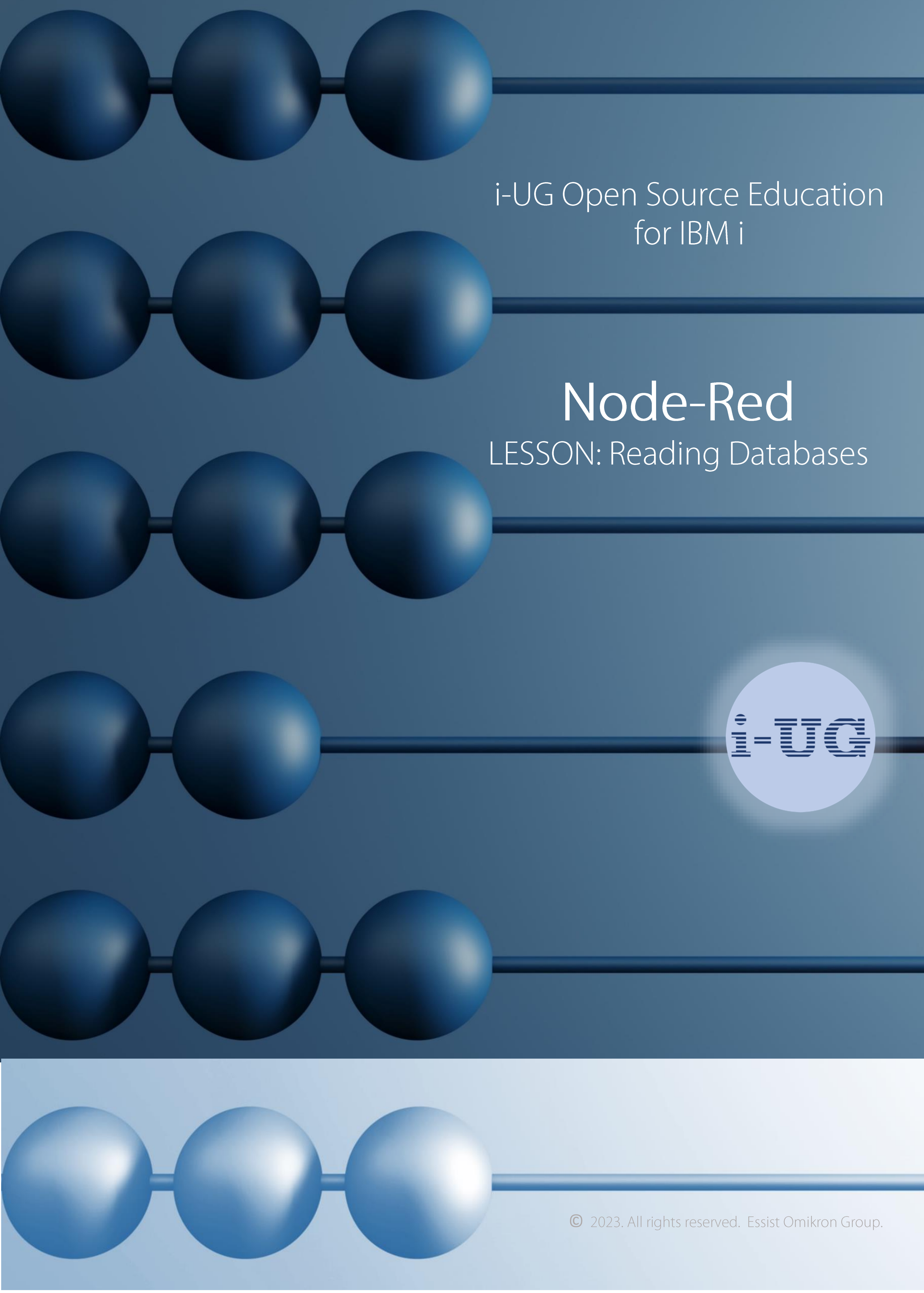


i-UG Open Source Education for IBM i

Node-Red

LESSON: Reading Databases

1. A Database

One of the many useful functions of Node-Red is the ability to read, update and add to a database. This is a crucial part of your education as storing information is vital to any company, and reading it is even more valuable. Along the way, we need to maintain and update the data to keep it relevant to our objectives.

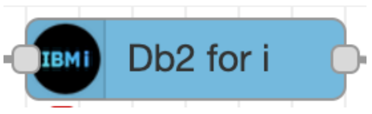
We will expand a little on the previous lesson where we created a user interface that makes the database more manageable on a direct basis. However, the Database will take on a life of its own as we ramp up the data we send to it and demand information from it over time.

But first you need to connect a database to node-red, so let's start there.

2. Connect to the Database

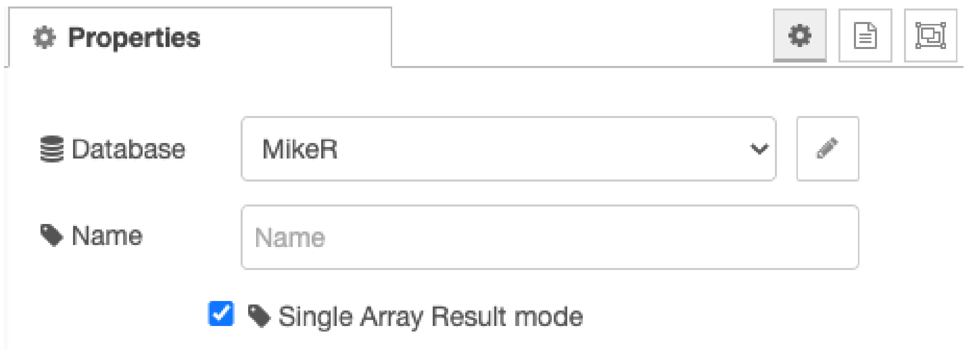
For this task, we're going to need a node that can connect into the database we're going to be using. For this exercise we will be using the Db2 for i node. This links you into a world class IBM DB2 database, which will be the database were using as an example.

Select the Db2 for i node and bring it on to the palette.



Db2 for i node

Open up the node and give our Database Connection a name in the Database Box. The Name box is for documentation so that you know which database we are connecting to (in future, you may be accessing many databases0. In this example ,if the leave the name blank, it actually defaults to a completely satisfactory name for us – Db2.



Properties

Database: MikeR

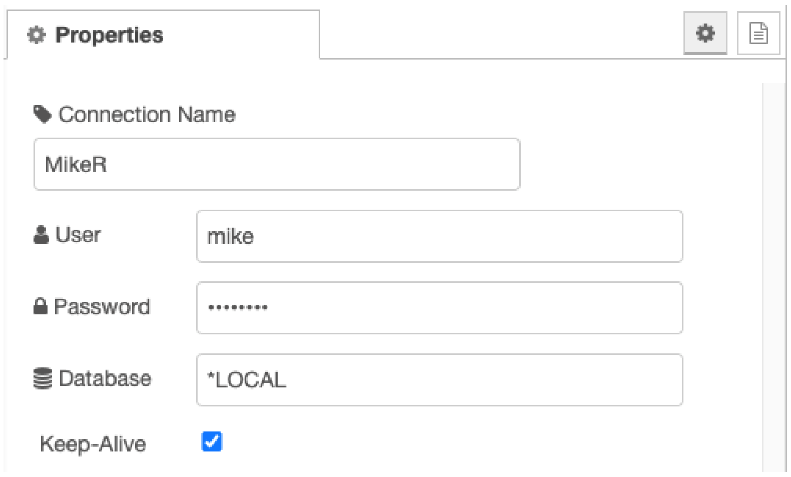
Name: Name

☑ Single Array Result mode

The Database is very secure, but we will be using minimal security and will need to use our secure profile to log in to the DB2 Database on the IBM Power system.

