaho/WEB-INF/lib/

  - Dialect-specific SQL support for listed data sources
  - Generic database connection available for non-listed data sources

    - Generic SQL dialect used for SQL-92 compliant data sources

Demonstration – Database Connections

Demonstration - Connect to Database



Database Connection

Connection Name: Sampledata

Connection Type: Hypersonic

Access: Native (JDBC)

Settings

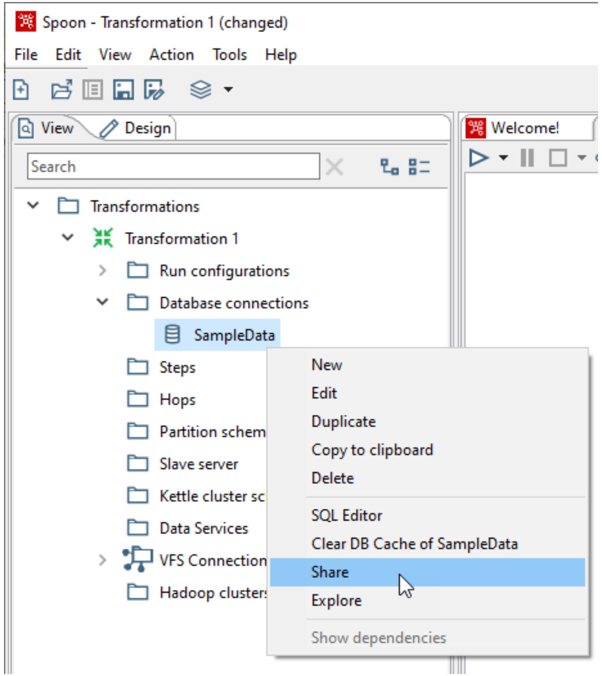
Host Name: localhost

Database Name: SampleData

Port Number: 9001

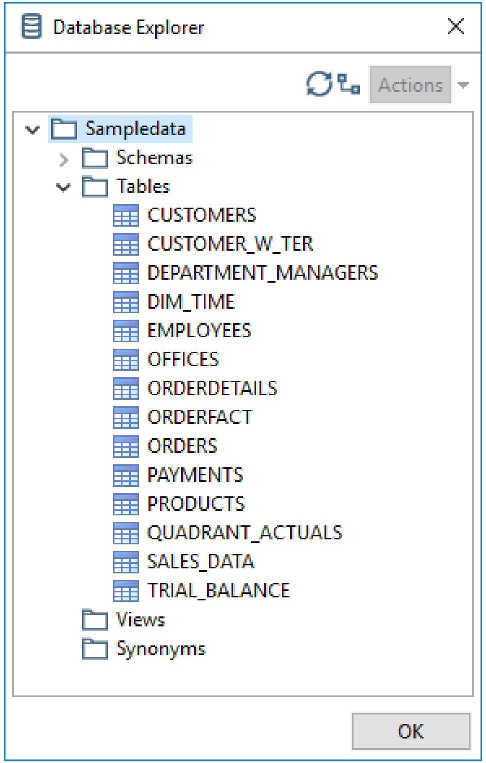
User Name: pentaho_admin

Password: ••••••••

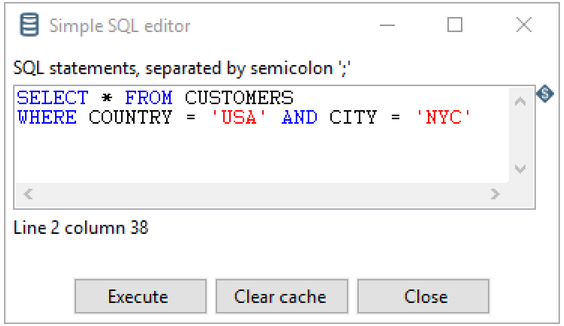


Spoon - Transformation 1 (changed)

Demostration - Connect to Database



- Sampledata
  - Schemas
  - Tables
    - CUSTOMERS
    - CUSTOMER_W_TER
    - DEPARTMENT_MANAGERS
    - DIM_TIME
    - EMPLOYEES
    - OFFICES
    - ORDERDETAILS
    - ORDERFACT
    - ORDERS
    - PAYMENTS
    - PRODUCTS
    - QUADRANT_ACTUALS
    - SALES_DATA
    - TRIAL_BALANCE
  - Views
  - Synonyms



SQL statements, separated by semicolon ';'

```
SELECT * FROM CUSTOMERS
WHERE COUNTRY = 'USA' AND CITY = 'NYC'
```

Line 2 column 38

Client Tools:

- RazorSQL
- TOAD
- Navicat Premium
- Squirrel (Open source)
- DBeaver (Open Source)

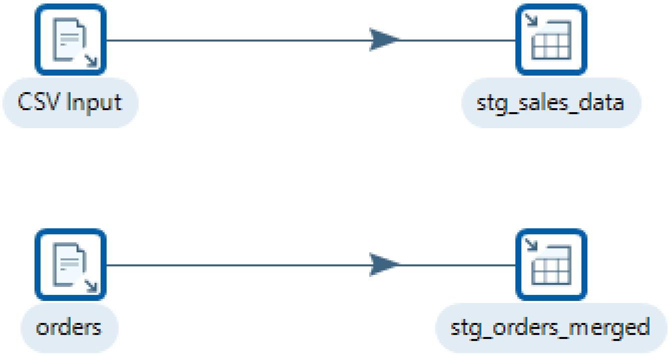
- https://en.wikipedia.org/wiki/Comparison_of_database_tools

