Azure Data Factory V2 – Incremental loading with configuration stored in a table – Complete solution, step by step.

sql.pawlikowski.pro/2018/07/01/en-azure-data-factory-v2-incremental-loading-with-configuration-stored-in-a-table

Michał Pawlikowski 1 July 2018

This post explains things that are difficult to find even in English. That's why I will break my rule and will not write it in my native language! Po wersję polską zapraszam do google translate :>

Introduction

Loading data using Azure Data Factory v2 is really simple. Just drop Copy activity to your pipeline, choose a source and sink table, configure some properties and that's it – done with just a few clicks!

But what if you have dozens or hundreds of tables to copy? Are you gonna do it for every object?

Fortunately, you do not have to do this! All you need is dynamic parameters and a few simple tricks

Also, this will give you the option of creating incremental feeds, so that – at next run – it will transfer only newly added data.

Mappings

Before we start diving into details, let's demystify some basic ADFv2 mapping principles.

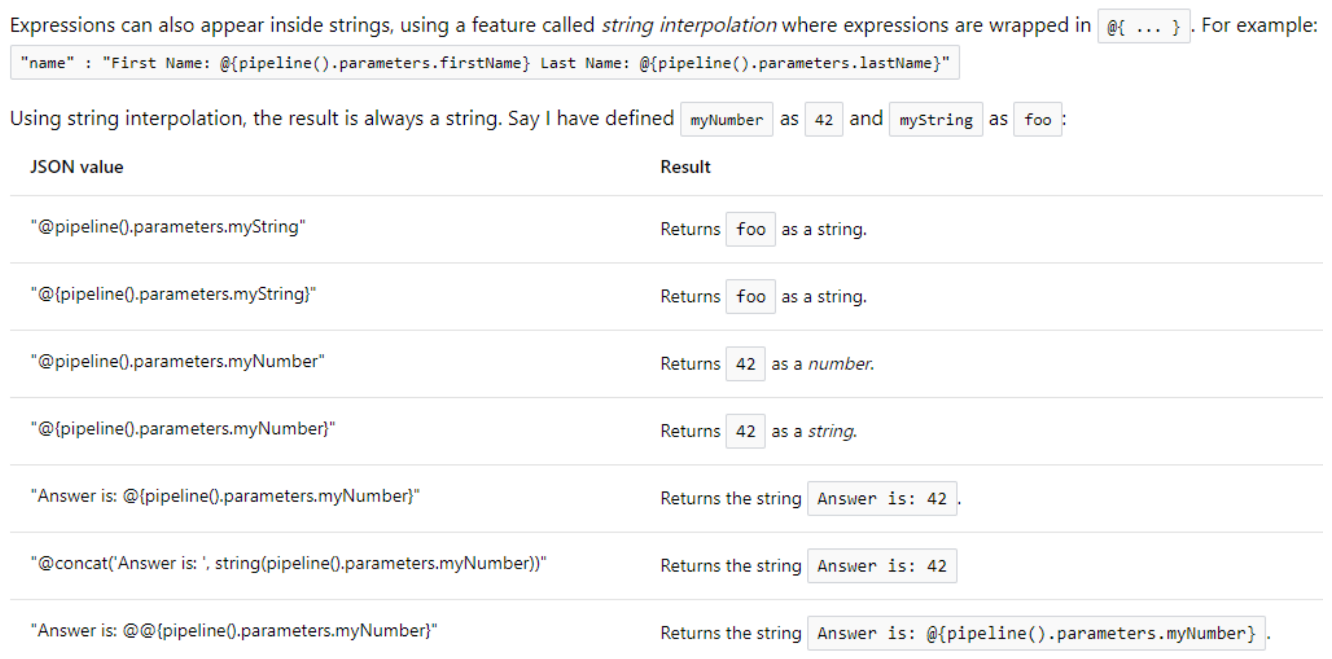
- **Copy activity doesn't need to have defined column mappings at all**,
- **it can dynamically map them** using its own mechanism which retrieves source and destination (sink) metadata,
- if you **use polybase**, it will do it using **column order** (1st column from source to 1st column at destination etc.),
- if you **do not use polybase**, it will map them **using their names but watch out – it's case sensitive matching!**
- So all you have to do is to just **keep the same structure and data types on the destination tables (sink)**, as they are in a source database.

Bear in mind, that if your columns are different between source and destination, you will have to provide custom mappings. This tutorial doesn't show how to do it, but it is possible to pass them using "Get metadata" activity to retrieve column specification from the source, then you have to parse it and pass as JSON structure into the mapping dynamic input. you can read about mappings in official documentation: https://docs.microsoft.com/en-us/azure/data-factory/copy-activity-schema-and-type-mapping

String interpolation – the key to success

My entire solution is based on one cool feature, that is called **string interpolation**. It is a part of built-in expression engine, that simply allows you to just inject any value from JSON object or an expression directly into string input, without any concatenate functions or operators. It's fast and easy. Just wrap your expression between `@{ ... }`. It will always return it as a string.

Below is a screen from official documentation, that clarifies how this feature works:


Expressions can also appear inside strings, using a feature called *string interpolation* where expressions are wrapped in `@{ ... }`. For example: `"name" : "First Name: @{pipeline().parameters.firstName} Last Name: @{pipeline().parameters.lastName}"`

Using string interpolation, the result is always a string. Say I have defined `myNumber` as `42` and `myString` as `foo`:

JSON value	Result
"@pipeline().parameters.myString"	Returns `foo` as a string.
"@{pipeline().parameters.myString}"	Returns `foo` as a string.
"@pipeline().parameters.myNumber"	Returns `42` as a *number*.
"@{pipeline().parameters.myNumber}"	Returns `42` as a *string*.
"Answer is: @{pipeline().parameters.myNumber}"	Returns the string `Answer is: 42`.
"@concat('Answer is: ', string(pipeline().parameters.myNumber))"	Returns the string `Answer is: 42`
"Answer is: @@{pipeline().parameters.myNumber}"	Returns the string `Answer is: @{pipeline().parameters.myNumber}`.

Read more about JSON expressions at https://docs.microsoft.com/en-us/azure/data-factory/control-flow-expression-language-functions#expressions

So what we are going to do? :>

Good question

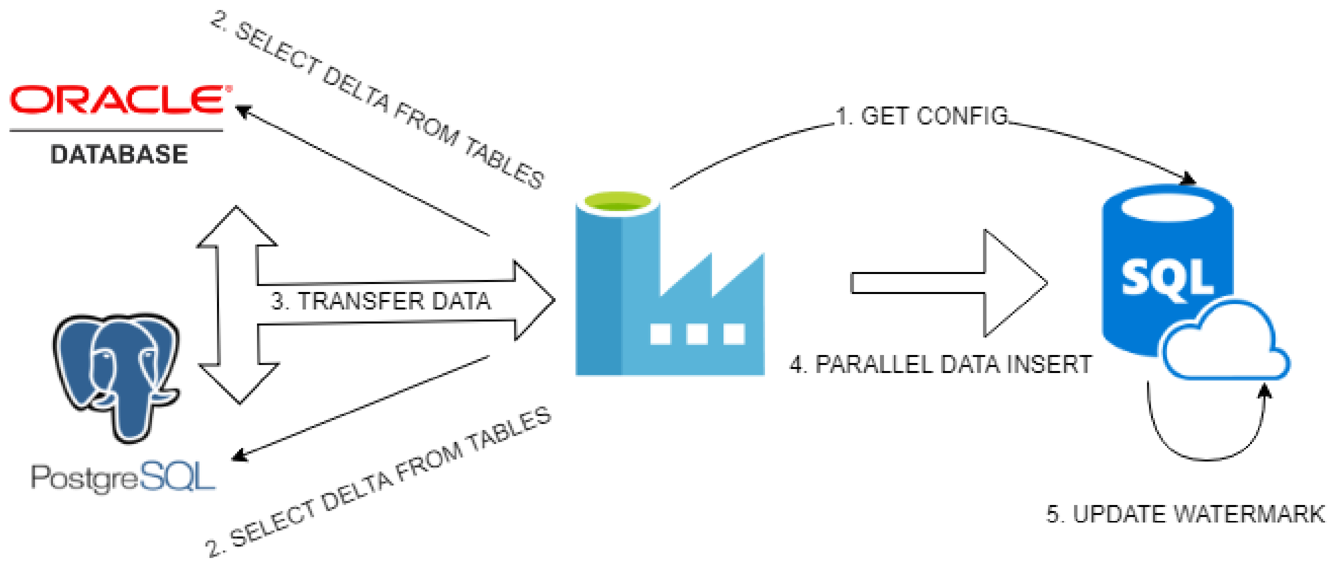
In my example, **I will show you how to transfer data incrementally from Oracle and PostgreSQL tables into Azure SQL Database**.