The credential you will use are the credentials given to you at the start of the Education.

Fill them here as shown: -



Properties

Connection Name: MikeR

User: mike

Password: ••••••••

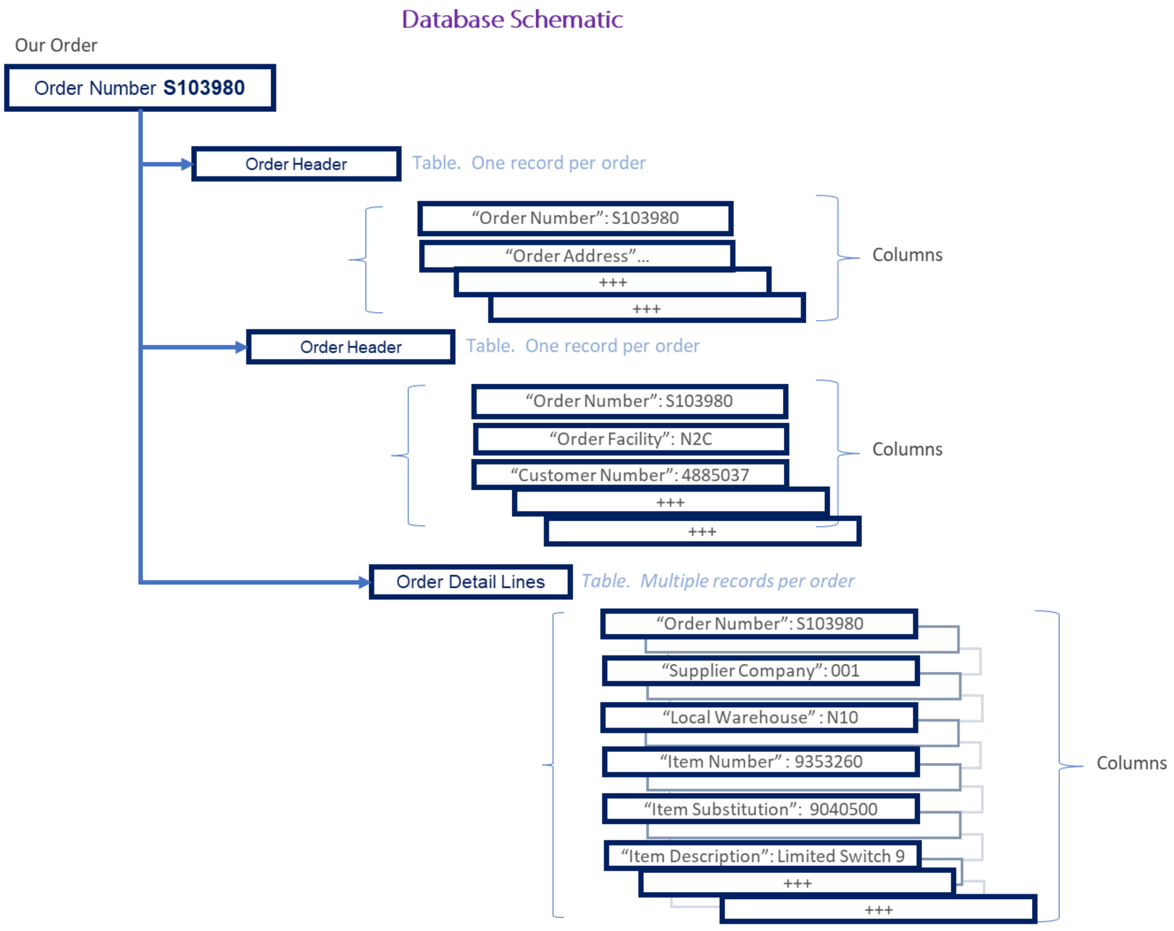
Database: *LOCAL

Keep-Alive: ✓

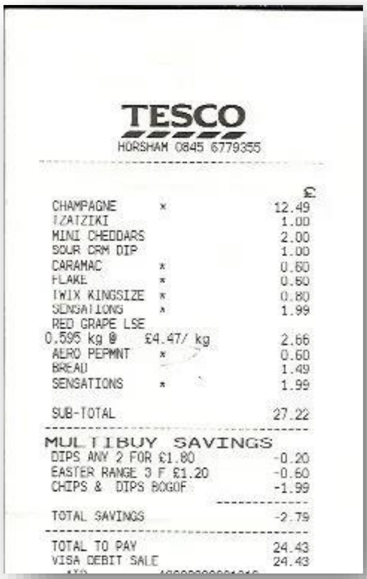
Using *LOCAL as the address for the Database will result in us attaching to the main DB2 database on the IBM POWER system running IBM i that we are running Node-RED on in this education. The credentials you key in here will secure you access to the IBM DB2 database.

The next thing we need to do is to ask the database to give us some information. We will ask the database for information about a recent order that was placed. What we need to know is who the order was for and when. We also need to know the details of the order.

This classical information that is always needed and a good bedrock on what we will need to look for in the future. Here is the simple schematic of this part of the Database: -



If you think of any invoice you may have received, even a restaurant bill or a Supermarket bill, you will have an invoice or a receipt that follows this pattern. Information about the restaurant or Store at the top – once, followed by the items you have bought or eaten in a list below. Many of these secondary lines will be different items, but all are represented in the same format.



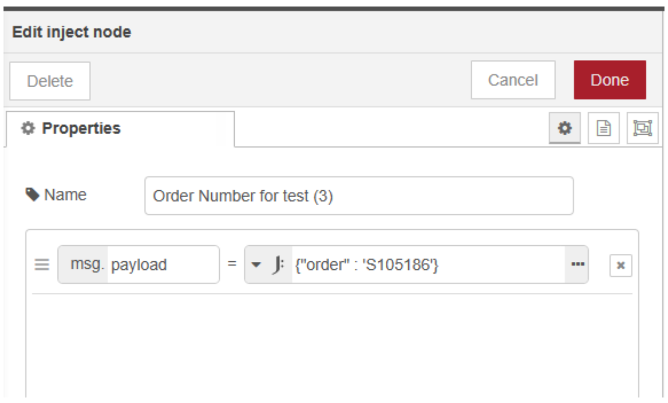
```
TESCO
HORSHAM 0845 6779355
------------------------------------
                                  £
CHAMPAGNE       *             12.49
TZATZIKI                       1.00
MINI CHEDDARS                  2.00
SOUR CRM DIP                   1.00
CARAMAC         *              0.60
FLAKE           *              0.60
TWIX KINGSIZE   *              0.80
SENSATIONS      *              1.99
RED GRAPE LSE
0.595 kg @    £4.47/ kg        2.66
AERO PEPMNT     *              0.60
BREAD                          1.49
SENSATIONS      *              1.99

SUB-TOTAL                     27.22
------------------------------------
MULTIBUY SAVINGS
DIPS ANY 2 FOR £1.80          -0.20
EASTER RANGE 3 F £1.20        -0.60
CHIPS &  DIPS BOGOF           -1.99
                      -------------
TOTAL SAVINGS                 -2.79
------------------------------------
TOTAL TO PAY                  24.43
VISA DEBIT SALE               24.43
```

So let's read our database: -

a) Stage 1 – Header Information

This first stage fetches the order header information on the Order we are interested in from the Sales Order Headers table - ORDERHEADX.