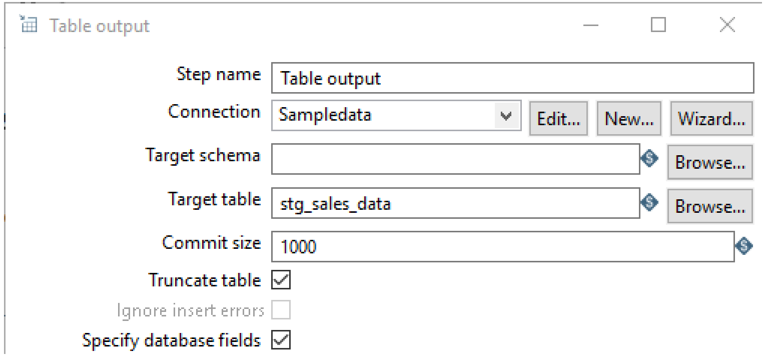
Database Operations

Lab 1 – Write to a Table

Lab 1 - Write to Table

- If you work with databases, one of the main objectives will be to extract, load and transform your data. Steel Wheels has several data sources that require loading into a database to discover, cleanse, conform, enrich and validate the data for reports.





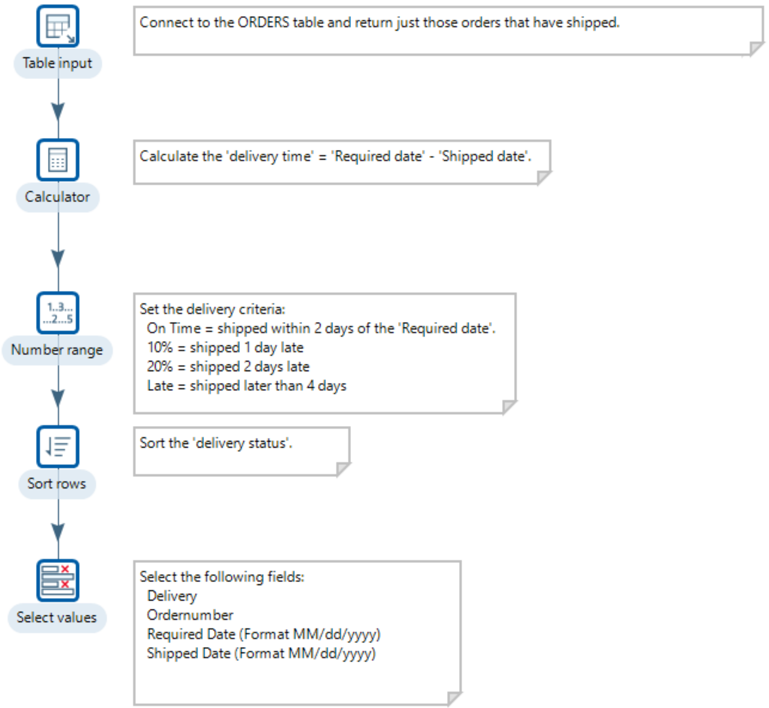
Step name: Table output
Connection: Sampledata
Target schema:
Target table: stg_sales_data
Commit size: 1000
Truncate table ☑
Ignore insert errors □
Specify database fields ☑

HITACHI
Inspire the Next

Lab 2 – Read from a Table

Lab 2 - Reading from a Database Table

- So far you have just connected to a database..



Connect to the ORDERS table and return just those orders that have shipped.

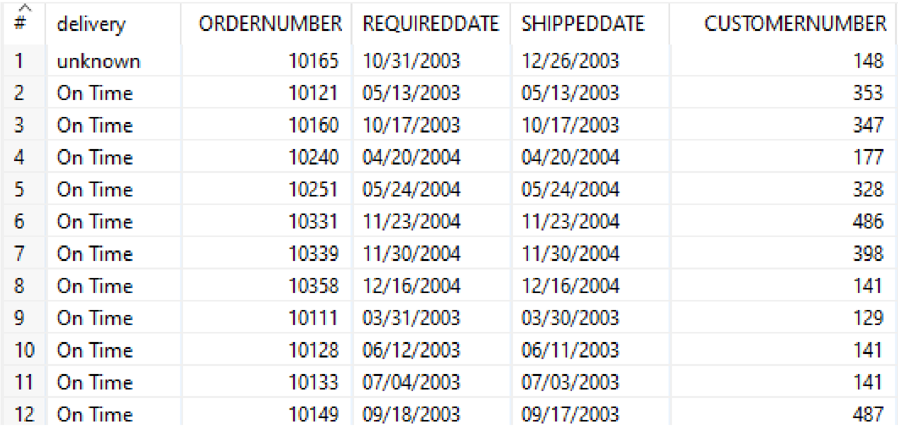
Calculate the 'delivery time' = 'Required date' - 'Shipped date'.

Set the delivery criteria:
On Time = shipped within 2 days of the 'Required date'.
10% = shipped 1 day late
20% = shipped 2 days late
Late = shipped later than 4 days

Sort the 'delivery status'.

Select the following fields:
Delivery
Ordernumber
Required Date (Format MM/dd/yyyy)
Shipped Date (Format MM/dd/yyyy)

Steel Wheels wish to produce a report
Tracking the 'Delivery Status' of each order.



#	delivery	ORDERNUMBER	REQUIREDDATE	SHIPPEDDATE	CUSTOMERNUMBER
1	unknown	10165	10/31/2003	12/26/2003	148
2	On Time	10121	05/13/2003	05/13/2003	353
3	On Time	10160	10/17/2003	10/17/2003	347
4	On Time	10240	04/20/2004	04/20/2004	177
5	On Time	10251	05/24/2004	05/24/2004	328
6	On Time	10331	11/23/2004	11/23/2004	486
7	On Time	10339	11/30/2004	11/30/2004	398
8	On Time	10358	12/16/2004	12/16/2004	141
9	On Time	10111	03/31/2003	03/30/2003	129
10	On Time	10128	06/12/2003	06/11/2003	141
11	On Time	10133	07/04/2003	07/03/2003	141
12	On Time	10149	09/18/2003	09/17/2003	487

Overview of Metadata Injection

- Metadata injection refers to the dynamic passing of metadata to PDI transformations at run time in order to control complex data integration logic.

