



i-UG Open Source Education  
for IBM i

# Node-Red

LESSON 8: Writing to and  
Updating Databases



i-UG

## 1. A Database

We have already covered how important it is to read, update and add to a database, and how Node-RED has the ability to help this process. This is a crucial part of your education as storing and maintaining information is vital to any company.

We will expand a little on the previous lesson where we were reading *from* Databases. Much of the processing and the use of Nodes will be very similar, but the SQL we will use to manipulate the Database will, of course, differ as now we are not just selecting, we are updating and adding.

We have already connected a database to node-red, but we can just re-cover that now.

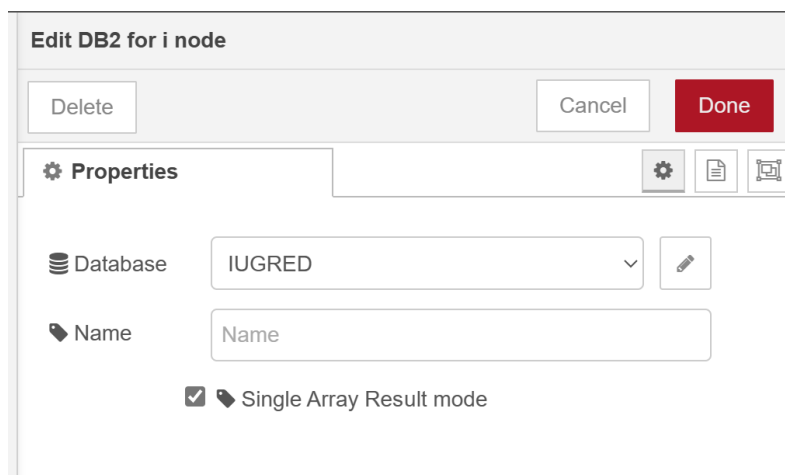
## 2. Connect to the Database

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 we're using as an example.

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



Open up the Node. It should already populate the configuration as we have already attached to this database in earlier lessons. If not, 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 databases). In this example, if we leave the name blank, it actually defaults to a completely satisfactory name for us – Db2 for i.

The image shows the 'Edit DB2 for i node' configuration panel. At the top, there are three buttons: 'Delete', 'Cancel', and 'Done'. Below these is a 'Properties' section with a gear icon and three sub-icons (gear, document, and a screen with a cursor). The 'Database' field is a dropdown menu showing 'IUGRED' with a pencil icon to its right. The 'Name' field is a text input box containing the placeholder text 'Name'. At the bottom, there is a checkbox labeled 'Single Array Result mode' which is checked.

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.