Drag in an inject node and open it: -



Edit inject node

Delete | Cancel | Done

Properties

Name: Order Number for test (3)

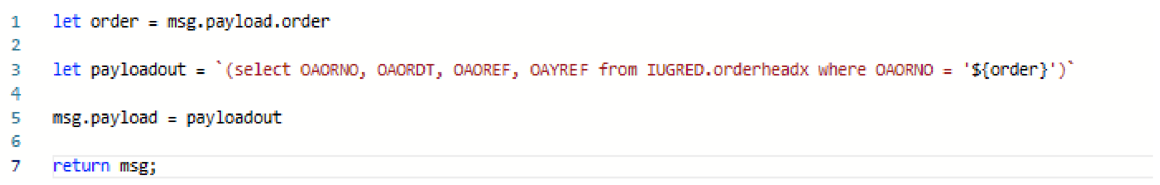
msg. payload = J: {"order" : 'S105186'}

Change the msg.payload to a JSONata expression (J:) and enter the following expression into the text field: {"order" : 'S105186'} (This is in fact a JSON object).

This will act as the specific order we're looking for. Later we will change this mechanism so the order can be typed into the dashboard we will create.

The Inject node sends a JSON object in the **payload** msg "order" : 'S105186'. This is then retrieved as an extension to **payload…** **msg.payload.order** which = S105186

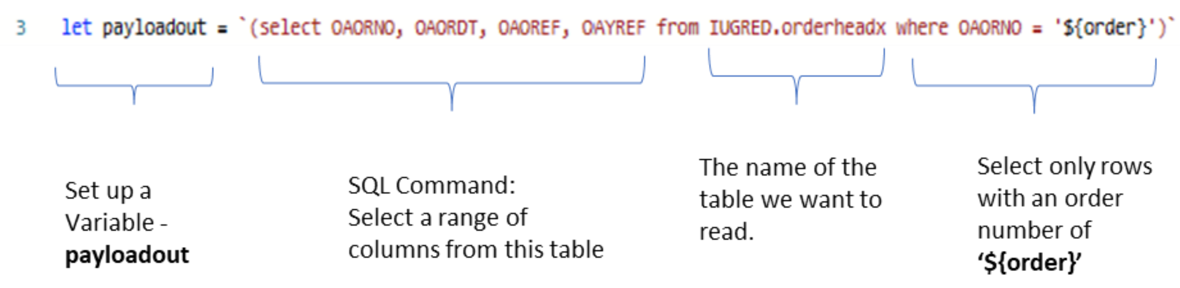
Next, connect a function node, the purpose of this node will be to find the header for the information we are looking for. Into this node, we will insert the following code.



```
let order = msg.payload.order

let payloadout = `(select OAORNO, OAORDT, OAOREF, OAYREF from IUGRED.orderheadx where OAORNO = '${order}')`

msg.payload = payloadout

return msg;
```

This is essentially creating an SQL statement to read our DB2 Table. First we set up a variable named **'order'** and we put the Payload that we have received from the Inject node into it – **msg.payload.order**.

Then we are going to construct an SQL Statement to retrieve some specific data from within the database.



```
let payloadout = `(select OAORNO, OAORDT, OAOREF, OAYREF from IUGRED.orderheadx where OAORNO = '${order}')`
```

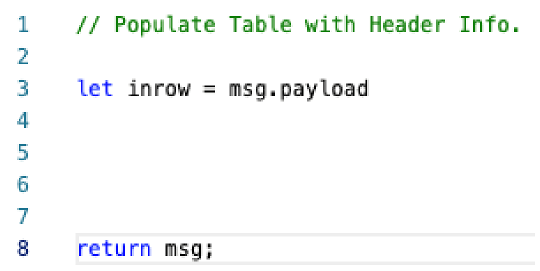
Set up a Variable - **payloadout**
SQL Command: Select a range of columns from this table
The name of the table we want to read.
Select only rows with an order number of **'${order}'**

We will set up and populate a variable – payloadout, and build the SQL Statement as in the figure above.

We will use the order number which was injected and now resides in a variable called **order**

To make the SQL Statement recognise this order number variable, we need to set it as a substitution parameter in the SQL Statement. By wrapping the variable name in { }, preceding it with '$' and then we need to wrap all this in single quotes because this column is alphameric. Now, the SQL statement will recognise this as a substitution parameter and effectively insert '*S105186*' into the SQL *'where'* parameter.

Next up we will connect up to the DB2 for i node, meaning this command will be sent into the database to retrieve the information.



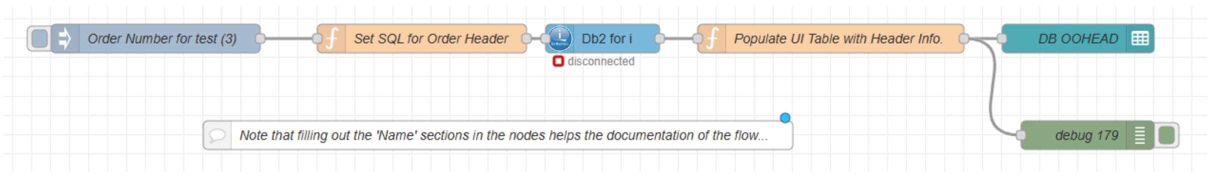
```
// Populate Table with Header Info.

let inrow = msg.payload





return msg;
```

Following that will be another function node which will populate our table, making the information presentable. It has the data in the payload **msg** as that was the last statement that we used in the previous Function node (**msg.payload** = **payloadout**).

To finish off this stage we will attach a table node to the end of the flow. (This will require creating a section on the dashboard for this section. If necessary, refer to the previous lesson to see the step by step guide on how to do this).

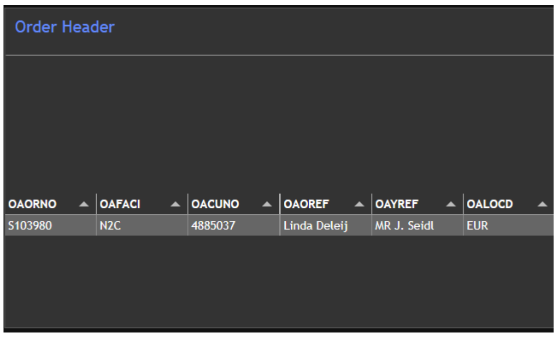
Finally, we'll add a Debug node just to make sure the correct information is coming through.

It should end up looking like this: -


Order Number for test (3)
Set SQL for Order Header
Db2 for i
disconnected
Populate UI Table with Header Info.
DB OOHEAD
debug 179
Note that filling out the 'Name' sections in the nodes helps the documentation of the flow...