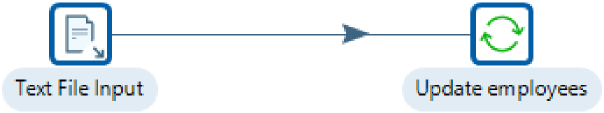
- The metadata (from the data source, a user defined file, or an end user request) can be injected on the fly into a transformation template, providing the “instructions” to generate actual transformations.

- This enables teams to drive hundreds of data ingestion and preparation processes through just a few actual transformations, heavily accelerating time to data insights and monetization.

Lab 3 – Update Records

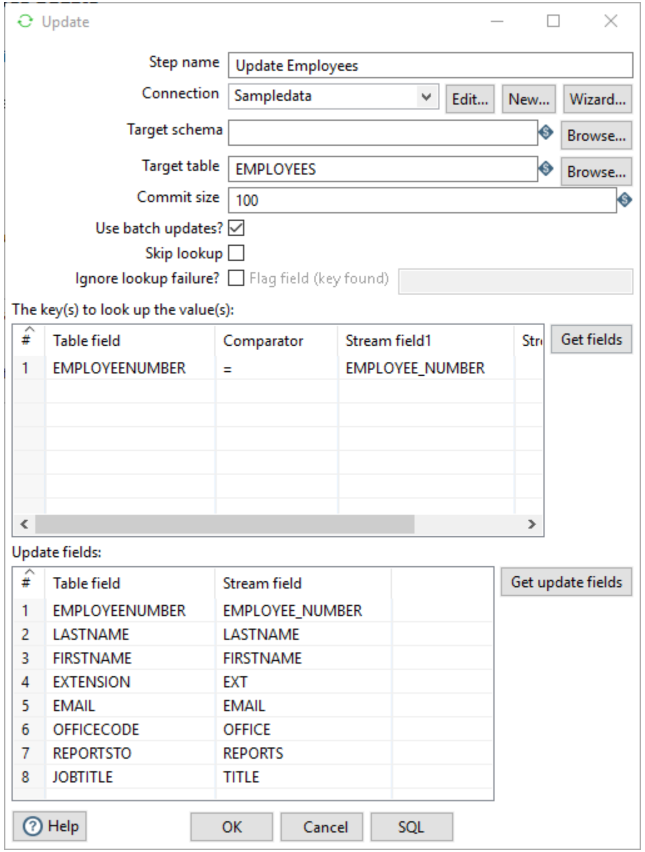
Lab 3 – Update Records

```
EMPLOYEE_NUMBER,LASTNAME,FIRSTNAME,EXT,EMAIL,OFFICE,REPORTS,TITLE
1002,Murphy,Diana,x5800,dmurphy@classicmodelcars.com,1,1000,CEO
1102,Bondur,Gerard,x5408,gbondur@classicmodelcars.com,4,1056,Regional Sales Manager (EMEA)
```



The stream fields are mapped to the database table by ensuring:
EMPLOYEE_NUMBER = EMPLOYEENUMBER

Ensure all the Field mappings are correct



Step name: Update Employees
Connection: Sampledata
Target schema:
Target table: EMPLOYEES
Commit size: 100
Use batch updates? ☑
Skip lookup ☐
Ignore lookup failure? ☐ Flag field (key found)

The key(s) to look up the value(s):

#	Table field	Comparator	Stream field1	Str
1	EMPLOYEENUMBER	=	EMPLOYEE_NUMBER	

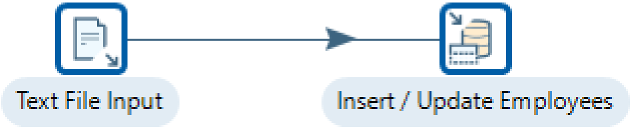
Update fields:

#	Table field	Stream field
1	EMPLOYEENUMBER	EMPLOYEE_NUMBER
2	LASTNAME	LASTNAME
3	FIRSTNAME	FIRSTNAME
4	EXTENSION	EXT
5	EMAIL	EMAIL
6	OFFICECODE	OFFICE
7	REPORTSTO	REPORTS
8	JOBTITLE	TITLE

Lab 4 – Insert / Update Records

Lab 4 – Insert / Update Records

```
EMPLOYEE_NUMBER, LASTNAME, FIRSTNAME, EXT, EMAIL, OFFICE, REPORTS, TITLE
1188,Firrelli,Julianne,x2174,jfirrelli@classicmodelcars.com,2,1143,Sales Manager
1619,King,Tom,x6324,tking@classicmodelcars.com,6,1088,Sales Rep
1810,Lundberg,Anna,x910,alundberg@classicmodelcars.com,2,1143,Sales Rep
1811,Schulz,Chris,x951,cschulz@classicmodelcars.com,2,1143,Sales Rep
```