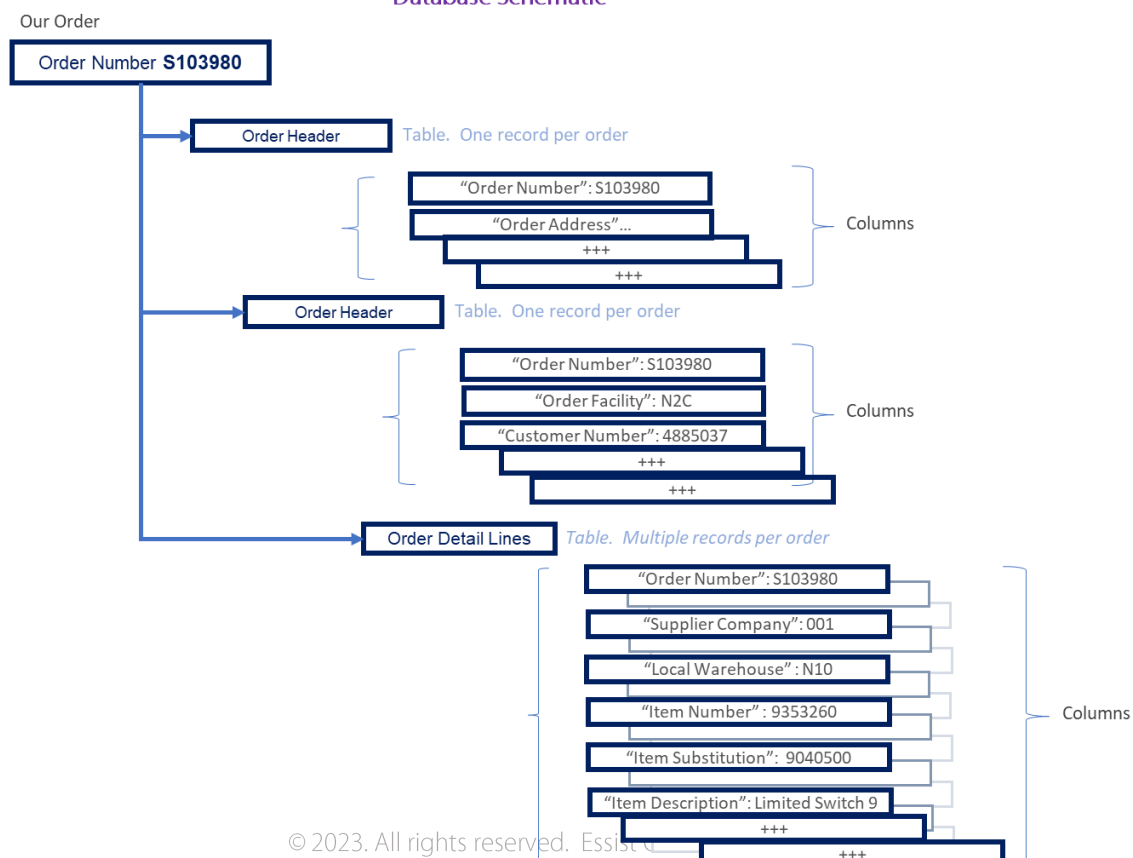
Fill them here as shown: -

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: -

#### Database Schematic





A Tesco receipt from Horsham, dated 0845 6779355. The receipt lists various items with their prices and quantities. The items are: CHAMPAGNE (12.49), TZATZIKI (1.00), MINI CHEDDARS (2.00), SOUR CREAM DIP (1.00), CARAMAC (0.60), FLAKEL (0.60), TWIX KINGSIZE (0.80), SENSATIONS (1.99), RED GRAPE LSE (0.895 kg @ £4.47/kg, 2.66), ALRO PEPMINT (0.60), BREAU (1.49), and SENSATIONS (1.99). The subtotal is 27.22. There are also multi-buy savings: DIPS ANY 2 FOR £1.80 (-0.20), EASTER RANGE 3 F £1.20 (-0.60), and CHIPS & DIPS BOGOF (-1.99). The total savings are -2.79. The total to pay is 24.43, and the visa debit sale is also 24.43.

TESCO	
HORSHAM 0845 6779355	
CHAMPAGNE	12.49
TZATZIKI	1.00
MINI CHEDDARS	2.00
SOUR CREAM DIP	1.00
CARAMAC	0.60
FLAKEL	0.60
TWIX KINGSIZE	0.80
SENSATIONS	1.99
RED GRAPE LSE	0.895 kg @ £4.47/ kg
ALRO PEPMINT	0.60
BREAU	1.49
SENSATIONS	1.99
SUB-TOTAL	27.22
MULTI-BUY 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

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.

So let's start to build and maintain our database: -

## a) Stage 1 – Updating Table Information

Here we will start by looking at some information which is clearly wrong in a table and correcting it.

To do this with Node-RED we will need to set up a Form Node and configure it to allow us to enter new data into the database table. We will also make use of the previous lesson – Reading Databases – to check the information and see our actual changes.

So, let's refresh ourselves on the mini-database that we are using for these lessons.

We are using a small database of three tables in a DB2 Library named IUGRED

CUSORDADD	A name and address table
ORDERHEADX	A table of sales order headers
ORDERLINEX	A table of sales order lines

### Single Table

We will use the work we have done in Lesson 7 – Reading the Database – to check the results of the actions in this lesson. So, Open the Flow Tab for Lesson 7 and hit the Inject button. It will probably still have one of the order numbers we have been using, say, S105184. That's OK.