All of this **using configuration stored in a table**, which in short, keeps information about Copy activity settings needed to achieve our goal

Adding new definitions into config will also automatically enable transfer for them, without any need to modify Azure Data Factory pipelines.

So you can transfer as many tables as you want, in one pipeline, at once. Triggering with one click

Every process needs diagram :>



Basically, we will do:

1. **Get configuration** from our config table inside Azure SQL Database using **Lookup activity**, then pass it to **Filter activity** to split configs for Oracle and PostgreSQL.
2. In **Foreach activity** created for every type of database, we will create simple logic that **retrieves maximum update date** from every table.
3. Then we will prepare dynamically expressions for **SOURCE** and **SINK** properties in **Copy activity**. MAX UPDATEDATE, retrieved above, and previous WATERMARK DATE, retrieved from config, will set our boundaries in WHERE clause. Every detail like **table name** or **table columns** we will pass as a query using string interpolation, directly from JSON expression. Sink destination will be also parametrized.
4. Now Azure Data Factory can **execute queries evaluated dynamically** from JSON expressions, it will **run them in parallel** just to speed up data transfer.
5. Every successfully transferred portion of incremental data for a given table **has to be marked as done**. We can do this saving MAX UPDATEDATE in configuration, so that next incremental load will know what to take and what to skip. We will use here: **Stored procedure activity.**

This example simplifies the process as much as it is possible. **Remember, in your solution you have to implement logic for every unsuccessful operation.** You can achieve that using On Failure control flow with some activities (chosen depending on your needs) and timeout/retry options set individually for every activity in your pipeline.

About sources

I will use PostgreSQL 10 and Oracle 11 XE installed on my Ubuntu 18.04 inside VirtualBox machine.

In Oracle, tables and data were generated from EXMP/DEPT samples delivered with XE version.

In PostgreSQL – from dvd rental sample database: http://www.postgresqltutorial.com/postgresql-sample-database/

I simply chose three largest tables from each database. You can find them in a configuration shown below this section.

Every database is accessible from my Self-hosted Integration Runtime. I will show an example how to add the server to Linked Services, but skip configuring Integration Runtime. You can read about creating self-hosted IR here: https://docs.microsoft.com/en-us/azure/data-factory/create-self-hosted-integration-runtime.

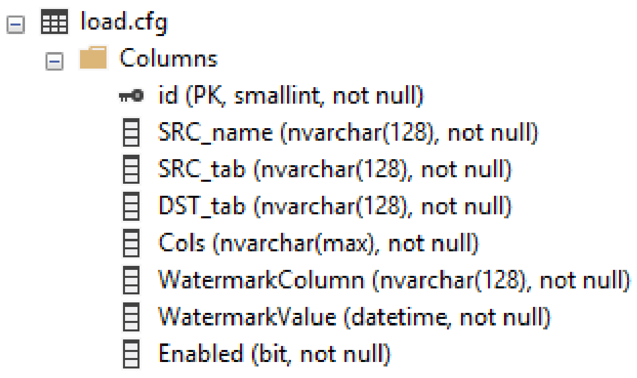
About configuration

In my Azure SQL Database I have created a simple configuration table:

Id is just an identity value, SRC_name is a type of source server (ORA or PG).

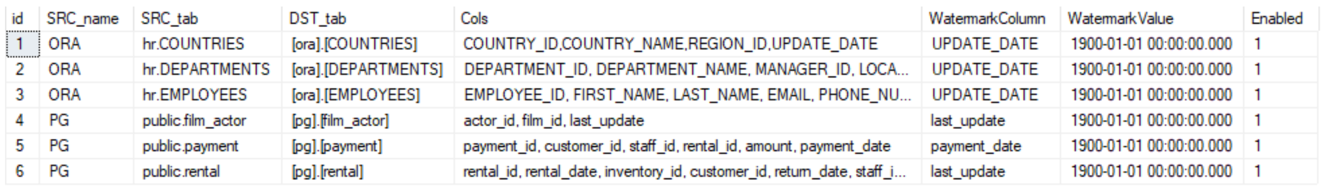
SRC and DST tab columns maps source and destination objects. Cols defines selected columns, Watermark Column and Value stores incremental metadata.

And finally Enabled just enables particular configuration (table data import).



```
load.cfg
  Columns
    id (PK, smallint, not null)
    SRC_name (nvarchar(128), not null)
    SRC_tab (nvarchar(128), not null)
    DST_tab (nvarchar(128), not null)
    Cols (nvarchar(max), not null)
    WatermarkColumn (nvarchar(128), not null)
    WatermarkValue (datetime, not null)
    Enabled (bit, not null)
```

As Andy rightly noted in the comment below this post, it is possible to use "Cols" also to implement SQL logic, like functions, aliases etc. The value from this column is rewritten directly to the query (more precisely – concatenated between SELECT and FROM clause). So you can use it according to your needs.
This is how it looks with initial configuration:



id	SRC_name	SRC_tab	DST_tab	Cols	WatermarkColumn	WatermarkValue	Enabled
1	ORA	hr.COUNTRIES	[ora].[COUNTRIES]	COUNTRY_ID,COUNTRY_NAME,REGION_ID,UPDATE_DATE	UPDATE_DATE	1900-01-01 00:00:00.000	1
2	ORA	hr.DEPARTMENTS	[ora].[DEPARTMENTS]	DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID, LOCA...	UPDATE_DATE	1900-01-01 00:00:00.000	1
3	ORA	hr.EMPLOYEES	[ora].[EMPLOYEES]	EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, PHONE_NU...	UPDATE_DATE	1900-01-01 00:00:00.000	1
4	PG	public.film_actor	[pg].[film_actor]	actor_id, film_id, last_update	last_update	1900-01-01 00:00:00.000	1
5	PG	public.payment	[pg].[payment]	payment_id, customer_id, staff_id, rental_id, amount, payment_date	payment_date	1900-01-01 00:00:00.000	1
6	PG	public.rental	[pg].[rental]	rental_id, rental_date, inventory_id, customer_id, return_date, staff_i...	last_update	1900-01-01 00:00:00.000	1

Create script:

Transact-SQL

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [load].[cfg](
[id] [SMALLINT] IDENTITY(1,1) NOT NULL,
[SRC_name] [NVARCHAR](128) NOT NULL,
[SRC_tab] [NVARCHAR](128) NOT NULL,
[DST_tab] [NVARCHAR](128) NOT NULL,
[Cols] [NVARCHAR](MAX) NOT NULL,
[WatermarkColumn] [NVARCHAR](128) NOT NULL,
[WatermarkValue] [DATETIME] NOT NULL,
[Enabled] [BIT] NOT NULL,
CONSTRAINT [PK_load] PRIMARY KEY CLUSTERED
(
[id] ASC
)WITH (STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
ALTER TABLE [load].[cfg] ADD  CONSTRAINT
[DF__cfg__WatermarkVa__4F7CD00D]  DEFAULT ('1900-01-01') FOR
[WatermarkValue]
GO
```

EDIT 19.10.2018

Microsoft announced, that now you can parametrize also linked connections!

https://azure.microsoft.com/en-us/blog/parameterize-connections-to-your-data-stores-in-azure-data-factory/

Let’s get started (finally :P)

Preparations!

Go to your Azure Data Factory portal @ https://adf.azure.com/

Select **Author** button with pencil icon:

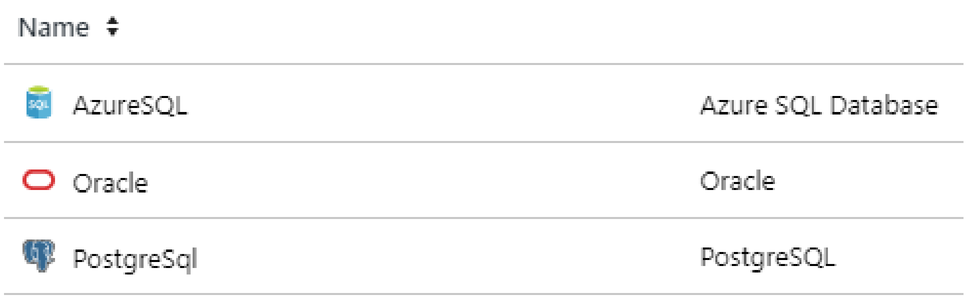
Author

Creating server connections (Linked Services)

We can't do anything without defining **Linked Services**, which are just connections to your servers (on-prem and cloud).

1. Go to Connections and click + New
2. Find your database type, select and click Continue
3. Give all needed data, like server ip/host, port, SID (Oracle need this), login and password.
4. You can Test connection if everything is ok. Click Finish to save your connection definition.

Some types of servers, such as PostgreSQL or MySQL, require separate .NET drivers. Check your server type here in Microsoft Docs and search for Prerequisites to match your scenario.
I have created three connections. Here are their names and server types:



Name	
AzureSQL	Azure SQL Database
Oracle	Oracle
PostgreSql	PostgreSQL

Creating datasets

Creating linked services is just telling ADF what are connection settings (like connection strings).

Datasets, on the other hand, points directly to database objects.

BUT they can be parametrized, so you can just create ONE dataset and use it passing different parameters to get data from multiple tables within same source database

Source datasets

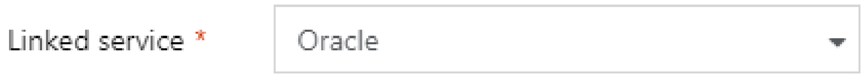
Source datasets don't need any parameters. We will later use built-in query parametrization to pass object names.

1. Go to and click + and choose
2. Choose your datataset type, for example
3. Rename it just as you like. We will use name: "ORA"
4. Set proper Linked service option, just like this for oracle database:

Factory Resources

Dataset

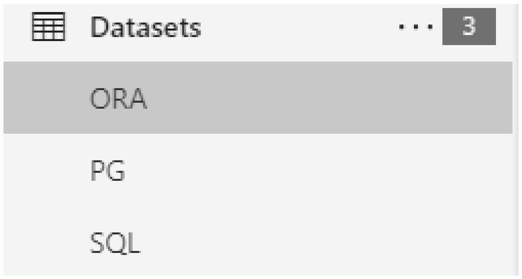
ORACLE
DATABASE


Linked service * Oracle

5. And that's it! No need to set anything else. Just repeat these steps for every source database, that you have.

In my example, I've created two source datasets, ORA and PG

As you can see, we need to create also the third dataset. It will work as a source too, BUT also as a parametrizable sink (destination). So creating it is little different than others.

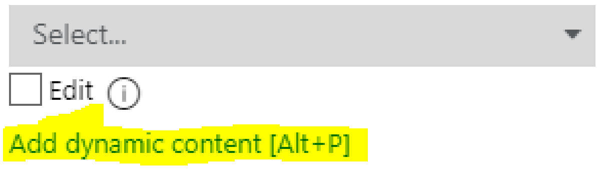


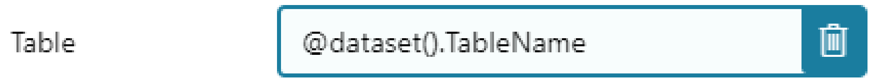
Sink dataset

Sinking data needs one more extra parameter, which will store destination table name.

1. Create dataset just like in the previous example, choose your destination type. In my case, it will be Azure SQL Database.
2. Go to Parameters, declare one String parameter called "TableName". Set the value to anything you like. It's just dummy value, ADF just doesn't like empty parameters, so we have to set a default value.
3. Now, go to Connection, set Table as dynamic content. This will be tricky :). Just click "Select...", don't choose any value, just click somewhere in empty space. The magic option "Add dynamic content" now appears! You have to click it or hit alt+p.
4. "Add Dynamic Content" windows is now visible. Type: "@dataset().TableName" or just click "TableName" in "Parameters" section below "Functions".

5. The table name is now parameterized. And looks like this:





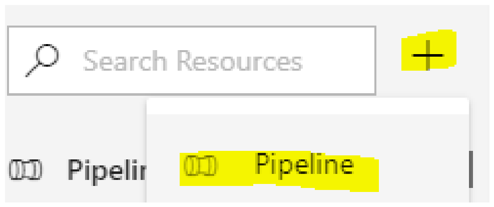
```
@dataset().TableName
```

Parametrizable PIPELINE with dynamic data loading.

Ok, our connections are defined. Now it's time to copy data :>

Creating pipeline

1. Go to you ADF and click **PLUS** symbol near search box on the left and choose "**Pipeline**":
2. Reanme it. I will use "**LOAD DELTA**".
3. Go to **Parameters**, create new String parameter called **ConfigTable**. Set value to our configuration table name: **load.cfg** . This will simply parametrize you configuration source. So that in the future it would be possible to load a completely different set of sources by changing only one parameter :>
4. In case you missed it, **SAVE** your work by clicking "**Save All**" if you're using GIT or "**Publish All**" if not ;]



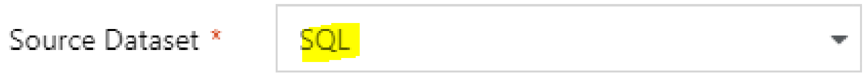
Creating Lookup – GET CFG

First, we have to get configuration. We will use **Lookup activity** to retrieve it from the database.

Bear in mind, that **lookup activity has some limits**. Currently, the maximum number of rows, that can be returned by Lookup activity is **5000**, and up to **2MB** in size. Also max duration for Lookup activity before timeout is one hour. Go to documentation for latest info and updates.

1.
    1. Drag and drop into your pipline
    2. **Rename it**. This is important, we will use this name later in our solution. I will use value “**GET CFG**“.

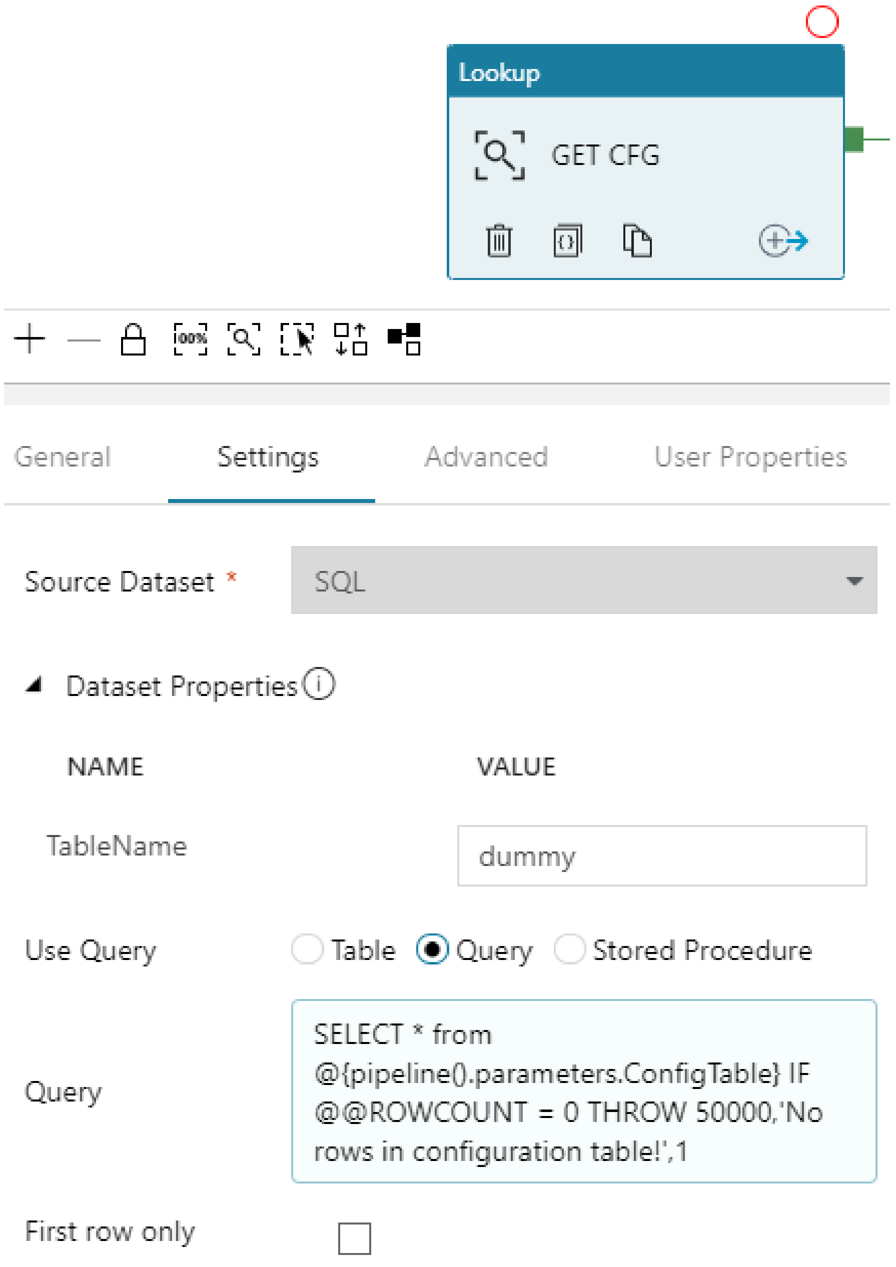
    3. In “**Settings**” choose


Source Dataset * SQL

    4. Now, don’t bother TableName set to dummy :> Just in “**Use Query**” set to “**Query**“, click “**Add dynamic content**” and type:
       JavaScript