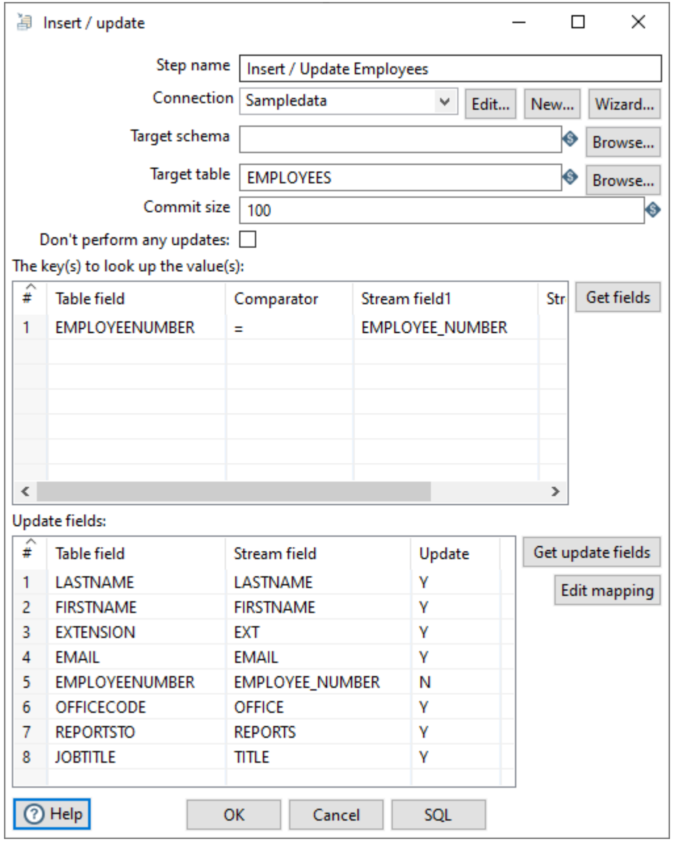


Text File Input

Insert / Update Employees

The stream fields are mapped to the database table by ensuring:
EMPLOYEE_NUMBER = EMPLOYEENUMBER

Ensure all the Field mappings are correct



Insert / update

Step name: Insert / Update Employees

Connection: Sampledata

Target schema:

Target table: EMPLOYEES

Commit size: 100

Don't perform any updates: □

The key(s) to look up the value(s):

#	Table field	Comparator	Stream field1	Str
1	EMPLOYEENUMBER	=	EMPLOYEE_NUMBER	

Update fields:

#	Table field	Stream field	Update
1	LASTNAME	LASTNAME	Y
2	FIRSTNAME	FIRSTNAME	Y
3	EXTENSION	EXT	Y
4	EMAIL	EMAIL	Y
5	EMPLOYEENUMBER	EMPLOYEE_NUMBER	N
6	OFFICECODE	OFFICE	Y
7	REPORTSTO	REPORTS	Y
8	JOBTITLE	TITLE	Y

Lab 5 – Delete Records

Delete Columns / Rows

- Sometimes you might have to delete data from a table. If the operation to do it is simple, for example:

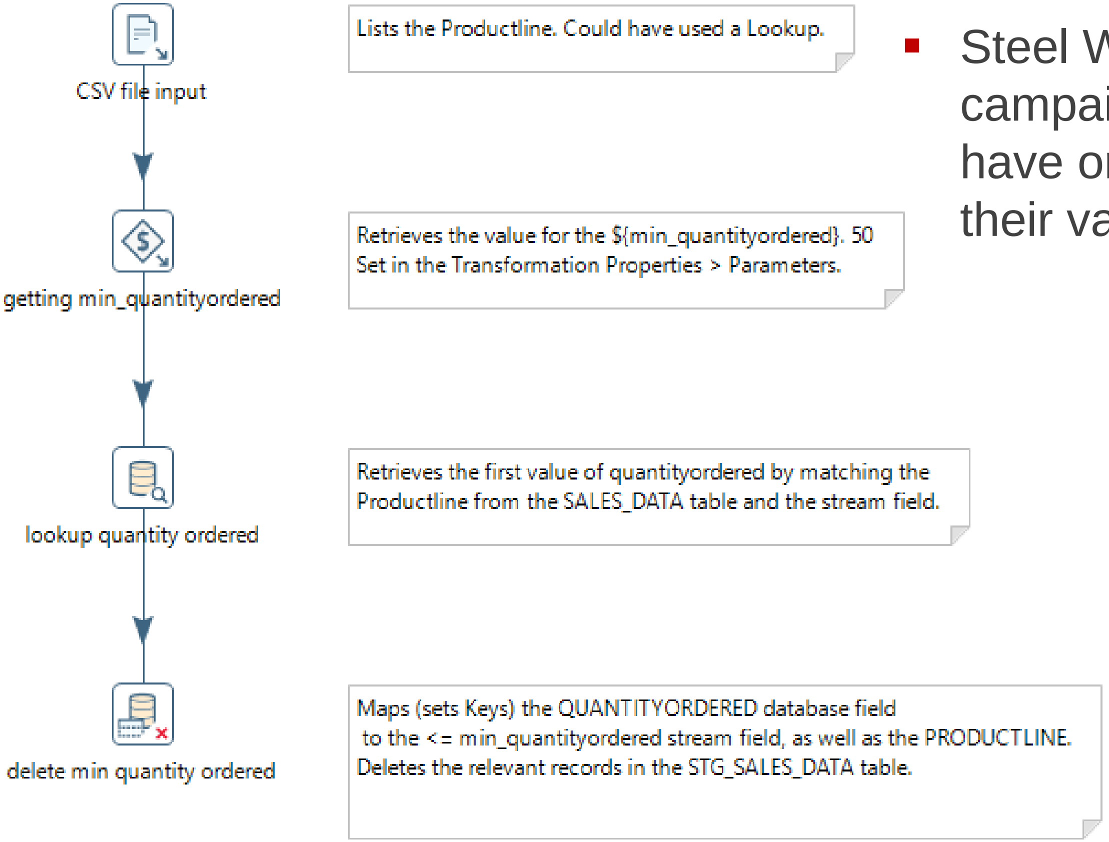
```
DELETE FROM LOG_TABLE WHERE VALID='N'
```

Or

```
DELETE FROM TMP_TABLE
```

- You could simply execute it by using an SQL job entry or an Execute SQL script step. If you face the second of the above situations, you can even use a Truncate table job entry.
- For more complex situations, you should use the Delete step.

Lab 5 – Delete Records



Lists the Productline. Could have used a Lookup.

Retrieves the value for the ${min_quantityordered}. 50
Set in the Transformation Properties > Parameters.

Retrieves the first value of quantityordered by matching the Productline from the SALES_DATA table and the stream field.