Hit Deploy…

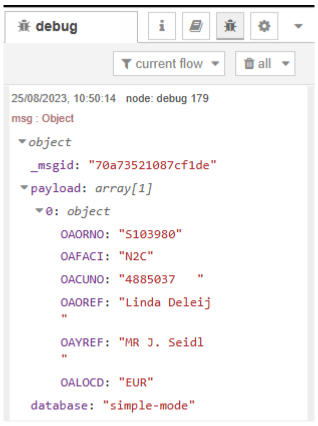
If you now look on the UI section of Node-RED (172.16.1.10:18nnn/ui), you will see the table represented.



Order Header

OAORNO	OAFACI	OACUNO	OAOREF	OAYREF	OALOCD
S103980	N2C	4885037	Linda Deleij	MR J. Seidl	EUR

Table node does quite a good job of presenting the table data.



```
debug
current flow   all
25/08/2023, 10:50:14   node: debug 179
msg : Object
▾object
  _msgid: "70a73521087cf1de"
 ▾payload: array[1]
   ▾0: object
      OAORNO: "S103980"
      OAFACI: "N2C"
      OACUNO: "4885037    "
      OAOREF: "Linda Deleij
      "
      OAYREF: "MR J. Seidl
      "
      OALOCD: "EUR"
  database: "simple-mode"
```

The Debug shows the actual data that was retrieved from the database and sent to the Table node.

b) Stage 2 – Order Details

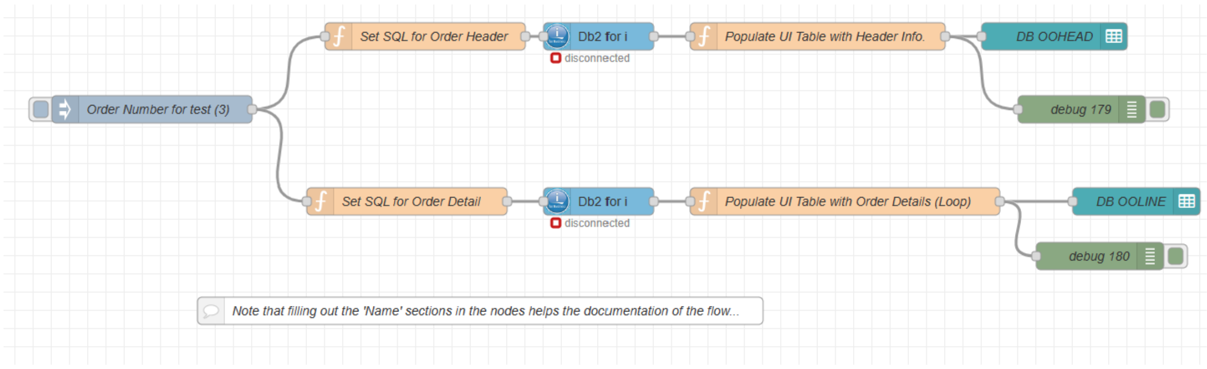
The second stage gets the order lines that match with the order details. It's much like the first stage.

Copy the flow you have just created to get the Order Header and place it on the Palette just under the current flow. We are going to modify the function nodes slightly so that we read the Order Detail Table rather than the Order Header, but other than that, it will follow exactly the same pattern.

However, it is often the case that unlike the Order Header, where we just return one row, on the Order Details we could return very many rows.

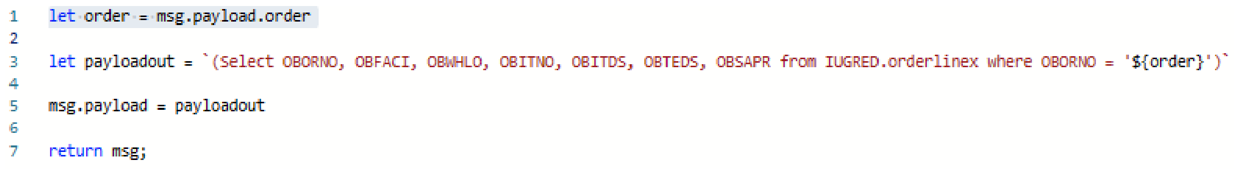
In this case, the SQL will return an Array of Objects, which we will need to loop through as we send each Order Detail line to the Report node. Don't forget to change the 'name' on the nodes to show that we are now dealing with Order details.

It is going to end up looking like this: -


Set SQL for Order Header
Db2 for i
disconnected
Populate UI Table with Header Info.
DB OOHEAD
Order Number for test (3)
debug 179
Set SQL for Order Detail
Db2 for i
disconnected
Populate UI Table with Order Details (Loop)
DB OOLINE
debug 180
Note that filling out the 'Name' sections in the nodes helps the documentation of the flow...

Change the SQL Function Node for the Order details to look at this new Order Detail Table. Note that we are still using the Order Number from the initial injection node. This will be injected into both the Order Header flow and the Order Detail flow at the same time. Useful. Add in some Debug nodes. After each Function Node that is related to populating the UI Tables

Change the SQL in the Order Detail Function Node to look like this: -

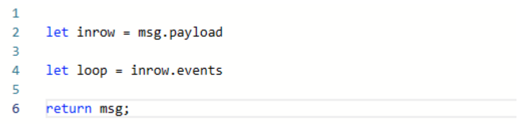


```
let order = msg.payload.order

let payloadout = `(Select OBORNO, OBFACI, OBWHLO, OBITNO, OBITDS, OBTEDS, OBSAPR from IUGRED.orderlinex where OBORNO = '${order}')`

msg.payload = payloadout

return msg;
```

This reflects the rows that are characteristic of an order detail and are stored in the sales Order Lines table – ORDLINEX in library IUGRED.

Now because we are using the Table Node and a JSON array, we don't need to learn how to set up a loop; we can do that at a later stage. We will simply send the Array to the Table Node and the Table Node will split this into the individual objects and display these as discrete records.

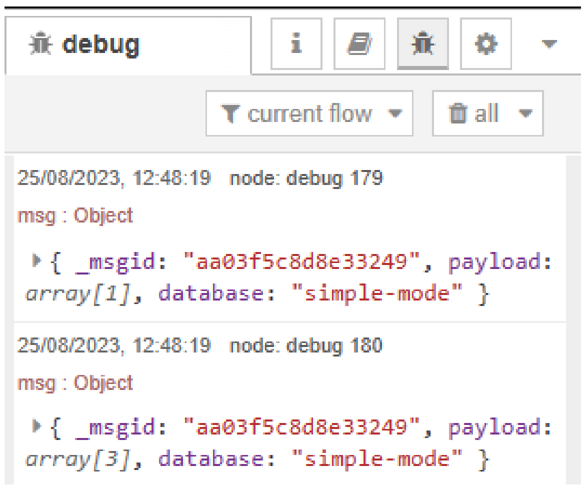
In the Populate function node for the Order Details, change the code to this: -



```

let inrow = msg.payload

let loop = inrow.events

return msg;
```

Deploy.