```
SELECT * from @{pipeline().parameters.ConfigTable}
IF @@ROWCOUNT = 0 THROW 50000,'No rows in configuration table!',1
```

    5. Unmark “**First row only**“, we need all rows, not just first. All should look like this:



Source Dataset * SQL

Dataset Properties

NAME	VALUE
TableName	dummy

Use Query: Table [✓ Query] Stored Procedure

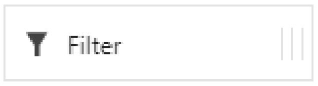
Query:

```
SELECT * from
@{pipeline().parameters.ConfigTable} IF
@@ROWCOUNT = 0 THROW 50000,'No
rows in configuration table!',1
```

First row only ☐

Creating Filters – ORA CFG & PG CFG

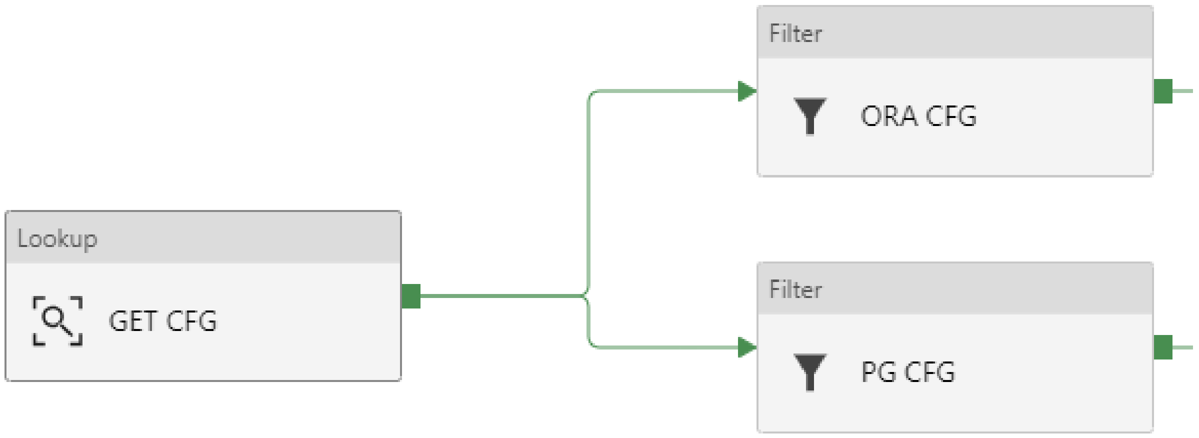
Now we have to split configs for oracle and PostgreSQL. We will use **Filter activity** on rows retrieved in "**GET CFG**" lookup.

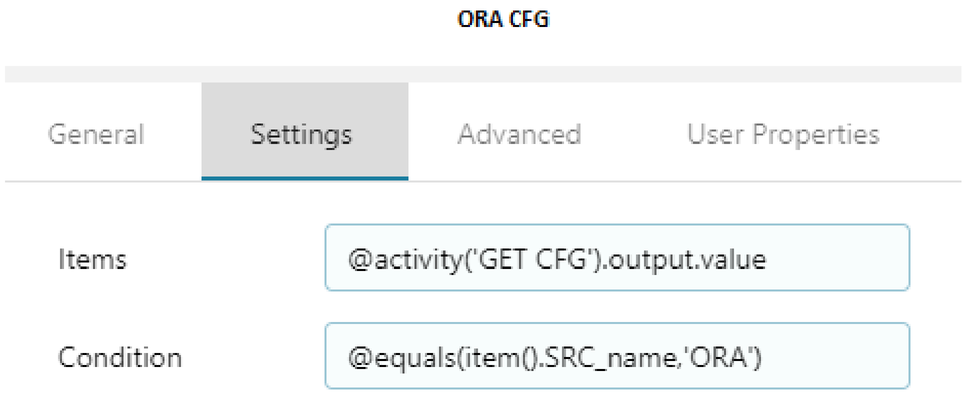


1. Drag and drop twice.
2. Rename the first block to "**ORA CFG**", second to "**PG CFG**".
3. Now go to "**ORA CFG**", then "**Settings**".
4. In **Items**, click **Add dynamic content** and type: @activity('GET CFG').output.value . As you probably guess, this will point directly to GET CFG output rows

5. In **Condition**, click **Add dynamic content** and type: @equals(item().SRC_name,'ORA') . We have to match rows for oracle settings. So we know, that there is a column in config table called “**SRC_name**“. We can use it to filter out all rows, except that with value ‘ORA’ .
6. Do the same with lookup activity “ **PG CFG**“. Of course, change the value for a condition.

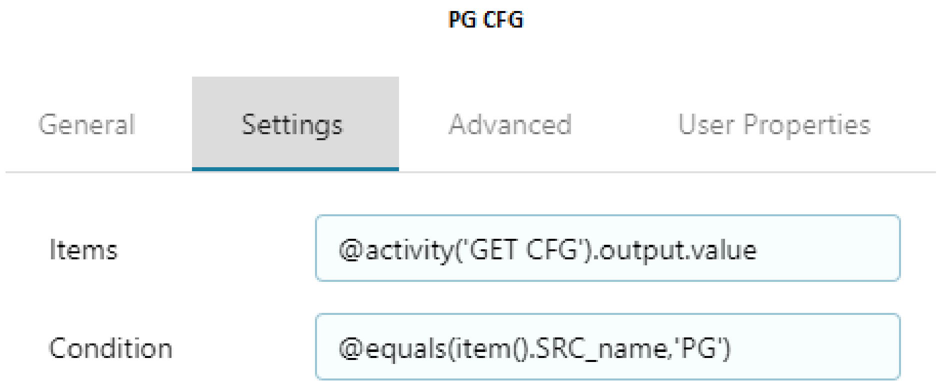
It should look like this:





Items @activity('GET CFG').output.value

Condition @equals(item().SRC_name,'ORA')



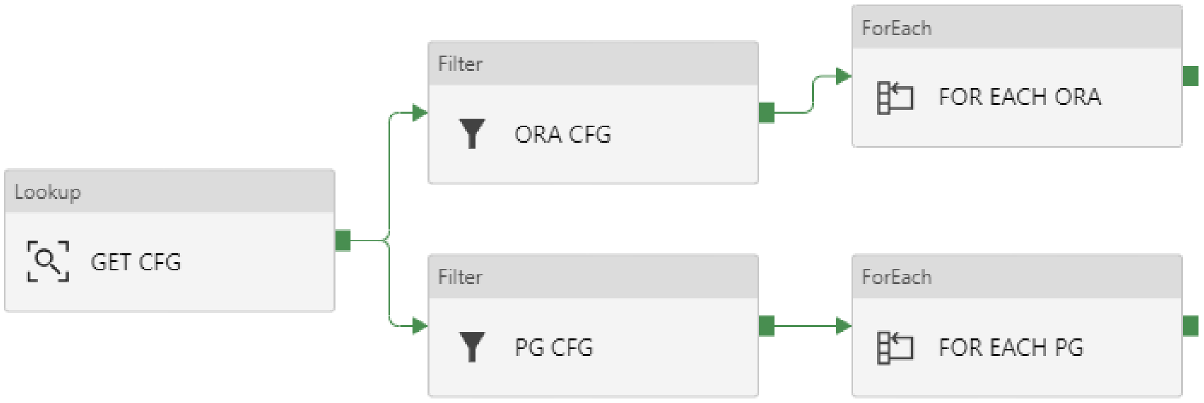
Items @activity('GET CFG').output.value

Condition @equals(item().SRC_name,'PG')

Creating ForEach – FOR EACH ORA & FOR EACH PG

Now it's time to iterate over each row filtered in separate containers (ORA CFG and PG CFG).

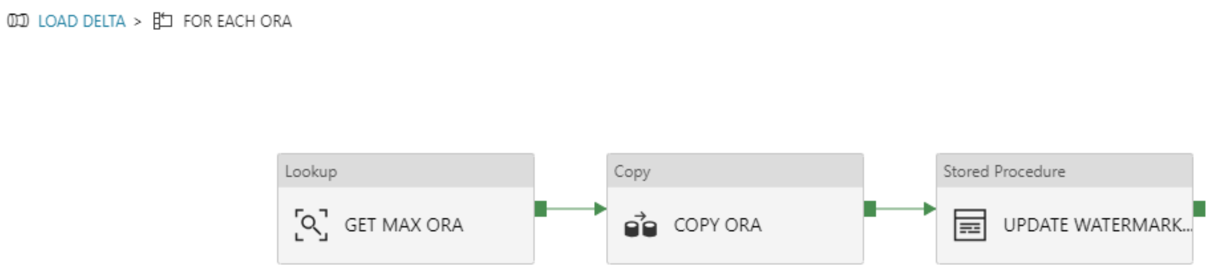
1. Drag and drop two blocks, rename them as "**FOR EACH ORA**" and "**FOR EACH PG**". Connect each to proper filter acitivity. Just like in this example:



2. Click "**FOR EACH ORA**", go to "**Settings**", in **Items** clik **Add dynamic content** and type: @activity('ORA CFG').output.value . We are telling ForEach, that it has to iterate over results returned in "ORA CFG". They are stored in JSON array.
3. Do this also in **FOR EACH PG**. Type: @activity('PG CFG').output.value
4. Now, you can edit **Activities** and add only "**WAIT**" activity to debug your pipeline. I will skip this part. Just remember to delete **WAIT** block at the end of your tests.

Inside ForEach – GET MAX ORA -> COPY ORA -> UPDATE WATERMARK ORA

Place these blocks into **FOR EACH ORA**. Justo go there, click "**Activities**" and then



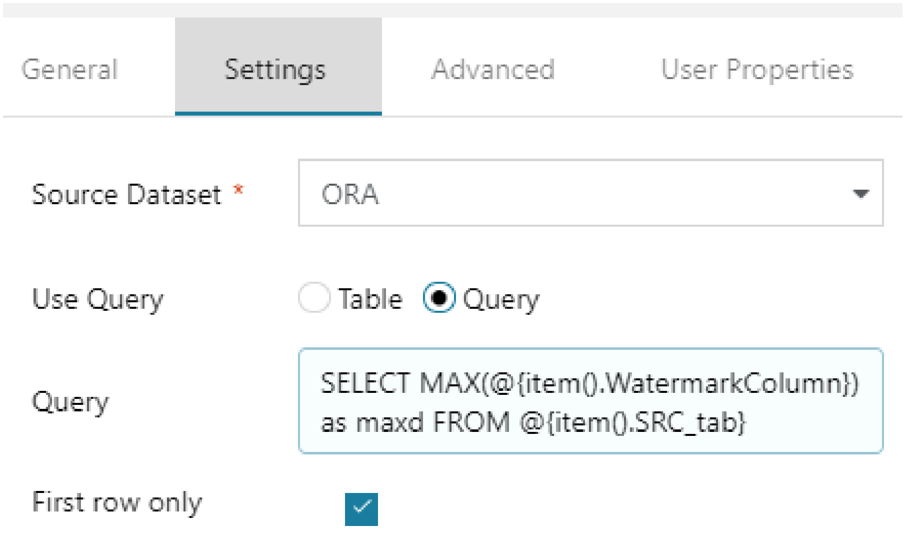
Every row, that ForEach activity is iterating over, is accessible using @item() .
And every column in that row, can be reached just by using @item().ColumnName .

Remember, that you can surround every expression in brackets @{ } to use it as a string interpolation. Then you can concatenate it with other strings and expressions just like that: Value of the parameter WatermarkColumn is: @{item().WatermarkColumn}

GET MAX ORA

1. Go to “**GET MAX ORA**“, then **Settings**
2. Choose your **source dataset** “**ORA**“, **Use Query**: “**Query**” and click **Add dynamic content**
3. Type SELECT MAX(@{item().WatermarkColumn}) as maxd FROM @{item().SRC_tab} . This will get a **maximum date** in your watermark column. We will use it as **RIGHT BOUNDRY** for delta slice.
4. Check if **First row only** is turned **on.**

It should look like this:



Source Dataset *	ORA
Use Query	Table [✓ Query]
Query	SELECT MAX(@{item().WatermarkColumn}) as maxd FROM @{item().SRC_tab}
First row only	✓

COPY ORA

Now the most important part :> Copy activity with a lot of parametrized things… So pay attention, it’s not so hard to understand but every detail matters.

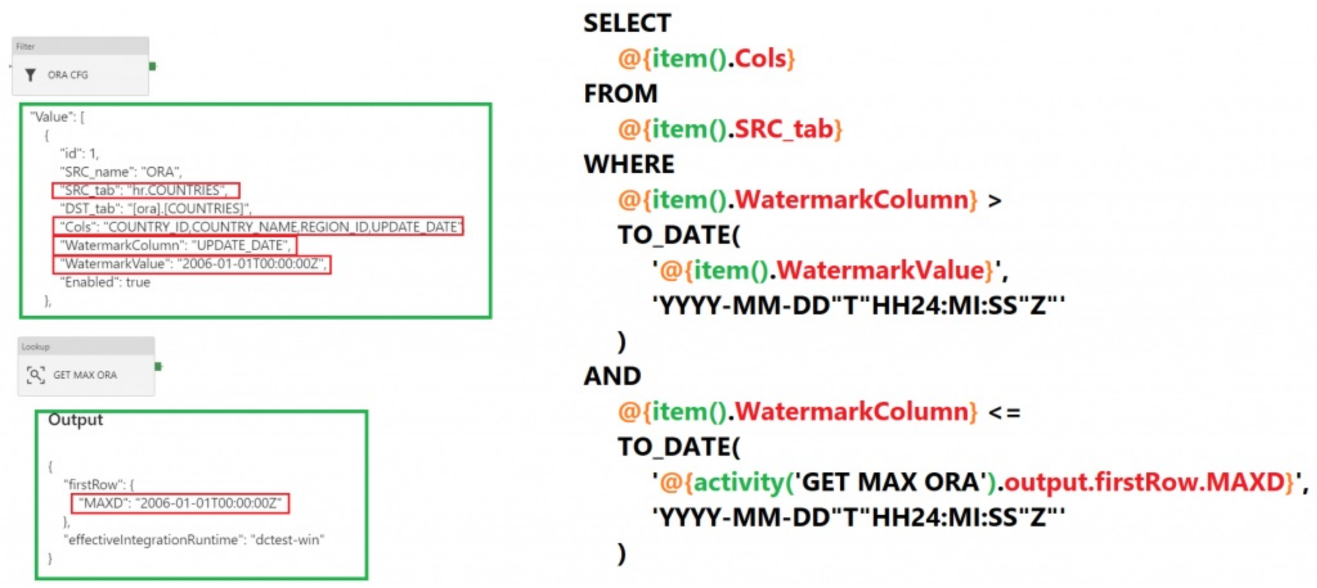
Source

1. In source settings, choose **Source Dataset** to **ORA**, in **Use query** select **Query**.

2. Below **Query** input, click **Add dynamic content** and paste this:
   JavaScript