Maps (sets Keys) the QUANTITYORDERED database field
to the <= min_quantityordered stream field, as well as the PRODUCTLINE.
Deletes the relevant records in the STG_SALES_DATA table.

- Steel Wheels are launching a campaign, focusing on Customers who have ordered more than 50 of each of their various Productlines.

Slowly Changing Dimensions

Slowly Changing Dimensions

- SCD management methodologies referred to as Type 0 through 6. Type 6 SCDs are also sometimes called Hybrid SCDs.

- A type 1 slowly changing dimension is the most basic one and doesn’t require any special modelling or additional fields.
  SCD type 1 columns just get **overwritten** with new values when they come into the data warehouse.

- The Type 2 method tracks historical data by creating multiple records for a given natural key in the dimensional tables with separate surrogate key (technical key) and/or different version numbers. With Type 2, we have unlimited history preservation as a new record is inserted each time a change is made.

Type I SCD

- **Type 1** - Overwriting the old value. In this method, no history of dimension changes is kept in the database. The old dimension value is simply overwritten with the new one. This type is easy to maintain and is often use for data which changes are caused by processing corrections (e.g. removal special characters, correcting spelling errors).

Before the change:

Customer_ID	Customer_Name	Customer_Type
1	Cust_1	Corporate

After the change:

Customer_ID	Customer_Name	Customer_Type
1	Cust_1	Retail

Type II SCD

- **Type 2** - Creating a new additional record. In this methodology, all history of dimension changes is kept in the database. You capture attribute change by adding a new row with a new surrogate key (technical key) to the dimension table. Both the prior and new rows contain as attributes the natural key (or another durable identifier).

- Also ‘current version' and ‘effective date' columns are used in this method. There could be only one record with current version set to ‘1‘; incrementing everytime a new record is inserted.

- For 'effective date' columns, i.e. start_date and end_date, the end_date for current record usually is set to value 9999-12-31. Introducing changes to the dimensional model in type 2 could be very expensive database operation so it is not recommended to use it in dimensions where a new attribute could be added in the future.

Type II SCD

Before the change:

Customer_ID	Customer_Name	Customer_Type	Start_Date	End_Date	Version
1	Cust_1	Corporate	22-07-2010	31-12-9999	1

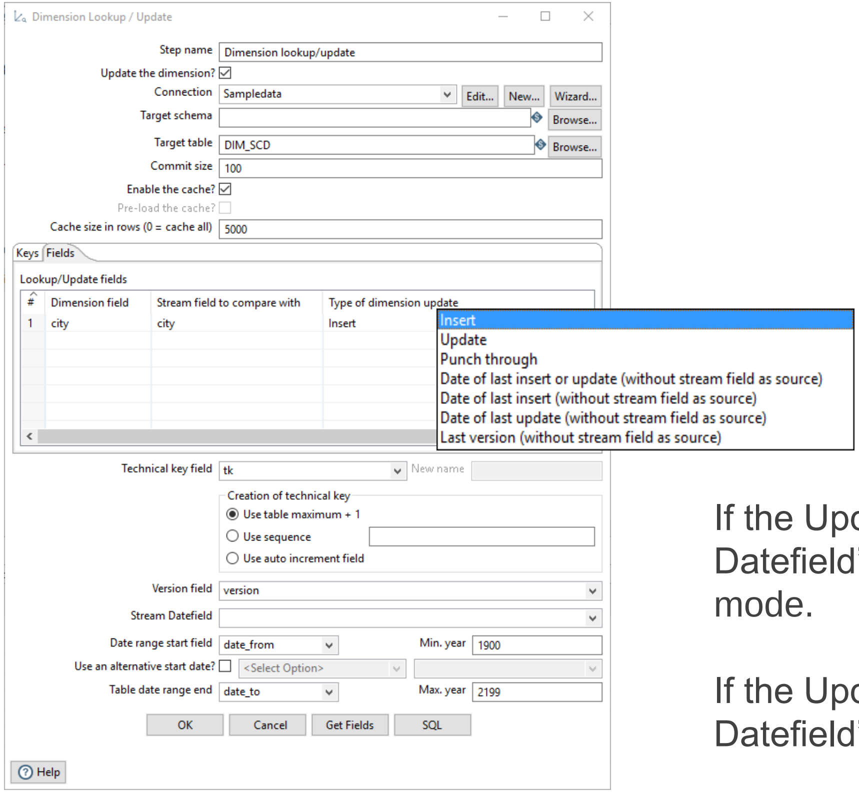
After the change:

Technical Key	Customer_ID	Customer_Name	Customer_Type	Start_Date	End_Date	Version
1	1	Cust_1	Corporate	22-07-2010	17-05-2012	1
2	1	Cust_1	Retail	17-05-2012	31-12-9999	2

Lab 6 – Slowly Changing Dimensions

Lab 6 - Dimension Lookup / Update Type 1

- Operates in 2 modes:



Dimension Lookup / Update

Step name	Dimension lookup/update
Update the dimension?	☑
Connection	Sampledata
Target schema	
Target table	DIM_SCD
Commit size	100
Enable the cache?	☑
Pre-load the cache?	□
Cache size in rows (0 = cache all)	5000

Keys | Fields

Lookup/Update fields

#	Dimension field	Stream field to compare with	Type of dimension update
1	city	city	Insert

Insert
Update
Punch through
Date of last insert or update (without stream field as source)
Date of last insert (without stream field as source)
Date of last update (without stream field as source)
Last version (without stream field as source)