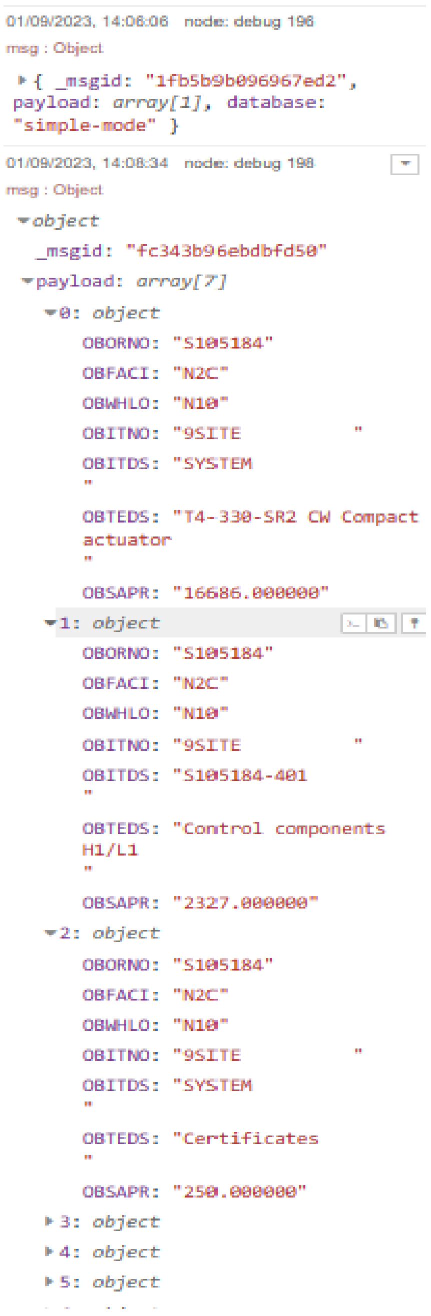
The Debug panel should look like this: -



```
25/08/2023, 12:48:19   node: debug 179
msg : Object
▸{ _msgid: "aa03f5c8d8e33249", payload: array[1], database: "simple-mode" }
25/08/2023, 12:48:19   node: debug 180
msg : Object
▸{ _msgid: "aa03f5c8d8e33249", payload: array[3], database: "simple-mode" }
```

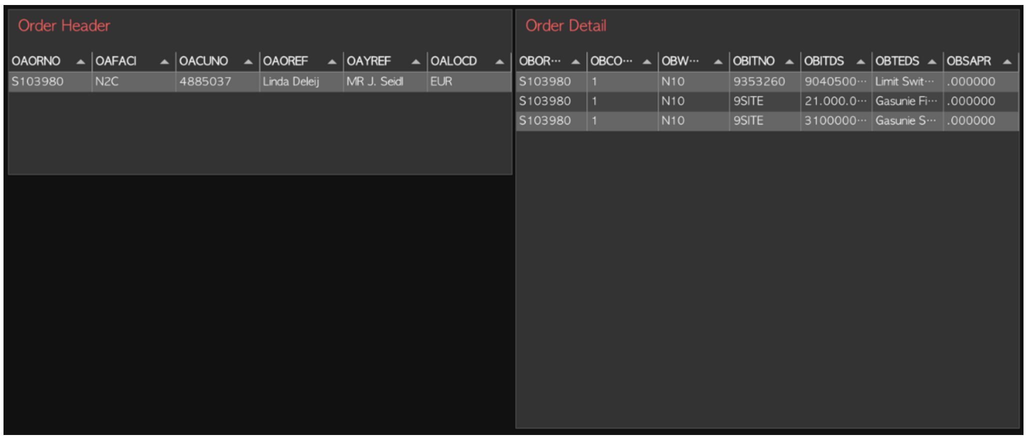
Click on the arrow on the lower debug to expand, and keep on expanding by clicking further arrows that appear.

What you will see is the whole Array laid out as discrete Objects. These Objects are the rows from the Order Detail table: -



```
01/09/2023, 14:06:06   node: debug 196
msg : Object
 ▸{ _msgid: "1fb5b9b096967ed2",
 payload: array[1], database:
 "simple-mode" }
01/09/2023, 14:08:34   node: debug 198
msg : Object
 ▾object
   _msgid: "fc343b96ebdbfd50"
  ▾payload: array[7]
    ▾0: object
       OBORNO: "S105184"
       OBFACI: "N2C"
       OBWHLO: "N10"
       OBITNO: "9SITE           "
       OBITDS: "SYSTEM
       "
       OBTEDS: "T4-330-SR2 CW Compact
       actuator
       "
       OBSAPR: "16686.000000"
    ▾1: object
       OBORNO: "S105184"
       OBFACI: "N2C"
       OBWHLO: "N10"
       OBITNO: "9SITE           "
       OBITDS: "S105184-401
       "
       OBTEDS: "Control components
       H1/L1
       "
       OBSAPR: "2327.000000"
    ▾2: object
       OBORNO: "S105184"
       OBFACI: "N2C"
       OBWHLO: "N10"
       OBITNO: "9SITE           "
       OBITDS: "SYSTEM
       "
       OBTEDS: "Certificates
       "
       OBSAPR: "250.000000"
    ▸3: object
    ▸4: object
    ▸5: object
```

This should provide a dashboard that looks like this when deployed and the order is injected: -



Order Header

OAORNO	OAFACI	OACUNO	OAOREF	OAYREF	OALOCD
S103980	N2C	4885037	Linda Deleij	MR J. Seidl	EUR

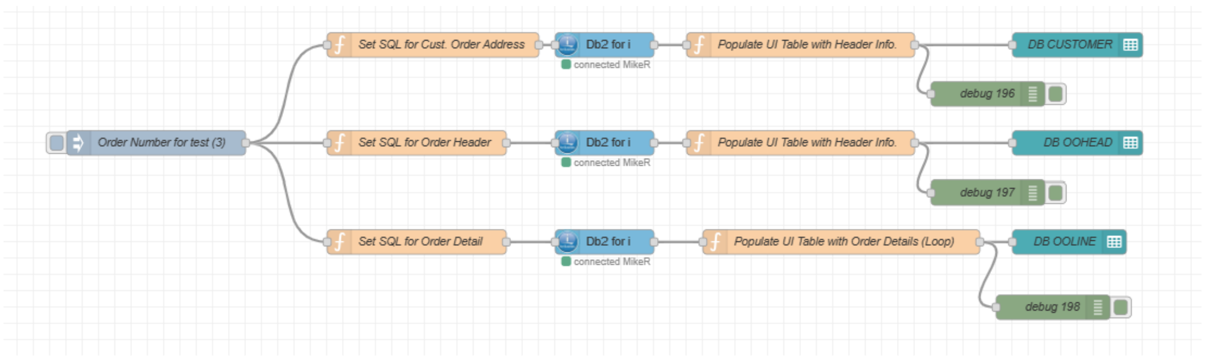
Order Detail

OBOR···	OBCO···	OBW···	OBITNO	OBITDS	OBTEDS	OBSAPR
S103980	1	N10	9353260	9040500···	Limit Swit···	.000000
S103980	1	N10	9SITE	21.000.0···	Gasunie Fi···	.000000
S103980	1	N10	9SITE	3100000···	Gasunie S···	.000000

Additional information for the user

Well, the user might need to know the delivery address of this order so let's also add that information.