Now, Open the **Databasesx** Tab on our **UI** from the last lesson. We should see something like this: -

Order Head

ODCUNM	ODCUA1	ODCUA2	ODCUA3	ODCUA4	ODPONO	
PETRO-CANADA N...	P.O.BOX 9	ATTENTION: ACCO...	2270 AA VOORBURG	NEDERLAND	2270 AA	
OAORNO	OAORDT	OAOREF	OAYREF			
S105184	20100708	Loek Prinsen	MR A. van der Drift			
OBORNO	OBFACI	OBWHLO	OBITNO	OBITDS	OBTEDS	OBSAPR
S105184	N2C	N10	9SITE	SYSTEM	T4-330-SR2 CW Co...	16686.000000
S105184	N2C	N10	9SITE	S105184-401	Control componen...	2327.000000
S105184	N2C	N10	9SITE	SYSTEM	Flushing	500.000000
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

Now, change the Sales Order Number in the Inject Node on Lesson 7. Use **105224**: -

The data will change to this Sales Order: -

Order Head

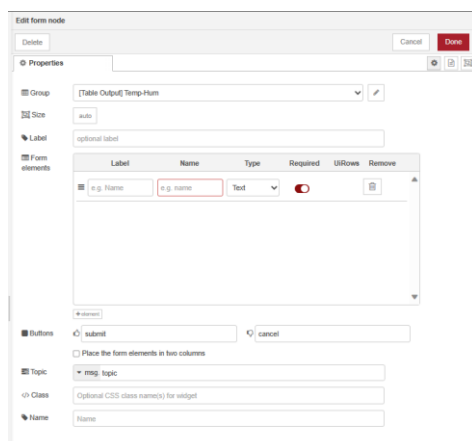
ODCUNM	ODCUA1	ODCUA2	ODCUA3	ODCUA4	ODPONO	
SACCAP INDUSTRI...	112/114 Avenue d...	Blois		FRANCE		
OAORNO	OAORDT	OAOREF	OAYREF			
S105224	20100817	Loek Prinsen				
OBORNO	OBFACI	OBWHLO	OBITNO	OBITDS	OBTEDS	OBSAPR
S105224	N2C	N10	9355776	SYS.CBA415-SR60 ...		5904.000000
S105224	N2C	N10	9355776	SYS.CBA415-SR60 ...		5904.000000
S105224	N2C	N10	9355776	SYS.CBA415-SR60 ...		5904.000000
S105224	N2C	N10	9355776	SYS.CBA415-SR60 ...		5904.000000
S105224	N2C	N10	9355776	SYS.CBA415-SR60 ...		5904.000000
S105224	N2C	N10	9355776	SYS.CBA415-SR60 ...		5904.000000
S105224	N2C	N10	9355777	SYS.CBA525-SR60 ...		5998.000000
S105224	N2C	N10	9355777	SYS.CBA525-SR60 ...		5998.000000
S105224	N2C	N10	9355778	SYS.CBA730-SR60 ...		6888.000000
S105224	N2C	N10	9355778	SYS.CBA730-SR60 ...		6888.000000
S105224	N2C	N10	9355776	SYS.CBA415-SR60 ...		5904.000000
S105224	N2C	N10	9355776	SYS.CBA415-SR60 ...		5904.000000
S105224	N2C	N10	9355777	SYS.CBA525-SR60 ...		5998.000000
S105224	N2C	N10	9355777	SYS.CBA525-SR60 ...		5998.000000
S105224	N2C	N10	99PR	Mounting frame	PRICE ADJUSTMENT	466.000000
S105224	N2C	N10	99VR	FREIGHT/EXPRESS ...		2000.000000

Whilst the data is ostensibly similar, we can see that the Address has not been filled in fully. In particular, the Post Code has been left blank: -

Order Head						
ODCUNM	ODCUA1	ODCUA2	ODCUA3	ODCUA4	ODPONO	
SACCAP INDUSTRI...	112/114 Avenue d...	Blois		FRANCE		

Although we have a Company Name and a Country, it might be difficult to send a driver here. So, let's update the Customer Order Address for this Order and add in the correct Post Code.