Technical key field	tk
New name	

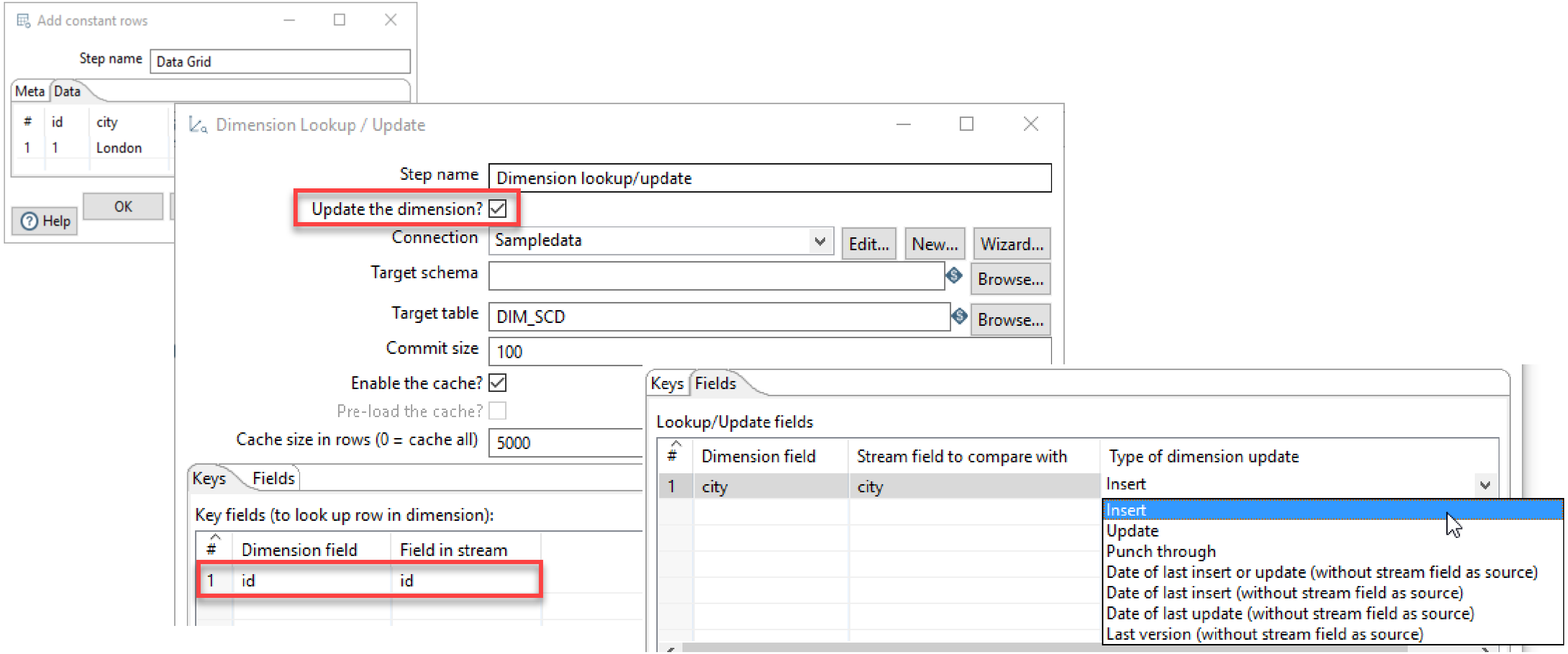
Creation of technical key
● Use table maximum + 1
○ Use sequence
○ Use auto increment field

Version field	version		
Stream Datefield			
Date range start field	date_from	Min. year	1900
Use an alternative start date?	□ <Select Option>		
Table date range end	date_to	Max. year	2199

If the Update option is selected, with no ‘Stream Datefield’, the step operates in Type I Update /Insert mode.

If the Update option is left unchecked, with no ‘Stream Datefield’, the step operates in Type 1 Lookup mode.

Lab 6 - Type 1 Insert / Update



Step name: Data Grid

#	id	city
1	1	London

Step name: Dimension lookup/update

Update the dimension? ☑

Connection: Sampledata

Target schema:

Target table: DIM_SCD

Commit size: 100

Enable the cache? ☑

Pre-load the cache? ☐

Cache size in rows (0 = cache all): 5000

Key fields (to look up row in dimension):

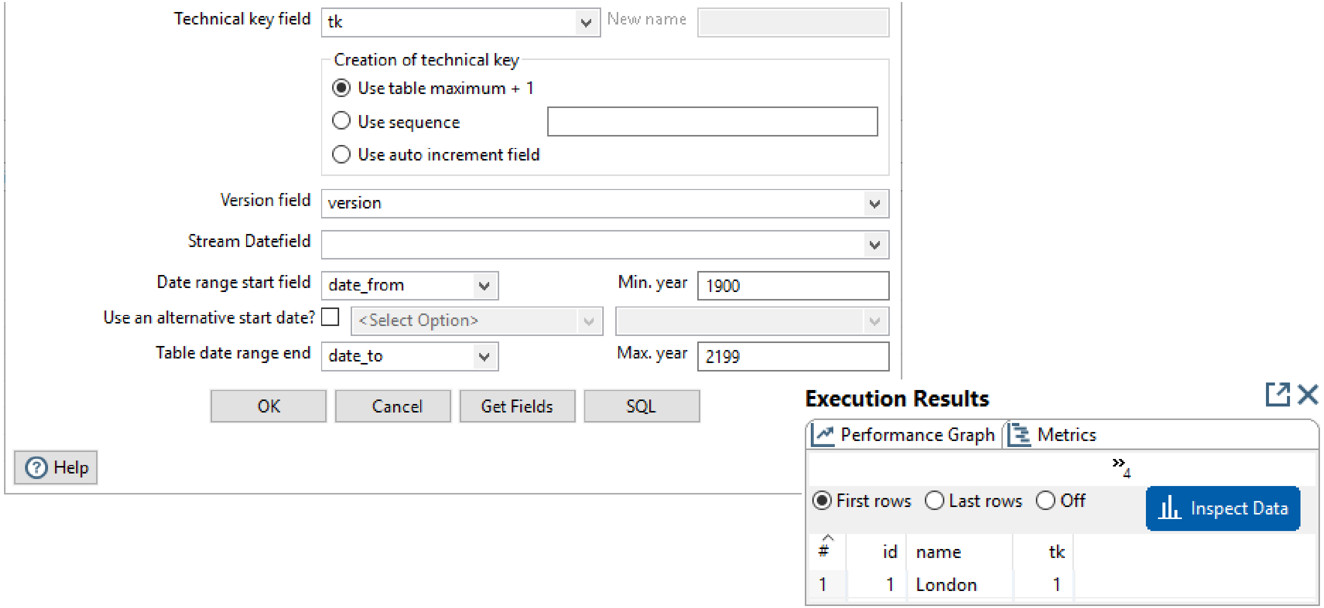
#	Dimension field	Field in stream
1	id	id