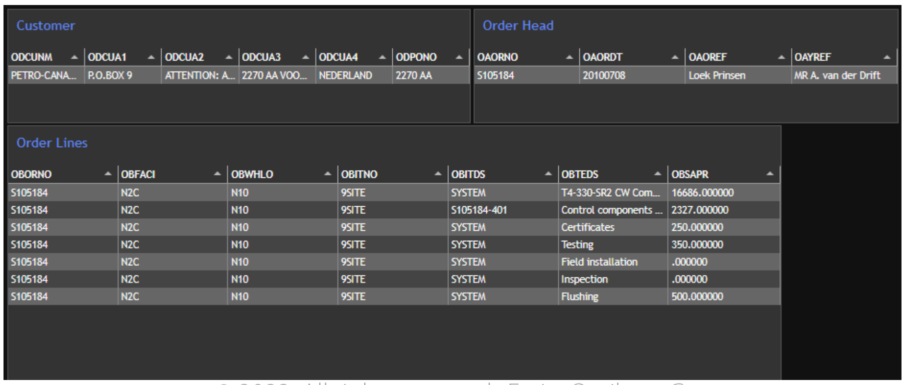
Copy the Order Header set part of the Flow and add this in at the top of the current Flow. It should now look something like this: -



Change the Table Node to drop this address information into its own part of the same UI tab, let's call it 'Customer'. The only other change we need to make is to change the SQL Function Node to run an SQL against the correct table – CUSORDADD. Node-RED will take care of the rest…

```
let order = msg.payload.order

let payloadout = `(select ODCUNM, ODCUA1, ODCUA2, ODCUA3, ODCUA4, ODPONO FROM IUGRED.cusordadd where ODORNO = '${order}')`

msg.payload = payloadout

return msg;
```

The UI should start to look like this, with three discrete areas: -



Customer

ODCUNM	ODCUA1	ODCUA2	ODCUA3	ODCUA4	ODPONO
PETRO-CANA...	P.O.BOX 9	ATTENTION: A...	2270 AA VOO...	NEDERLAND	2270 AA

Order Head

OAORNO	OAORDT	OAOREF	OAYREF
S105184	20100708	Loek Prinsen	MR A. van der Drift

Order Lines

OBORNO	OBFACI	OBWHLO	OBITNO	OBITDS	OBTEDS	OBSAPR
S105184	N2C	N10	9SITE	SYSTEM	T4-330-SR2 CW Com...	16686.000000
S105184	N2C	N10	9SITE	S105184-401	Control components ...	2327.000000
S105184	N2C	N10	9SITE	SYSTEM	Certificates	250.000000
S105184	N2C	N10	9SITE	SYSTEM	Testing	350.000000
S105184	N2C	N10	9SITE	SYSTEM	Field installation	.000000
S105184	N2C	N10	9SITE	SYSTEM	Inspection	.000000
S105184	N2C	N10	9SITE	SYSTEM	Flushing	500.000000

Try changing the Order Numbers in the Inject node and see the details on the UI Table change: -

S105224

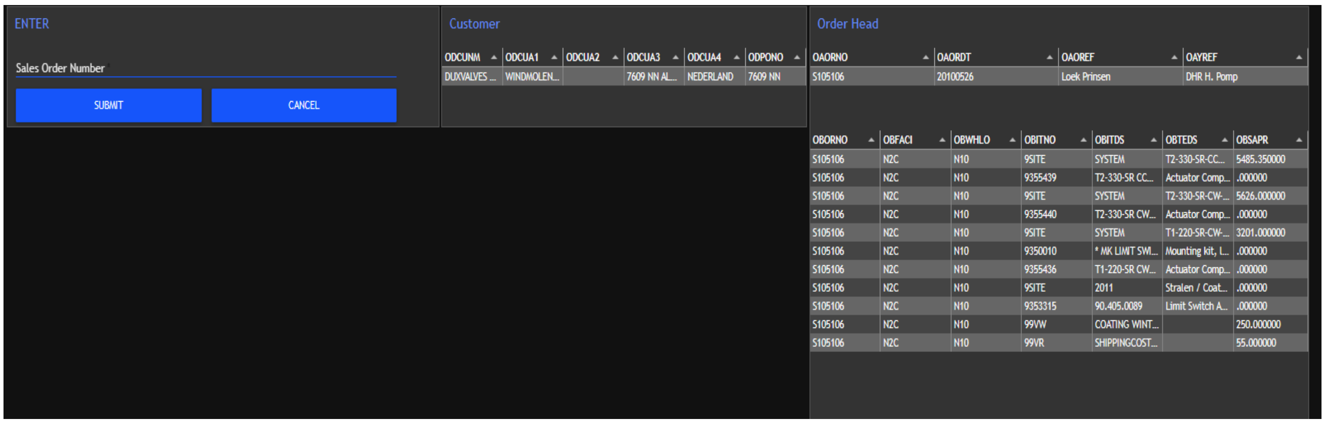
S105106

Remember, you'll have to Deploy the flows after each change.

Ease of use for the User

To make it easier for the user to just select an Order Number and get all this information for that number, you could now combine this with an Forms Node and allow the user to simply input the Order Number to change the information being presented.

This can then replace the Inject Node and you can simply enter new Order Numbers and see the Tables Change without ever needing to re Deploy the Flow: -



ENTER

Sales Order Number

SUBMIT CANCEL

Customer

ODCUNM	ODCUA1	ODCUA2	ODCUA3	ODCUA4	ODPONO
DUXVALVES ...	WINDMOLEN...		7609 NN AL...	NEDERLAND	7609 NN

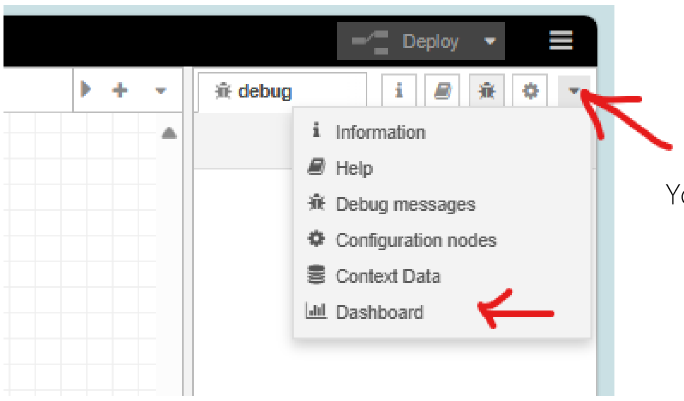
Order Head

OAORNO	OAORDT	OAOREF	OAYREF
S105106	20100526	Loek Prinsen	DHR H. Pomp

OBORNO	OBFACI	OBWHLO	OBITNO	OBITDS	OBTEDS	OBSAPR
S105106	N2C	N10	9SITE	SYSTEM	T2-330-SR-CC...	5485.350000
S105106	N2C	N10	9355439	T2-330-SR CC...	Actuator Comp...	.000000
S105106	N2C	N10	9SITE	SYSTEM	T2-330-SR-CW-...	5626.000000
S105106	N2C	N10	9355440	T2-330-SR CW...	Actuator Comp...	.000000
S105106	N2C	N10	9SITE	SYSTEM	T1-220-SR-CW-...	3201.000000
S105106	N2C	N10	9350010	* MK LIMIT SWI...	Mounting kit, L...	.000000
S105106	N2C	N10	9355436	T1-220-SR CW...	Actuator Comp...	.000000
S105106	N2C	N10	9SITE	2011	Stralen / Coat...	.000000
S105106	N2C	N10	9353315	90.405.0089	Limit Switch A...	.000000
S105106	N2C	N10	99VW	COATING WINT...		250.000000
S105106	N2C	N10	99VR	SHIPPINGCOST...		55.000000