```
SELECT
@{item().Cols} FROM @{item().SRC_tab}
WHERE
@{item().WatermarkColumn} >
TO_DATE('@{item().WatermarkValue}', 'YYYY-MM-DD"T"HH24:MI:SS"Z"')
AND
@{item().WatermarkColumn} <=
TO_DATE('@{activity('GET MAX ORA').output.firstRow.MAXD}', 'YYYY-
MM-DD"T"HH24:MI:SS"Z"')

```

Now, this needs some explanation



```
"Value": [
  {
    "id": 1,
    "SRC_name": "ORA",
    "SRC_tab": "hr.COUNTRIES",
    "DST_tab": "[ora].[COUNTRIES]",
    "Cols": "COUNTRY_ID,COUNTRY_NAME,REGION_ID,UPDATE_DATE",
    "WatermarkColumn": "UPDATE_DATE",
    "WatermarkValue": "2006-01-01T00:00:00Z",
    "Enabled": true
  },
```

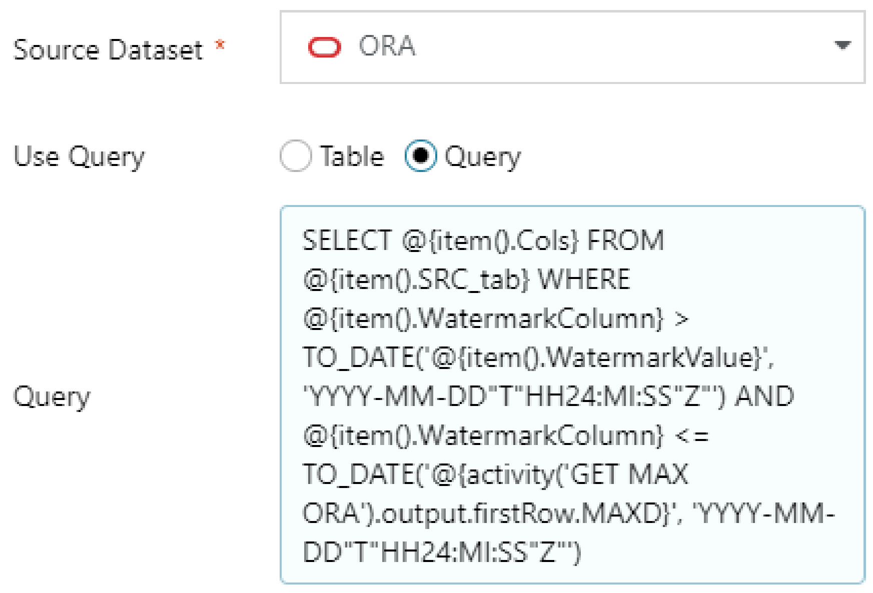
Output

```
{
  "firstRow": {
    "MAXD": "2006-01-01T00:00:00Z"
  },
  "effectiveIntegrationRuntime": "dctest-win"
}
```

```
SELECT
  @{item().Cols}
FROM
  @{item().SRC_tab}
WHERE
  @{item().WatermarkColumn} >
  TO_DATE(
    '@{item().WatermarkValue}',
    'YYYY-MM-DD"T"HH24:MI:SS"Z"'
  )
AND
  @{item().WatermarkColumn} <=
  TO_DATE(
    '@{activity('GET MAX ORA').output.firstRow.MAXD}',
    'YYYY-MM-DD"T"HH24:MI:SS"Z"'
  )
```

- **ORA CFG** output has all columns and their values from our config.
- We will use **SRC_tab** as table name, **Cols** as columns for SELECT query, **WatermatkColumn** as LastChange DateTime column name and **WatermarkValue** for **LEFT BOUNDRY (greater than, >)**.
- **GET MAX ORA** output stores date of a last updated row in the source table. So this is why we are using it as a **RIGHT BOUNDRY (less than or equal, <=)**
- And the tricky thing, ORACLE doesn't support implicit conversion from the string with ISO 8601 date. So we need to extract it properly with **TO_DATE** function.

So the source is a query from ORA dataset:


Source Dataset * ORA

Use Query Table [✓ Query]

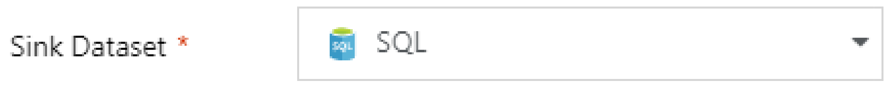
Query

```
SELECT @{item().Cols} FROM
@{item().SRC_tab} WHERE
@{item().WatermarkColumn} >
TO_DATE('@{item().WatermarkValue}',
'YYYY-MM-DD"T"HH24:MI:SS"Z"') AND
@{item().WatermarkColumn} <=
TO_DATE('@{activity('GET MAX
ORA').output.firstRow.MAXD}', 'YYYY-MM-
DD"T"HH24:MI:SS"Z"')
```

Sink

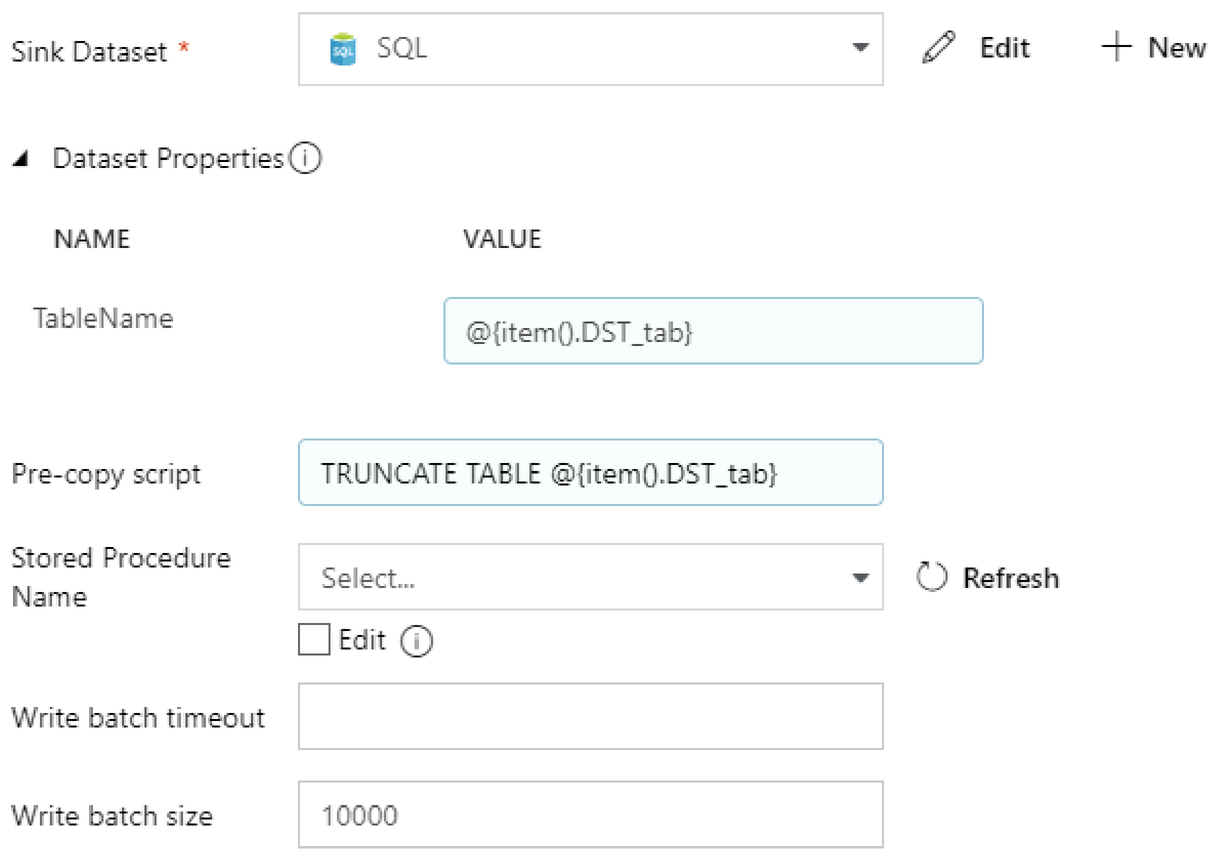
Sink is our destination. Here we will set parametrized table name and truncate query.

1. Select


Sink Dataset * SQL

2. Parametrize TableName as dynamic content with value: @{item().DST_tab}
3. Also, do the same with Pre-copy script and put there: TRUNCATE TABLE @{item().DST_tab}

As **De jan** properly noticed in comments, you are not obligated to use any Pre-copy script here. You can leave this box empty or run other commands like partitions switching.
It should look like this:



Sink Dataset *	SQL

Dataset Properties

NAME	VALUE
TableName	@{item().DST_tab}

Pre-copy script	TRUNCATE TABLE @{item().DST_tab}
Stored Procedure Name	Select...
Write batch timeout	
Write batch size	10000

Mappings and Settings

All other things should just be set to defaults. You don't have to parametrize mappings if you just copy data from and to tables that have the same structure.

Of course, you can dynamically create them if you want, but it is a good practice to transfer data 1:1 – both structure and values from source to staging.

UPDATE WATERMARK ORA

Now we have to confirm, that load has finished and then update previous watermark value with the new one.

We will use a stored procedure. The code is simple:

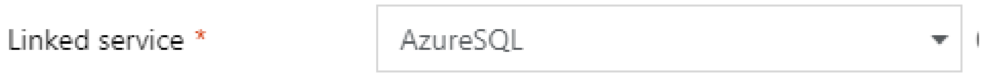
Transact-SQL