Lookup/Update fields

#	Dimension field	Stream field to compare with	Type of dimension update
1	city	city	Insert

Insert
Update
Punch through
Date of last insert or update (without stream field as source)
Date of last insert (without stream field as source)
Date of last update (without stream field as source)
Last version (without stream field as source)

Lab 6 - Type 1 Insert /Update



Technical key field: tk New name:

Creation of technical key
- ● Use table maximum + 1
- ○ Use sequence
- ○ Use auto increment field

Version field: version

Stream Datefield:

Date range start field: date_from Min. year: 1900

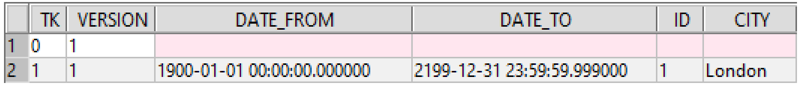
Use an alternative start date? □ <Select Option>

Table date range end: date_to Max. year: 2199

Execution Results

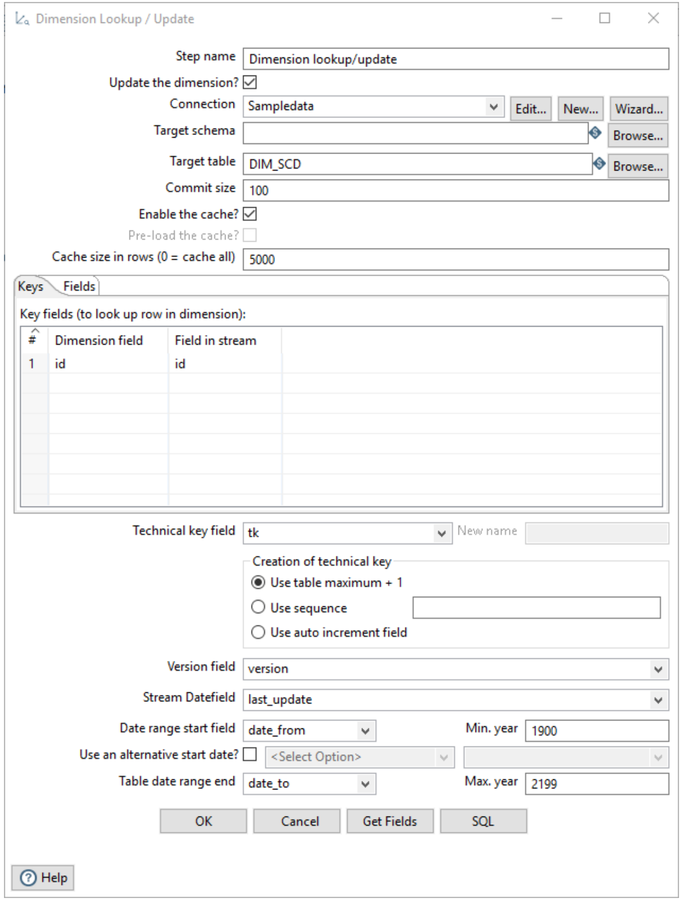
● First rows ○ Last rows ○ Off

#	id	name	tk
1	1	London	1



	TK	VERSION	DATE_FROM	DATE_TO	ID	CITY
1	0	1				
2	1	1	1900-01-01 00:00:00.000000	2199-12-31 23:59:59.999000	1	London

Lab 6 - Dimension Lookup / Update Type 2



Dimension Lookup / Update

Step name: Dimension lookup/update

Update the dimension? ☑

Connection: Sampledata

Target schema:

Target table: DIM_SCD

Commit size: 100

Enable the cache? ☑

Pre-load the cache? □

Cache size in rows (0 = cache all): 5000

Key fields (to look up row in dimension):

#	Dimension field	Field in stream
1	id	id

Technical key field: tk

New name:

Creation of technical key

● Use table maximum + 1

○ Use sequence

○ Use auto increment field

Version field: version

Stream Datefield: last_update

Date range start field: date_from

Min. year: 1900

Use an alternative start date? □ <Select Option>

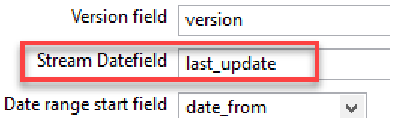
Table date range end: date_to

Max. year: 2199

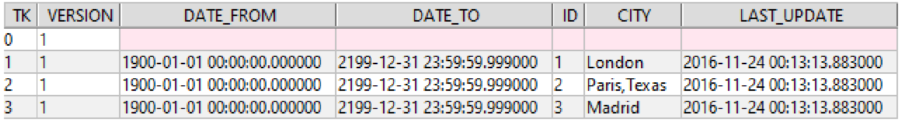
- If the Update option is selected with a Stream Datefield the step operates in Type 2 mode (Update /Insert)
- Historical record is preserved as updating the last_update, forces a new record to be inserted.