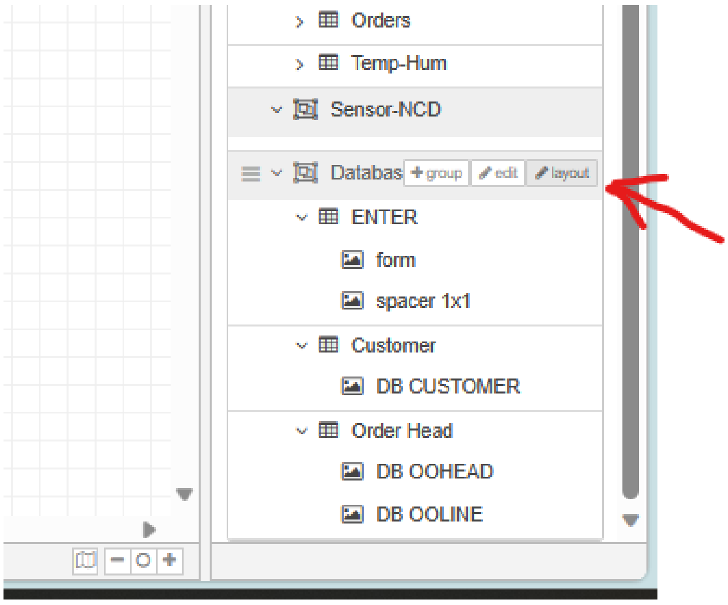
Just go back and look at the 'Engaging with the UI – Outputs lesson to refresh on how we set up a Forms Node. Just use one element again but this time call it Order Number.

Try changing the layout on the UI to make it more sensible.



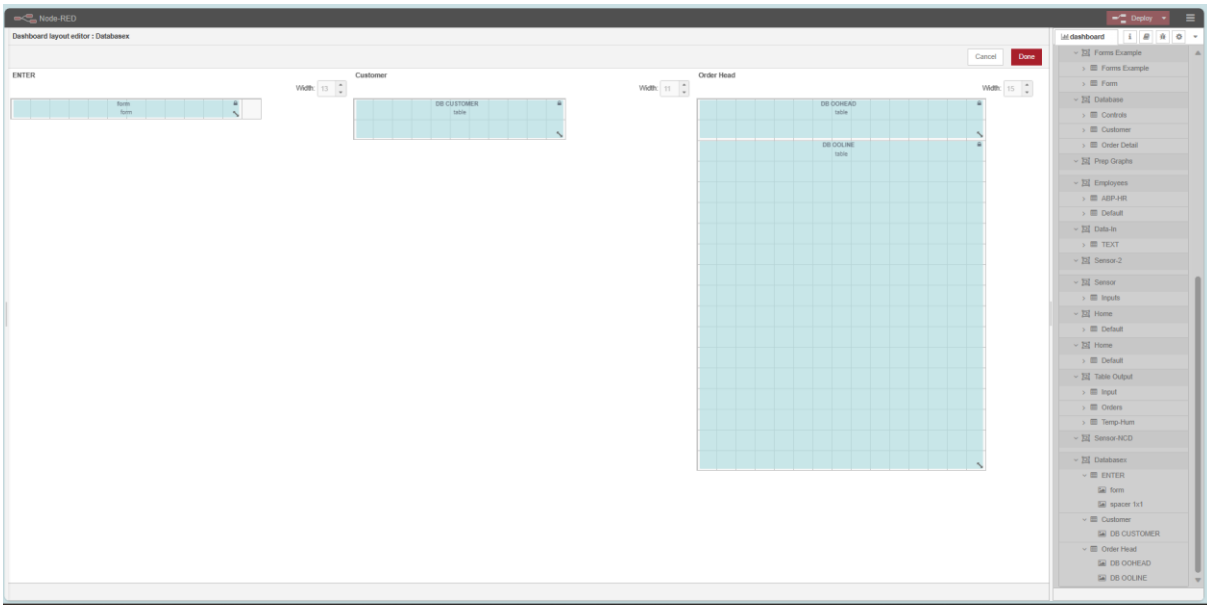
You can do this by selecting the Dashboard: -

And then scrolling down to the Databasex Tab and clicking the edit button.



Orders
Temp-Hum
Sensor-NCD
Databas + group edit layout
ENTER
form
spacer 1x1
Customer
DB CUSTOMER
Order Head
DB OOHEAD
DB OOLINE

Unlock the various elements and move them around. Deploy and look at the UI to see if it has worked. Keep going around this cycle.



With the new Forms Node included, now use the UI to enter various Sales Order numbers to retrieve the respective data from the three tables in our mini DB2 Database: - e.g.

S105224

S105106

Remember, now you *DON'T* have to keep deploying the Flow. It will just run forever…

c) Stage 3 – write out to a CSV file

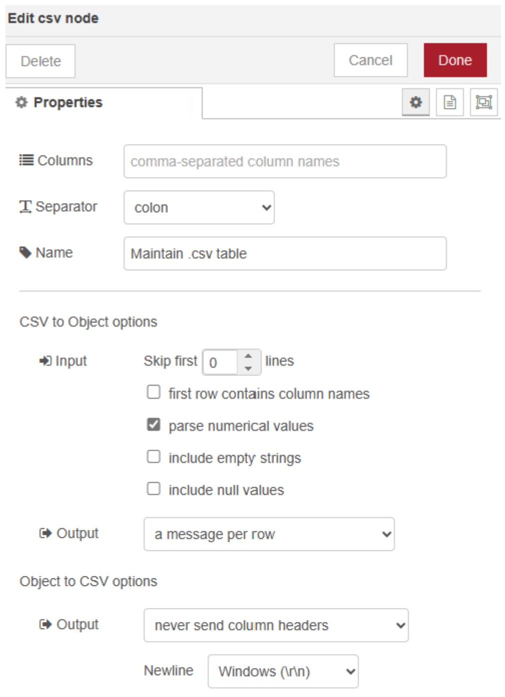
Now that we have done the heavy lifting and can send data to a UI Table, let's also write it to a CSV file so we can send this to someone, perhaps as a mail attachment in our daily job.

We will need to add two more Nodes to our current Flow: -

The CSV Node:

The Write File Node:

First, set up the CSV Node like this: -



Edit csv node

Delete | Cancel | Done

Properties

- Columns: comma-separated column names
- Separator: colon
- Name: Maintain .csv table

CSV to Object options

- Input: Skip first 0 lines
  - ☐ first row contains column names
  - ☑ parse numerical values
  - ☐ include empty strings
  - ☐ include null values
- Output: a message per row

Object to CSV options

- Output: never send column headers
- Newline: Windows (\r\n)

As we are sending an Array to this Node, we don't need to describe the columns, this will be taken from the Array. We will give the Node a name as this is good practice.

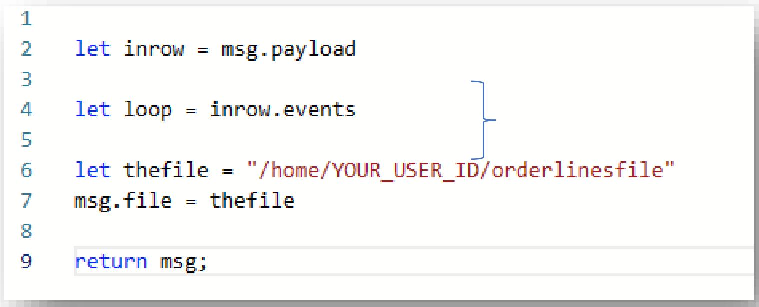
We are asking that each of the elements in the Array are separated by a colon (:) as it sends the string in msg.payload to the Write File Node.