```
CREATE PROC [load].[usp_UpdateWatermark]
   @id SMALLINT,
   @NewWatermark DATETIME
AS
SET NOCOUNT ON;
UPDATE load.cfg
SET WatermarkValue = @NewWatermark
WHERE id = @id;
GO



```

Create it on your Azure SQL database. Then use it in ADF:

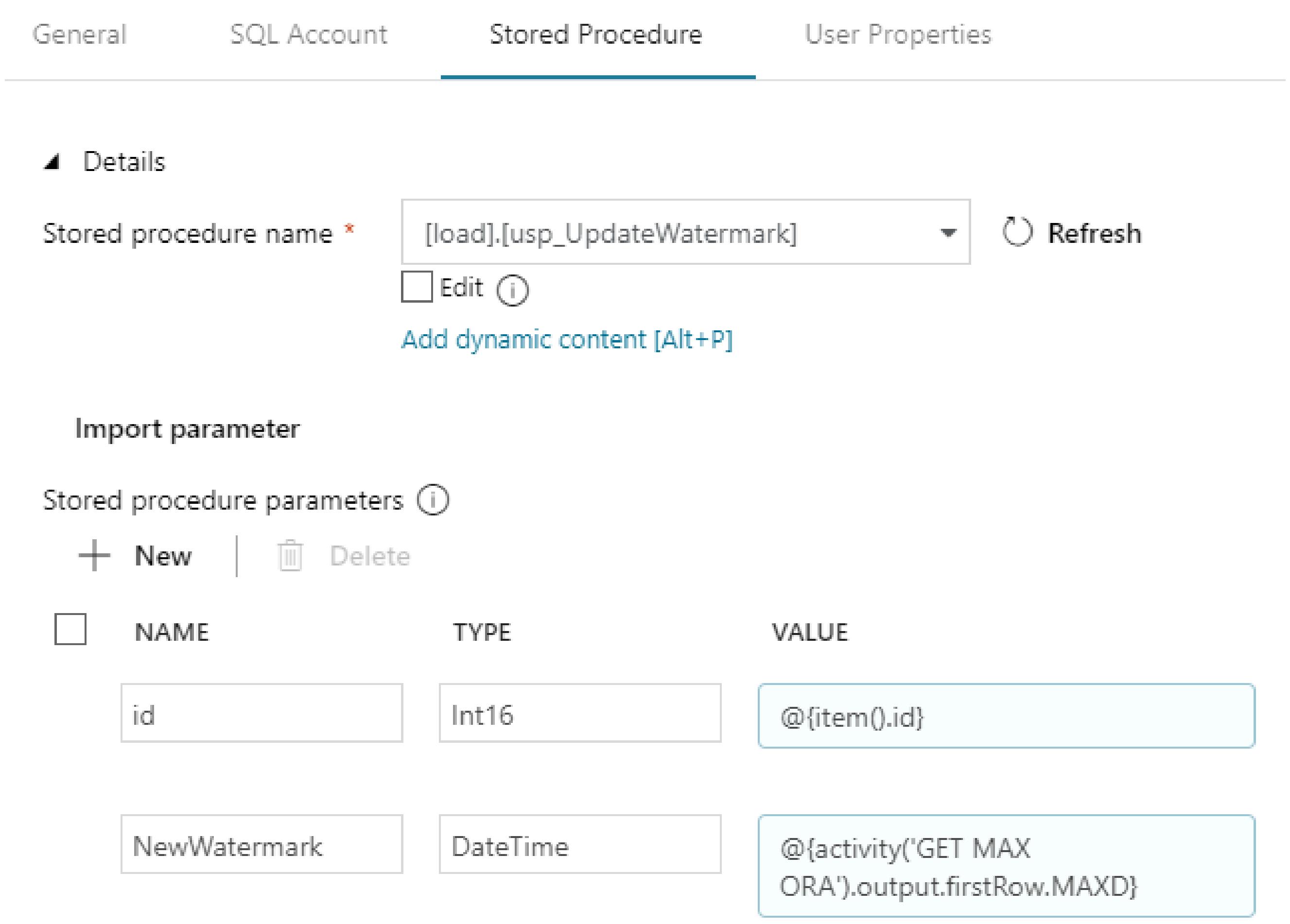
1. Drop into project, connect constraint from COPY ORA into it. Rename as "UPDATE WATERMARK ORA" and view properties.

2. In SQL Account set



Linked service * AzureSQL

3. Now go to "Stored Procedure", select our procedure name and click "**Import parameter**".
4. Now w have to pass values for procedure parametrs. And we will also parametrize them. Id should be @{item().id} and NewWatermatk has to be: @{activity('GET MAX ORA').output.firstRow.MAXD} .



Stored procedure name *: [load].[usp_UpdateWatermark]

Edit

Stored procedure parameters

NAME	TYPE	VALUE
id	Int16	@{item().id}
NewWatermark	DateTime	@{activity('GET MAX ORA').output.firstRow.MAXD}

And basically, that's all! This logic should copy rows from all Oracle tables defined in the configuration.

We can now test it. This can be done with "Debug" or just by triggering pipeline run.

If everything is working fine, we can just **copy/paste** all content from "**FOR EACH ORA**" into "**FOR EACH PG**".

Just remember to properly rename all activities to reflect new source/destination names (PG). Also, all parameters and SELECT queries have to be redefined. Luckily PostgreSQL support ISO dates out of the box.

Source code

Here are all components in JSON. You can use them to copy/paste logic directly inside ADF V2 code editor or save as files in GIT repository.

Below is source code for pipeline only. All other things can be downloaded in zip file in "Download all" at the bottom of this article.

Pipeline

JavaScript