Page 180

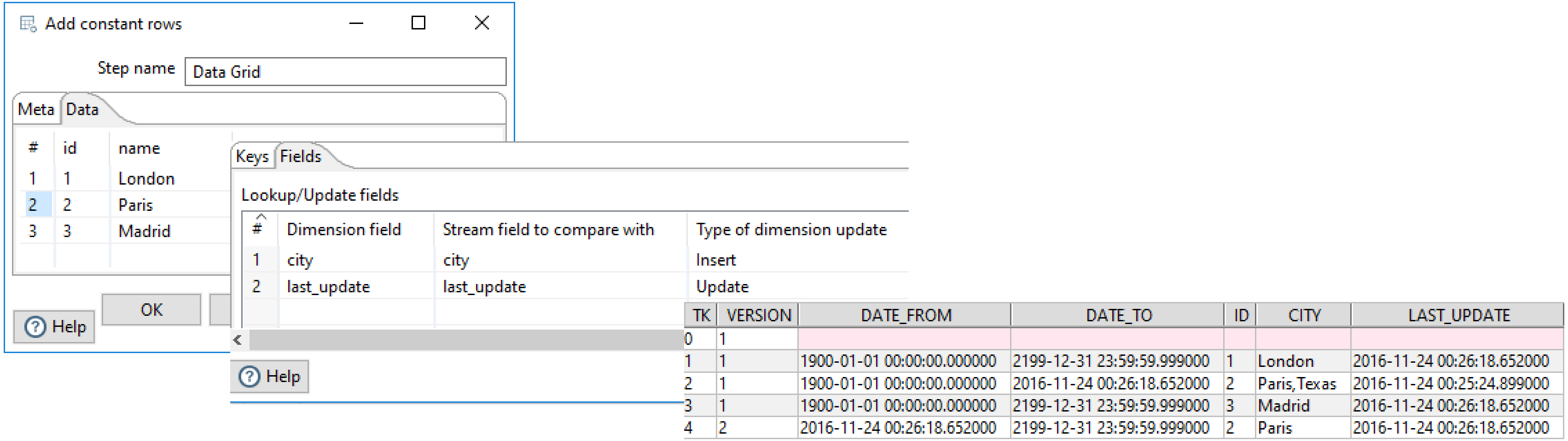
Dimension Insert / Update Type 2



Version field	version
Stream Datefield	last_update
Date range start field	date_from



TK	VERSION	DATE_FROM	DATE_TO	ID	CITY	LAST_UPDATE
0	1					
1	1	1900-01-01 00:00:00.000000	2199-12-31 23:59:59.999000	1	London	2016-11-24 00:13:13.883000
2	1	1900-01-01 00:00:00.000000	2199-12-31 23:59:59.999000	2	Paris,Texas	2016-11-24 00:13:13.883000
3	1	1900-01-01 00:00:00.000000	2199-12-31 23:59:59.999000	3	Madrid	2016-11-24 00:13:13.883000



Add constant rows

Step name: Data Grid

Meta | Data

#	id	name
1	1	London
2	2	Paris
3	3	Madrid

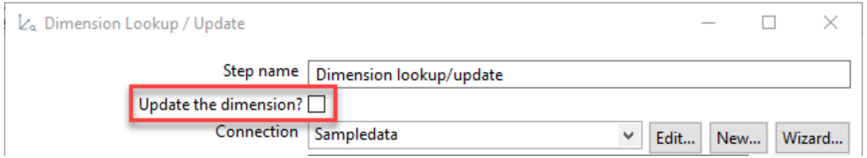
Keys | Fields

Lookup/Update fields

#	Dimension field	Stream field to compare with	Type of dimension update
1	city	city	Insert
2	last_update	last_update	Update

TK	VERSION	DATE_FROM	DATE_TO	ID	CITY	LAST_UPDATE
0	1					
1	1	1900-01-01 00:00:00.000000	2199-12-31 23:59:59.999000	1	London	2016-11-24 00:26:18.652000
2	1	1900-01-01 00:00:00.000000	2016-11-24 00:26:18.652000	2	Paris,Texas	2016-11-24 00:25:24.899000
3	1	1900-01-01 00:00:00.000000	2199-12-31 23:59:59.999000	3	Madrid	2016-11-24 00:26:18.652000
4	2	2016-11-24 00:26:18.652000	2199-12-31 23:59:59.999000	2	Paris	2016-11-24 00:26:18.652000

Lab 6: Type 1 Lookup

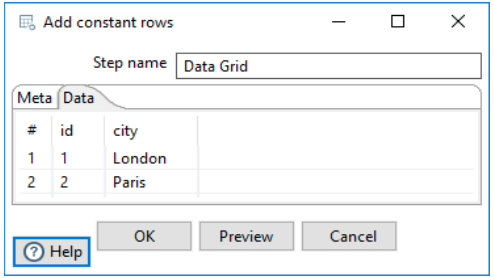


Dimension Lookup / Update

Step name: Dimension lookup/update

Update the dimension? □

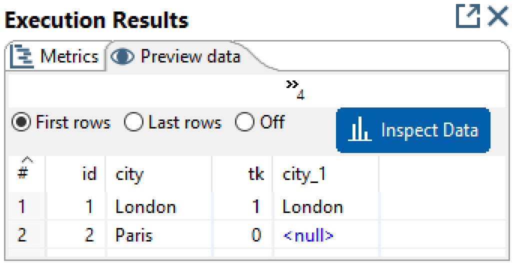
Connection: Sampledata



Add constant rows

Step name: Data Grid

#	id	city
1	1	London
2	2	Paris



Execution Results

● First rows ○ Last rows ○ Off

#	id	city	tk	city_1
1	1	London	1	London
2	2	Paris	0	<null>

- The record is not written to the table

Page 178

Module Recap

- In this module, you should have learned to:
  - Configure a Database Connection
  - Conduct Standard database operations

    - Create / Read

    - Update

    - Insert

    - Delete
  - Implement a Type II Slowly Changing Dimension

HITACHI
Inspire the Next

Thank You

HITACHI

Inspire the Next