We are asking the Node to parse any numerical variables.

One message per row.

Don't send column headings.

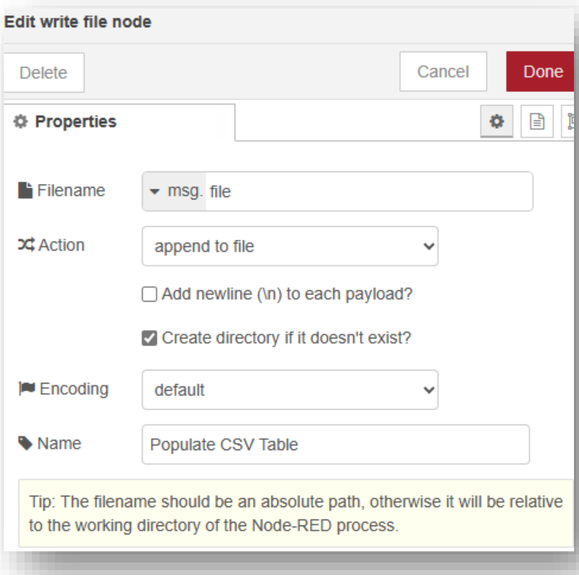
Hiust to make things a bit more dynamic, let's just change the Function Node for the populating of the Order Details UI Table and specify a Directory and name for the .csv file: -
Change it as follows: -



```

let inrow = msg.payload

let loop = inrow.events

let thefile = "/home/YOUR_USER_ID/orderlinesfile"
msg.file = thefile

return msg;
```

Add these lines. In the Path description, change the YOUR_USER_ID to your actual user ID so that this appears in *your* directory.

Next, let's set up the Write File Node.



Edit write file node

Delete | Cancel | Done

Properties

Filename	msg. file
Action	append to file
	☐ Add newline (\n) to each payload?
	☑ Create directory if it doesn't exist?
Encoding	default
Name	Populate CSV Table

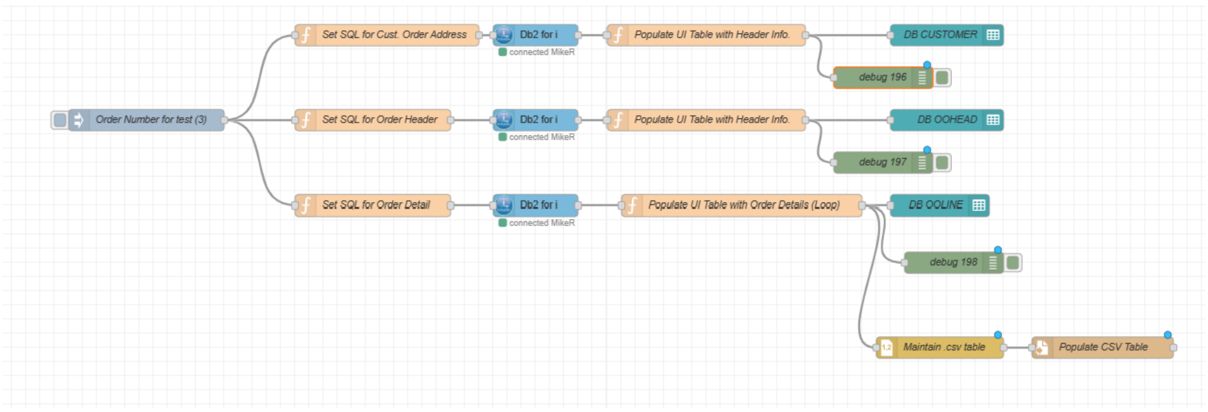
Tip: The filename should be an absolute path, otherwise it will be relative to the working directory of the Node-RED process.

Rather than just keying in the Path here, we will accept the Path from the **msg.file** that we set up in the earlier Function Node.
If there is a Table already there, add to it each time I run this Flow.

If there is no directory entry or Table as we start, create one.

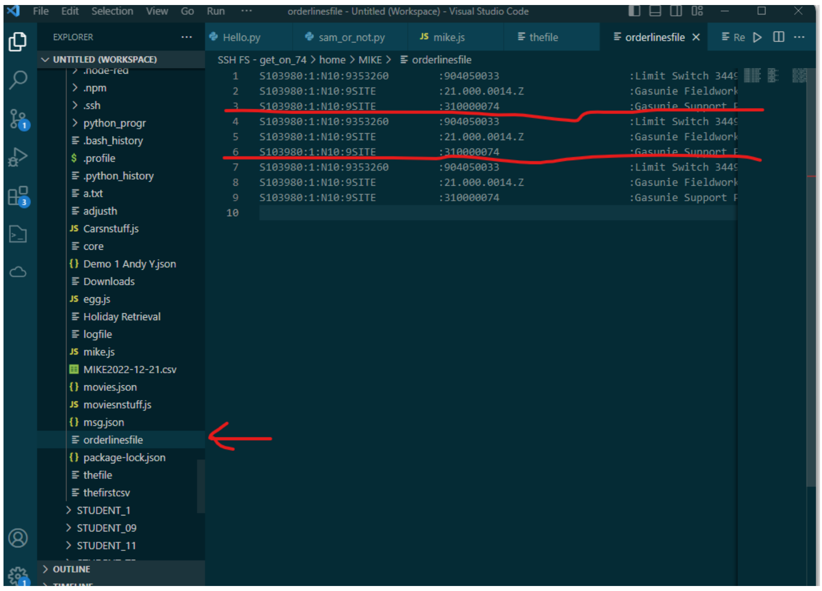
Give it a Name…

Link these up so it looks something like this: -



Deploy…

Now take a look at the UI, the Debug and the new *orderlinesfile.csv* file that we have created in your directory. I'm using **Visual Code** to view the .csv file.



```
S103980:1:N10:9353260      :904050033                  :Limit Switch 344
S103980:1:N10:9SITE        :21.000.0014.Z              :Gasunie Fieldwork
S103980:1:N10:9SITE        :310000074                  :Gasunie Support F
S103980:1:N10:9353260      :904050033                  :Limit Switch 344
S103980:1:N10:9SITE        :21.000.0014.Z              :Gasunie Fieldwork
S103980:1:N10:9SITE        :310000074                  :Gasunie Support F
S103980:1:N10:9353260      :904050033                  :Limit Switch 344
S103980:1:N10:9SITE        :21.000.0014.Z              :Gasunie Fieldwork
S103980:1:N10:9SITE        :310000074                  :Gasunie Support F
```

Not bad.

You have reached into a major IBM DB2 Database, displayed the Sales Order information on the User Interface/Dashboard and also created a .csv table in your directory that can be used elsewhere in the business.

And with that, the lesson is concluded. If you didn't manage to make one of the flows work then you can copy one of the flows from our GitHub repository here, and see where you went wrong. These functions, although they seem simple, are the backbone of many useful applications of Node-Red for major and small businesses.

https://github.com/JoshRyanEOG/i-UG_Education_Node-Red