```
{
"name": "LOAD DELTA",
"properties": {
"activities": [
{
"name": "GET CFG",
"type": "Lookup",
"policy": {
"timeout": "7.00:00:00",
"retry": 0,
"retryIntervalInSeconds": 30,
"secureOutput": false
},
"typeProperties": {
"source": {
"type": "SqlSource",
"sqlReaderQuery": {
"value": "SELECT * from @{pipeline().parameters.ConfigTable}\nIF
@@ROWCOUNT = 0 THROW 50000,'ojej...',1",
"type": "Expression"
}
},
"dataset": {
"referenceName": "SQL",
"type": "DatasetReference",
"parameters": {
"TableName": "dummy"
}
},
"firstRowOnly": false
}
},
{
"name": "FOR EACH ORA",
"type": "ForEach",
"dependsOn": [
{
"activity": "ORA CFG",
"dependencyConditions": [
"Succeeded"
]
}
],
"typeProperties": {
"items": {
"value": "@activity('ORA CFG').output.value",
"type": "Expression"
},
"isSequential": false,
"activities": [
{
"name": "COPY ORA",
"type": "Copy",
"dependsOn": [
{
"activity": "GET MAX ORA",
"dependencyConditions": [
"Succeeded"
]
}
],
"policy": {
"timeout": "7.00:00:00",
"retry": 0,
"retryIntervalInSeconds": 30,
"secureOutput": false
},
"userProperties": [
{
"name": "Destination",
"value": "@{item().DST_tab}"
}
],
"typeProperties": {
"source": {
"type": "OracleSource",
"oracleReaderQuery": {
"value": "SELECT \n @{item().Cols} FROM @{item().SRC_tab} \n\nWHERE
\n\n@{item().WatermarkColumn} > \nTO_DATE('@{item().WatermarkValue}',
'YYYY-MM-DD\"T\"HH24:MI:SS\"Z\"')\nAND\n@{item().WatermarkColumn}
<=\nTO_DATE('@{activity('GET MAX ORA').output.firstRow.MAXD}', 'YYYY-
MM-DD\"T\"HH24:MI:SS\"Z\"')",
"type": "Expression"
}
},
"sink": {
"type": "SqlSink",
"writeBatchSize": 10000,
"preCopyScript": {
"value": "TRUNCATE TABLE @{item().DST_tab}",
"type": "Expression"
}
},
"enableStaging": false,
"cloudDataMovementUnits": 0
},
"inputs": [
{
"referenceName": "ORA",
"type": "DatasetReference"
}
],
"outputs": [
{
"referenceName": "SQL",
"type": "DatasetReference",
"parameters": {
"TableName": {
"value": "@{item().DST_tab}",
"type": "Expression"
}
}
}
]
},
{
"name": "GET MAX ORA",
"type": "Lookup",
"policy": {
"timeout": "7.00:00:00",
"retry": 0,
"retryIntervalInSeconds": 30,
"secureOutput": false
},
"typeProperties": {
"source": {
"type": "OracleSource",
"oracleReaderQuery": {
"value": "SELECT MAX(@{item().WatermarkColumn}) as maxd FROM
@{item().SRC_tab} ",
"type": "Expression"
}
},
"dataset": {
"referenceName": "ORA",
"type": "DatasetReference"
}
}
},
{
"name": "UPDATE WATERMARK ORA",
"type": "SqlServerStoredProcedure",
"dependsOn": [
{
"activity": "COPY ORA",
"dependencyConditions": [
"Succeeded"
]
}
],
"policy": {
"timeout": "7.00:00:00",
"retry": 0,
"retryIntervalInSeconds": 30,
"secureOutput": false
},
"typeProperties": {
"storedProcedureName": "[load].[usp_UpdateWatermark]",
"storedProcedureParameters": {
"id": {
"value": {
"value": "@{item().id}",
"type": "Expression"
},
"type": "Int16"
},
"NewWatermark": {
"value": {
"value": "@{activity('GET MAX ORA').output.firstRow.MAXD}",
"type": "Expression"
},
"type": "DateTime"
}
}
},
"linkedServiceName": {
"referenceName": "AzureSQL",
"type": "LinkedServiceReference"
}
}
]
}
},
{
"name": "ORA CFG",
"type": "Filter",
"dependsOn": [
{
"activity": "GET CFG",
"dependencyConditions": [
"Succeeded"
]
}
],
"typeProperties": {
"items": {
"value": "@activity('GET CFG').output.value",
"type": "Expression"
},
"condition": {
"value": "@equals(item().SRC_name,'ORA')",
"type": "Expression"
}
}
},
{
"name": "PG CFG",
"type": "Filter",
"dependsOn": [
{
"activity": "GET CFG",
"dependencyConditions": [
"Succeeded"
]
}
],
"typeProperties": {
"items": {
"value": "@activity('GET CFG').output.value",
"type": "Expression"
},
"condition": {
"value": "@equals(item().SRC_name,'PG')",
"type": "Expression"
}
}
},
{
"name": "FOR EACH PG",
"type": "ForEach",
"dependsOn": [
{
"activity": "PG CFG",
"dependencyConditions": [
"Succeeded"
```

```
]
}
],
"typeProperties": {
"items": {
"value": "@activity('PG CFG').output.value",
"type": "Expression"
},
"isSequential": false,
"activities": [
{
"name": "Copy PG",
"type": "Copy",
"dependsOn": [
{
"activity": "GET MAX PG",
"dependencyConditions": [
"Succeeded"
]
}
],
"policy": {
"timeout": "7.00:00:00",
"retry": 0,
"retryIntervalInSeconds": 30,
"secureOutput": false
},
"userProperties": [
{
"name": "Destination",
"value": "@{item().DST_tab}"
}
],
"typeProperties": {
"source": {
"type": "RelationalSource",
"query": {
"value": "SELECT @{item().Cols} FROM @{item().SRC_tab} \n\nWHERE
\n\n@{item().WatermarkColumn} >
\n'@{item().WatermarkValue}'\nAND\n@{item().WatermarkColumn}
<=\n'@{activity('GET MAX PG').output.firstRow.MAXD}'",
"type": "Expression"
}
},
"sink": {
"type": "SqlSink",
"writeBatchSize": 10000,
"preCopyScript": {
```

```
"value": "TRUNCATE TABLE @{item().DST_tab}",
"type": "Expression"
}
},
"enableStaging": false,
"cloudDataMovementUnits": 0
},
"inputs": [
{
"referenceName": "PG",
"type": "DatasetReference"
}
],
"outputs": [
{
"referenceName": "SQL",
"type": "DatasetReference",
"parameters": {
"TableName": {
"value": "@{item().DST_tab}",
"type": "Expression"
}
}
}
]
},
{
"name": "GET MAX PG",
"type": "Lookup",
"policy": {
"timeout": "7.00:00:00",
"retry": 0,
"retryIntervalInSeconds": 30,
"secureOutput": false
},
"typeProperties": {
"source": {
"type": "RelationalSource",
"query": {
"value": "SELECT MAX(@{item().WatermarkColumn}) as maxd FROM
@{item().SRC_tab} ",
"type": "Expression"
}
},
"dataset": {
"referenceName": "PG",
"type": "DatasetReference"
}
```

```
}
},
{
"name": "UPDATE WATERMARK PG",
"type": "SqlServerStoredProcedure",
"dependsOn": [
{
"activity": "Copy PG",
"dependencyConditions": [
"Succeeded"
]
}
],
"policy": {
"timeout": "7.00:00:00",
"retry": 0,
"retryIntervalInSeconds": 30,
"secureOutput": false
},
"typeProperties": {
"storedProcedureName": "[load].[usp_UpdateWatermark]",
"storedProcedureParameters": {
"id": {
"value": {
"value": "@{item().id}",
"type": "Expression"
},
"type": "Int16"
},
"NewWatermark": {
"value": {
"value": "@{activity('GET MAX PG').output.firstRow.MAXD}",
"type": "Expression"
},
"type": "DateTime"
}
}
},
"linkedServiceName": {
"referenceName": "AzureSQL",
"type": "LinkedServiceReference"
}
}
]
}
}
],
"parameters": {
```

```
"ConfigTable": {
"type": "String",
"defaultValue": "load.cfg"
}
}
}
}
```

Download all

IncrementalCopy_ADFv2.zip