Start a new Flow. Drag in a Form Node and Open it: -



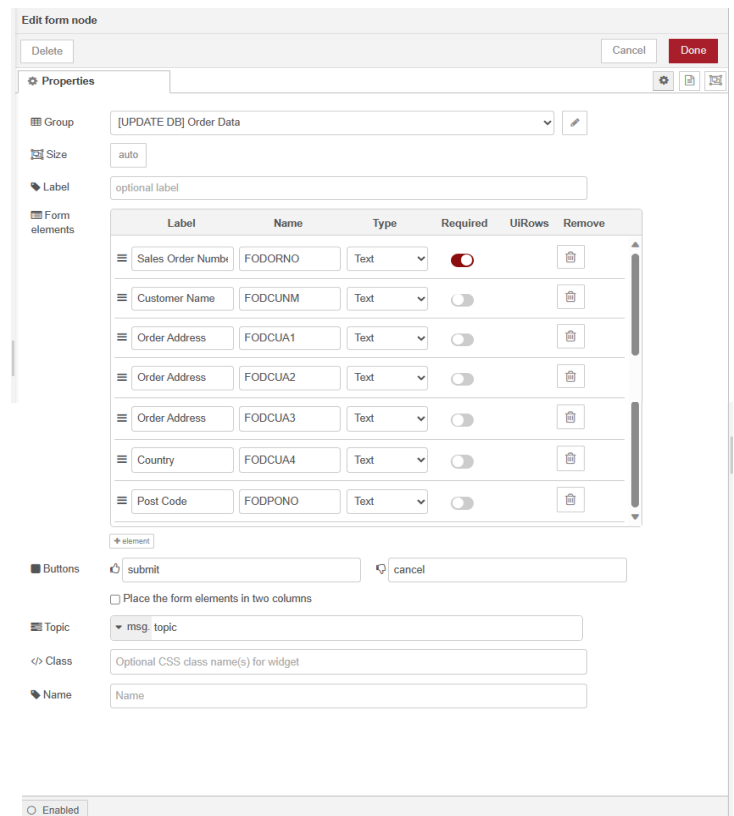
We are going to add in as many elements as we need in order to update the information we are interested in this record/row in the table.

Assign this to TAB 'Table Output' and Name 'Input' on the UI

Add in these element (Use Form Node Name from the table below): -

Name	Table Name	Form Node Name
Sales Order Number	ODORNO	FODORNO
Customer Name	ODCUNM	FODCUNM
Order Address	ODCUA1	FODCUA1
Order Address	ODCUA2	FODCUA2
Order Address	ODCUA3	FODCUA3
Country	ODCUA4	FODCUA4
Post Code	ODPONO	FODPONO

Here is the Table layout and the information we will use to set up our Form for Update.



It should end up looking like this

Notice that only the Sales Order Number is a mandatory (Required) element/field. This means that the form will not progress unless the Sales Order Number has been entered.

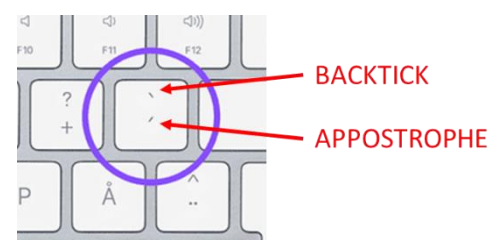
Leave all the other details as default.

Next, drag in a Function Node. We are going to put in a new SQL statement that will put the data that we have just entered in the Form Node directly into the database.

*N.B. In reality, whenever we are entering data into our database, we would look at the data entered and make sure it was reasonable data. Remember, G.I.G.O. – Garbage In Garbage Out. If we put in silly data, or worse, if we corrupt data in the database, we could destroy the whole database.*

*Here we are making just one check – that it is a correct and existing Order Number, In fact, it is SQL that is providing that facility.*

*N.B.2 We will be building an SQL statement in a String with substitution variables.. The string can only work if placed within 'BACKTICKS' It cannot work with 'APOSTROPHES'*





Below is the basic code that will most definitely update the columns we are interested in for this CUSORDADD table. Set up this code in your Function Node: -

```

1  let order = msg.payload.FODORNO
2  let cuname = msg.payload.FODCUNM
3  let addr1 = msg.payload.FODCUA1
4  let addr2 = msg.payload.FODCUA2
5  let addr3 = msg.payload.FODCUA3
6  let addr4 = msg.payload.FODCUA4
7  let pono = msg.payload.FODPONO
8
9
10 let payloadout = `update iugred.cusordadd SET ODCUNM = '${cuname}',
11   ODCUA1 = '${addr1}',
12   ODCUA2 = '${addr2}',
13   ODCUA3 = '${addr3}',
14   ODCUA4 = '${addr4}',
15   ODPONO = '${pono}'
16   WHERE ODORNO = '${order}' `
17
18 msg.payload = payloadout
19
20 return msg;

```

What is the code doing?

At the top of the code, we are setting up a work variable into which we are putting the information from the Form Node.

Next, we are building up an SQL statement that we will pass to the Db2 Node. Here we have used multiple lines as we think this is more readable, but you can just put the whole string in one line if you prefer.

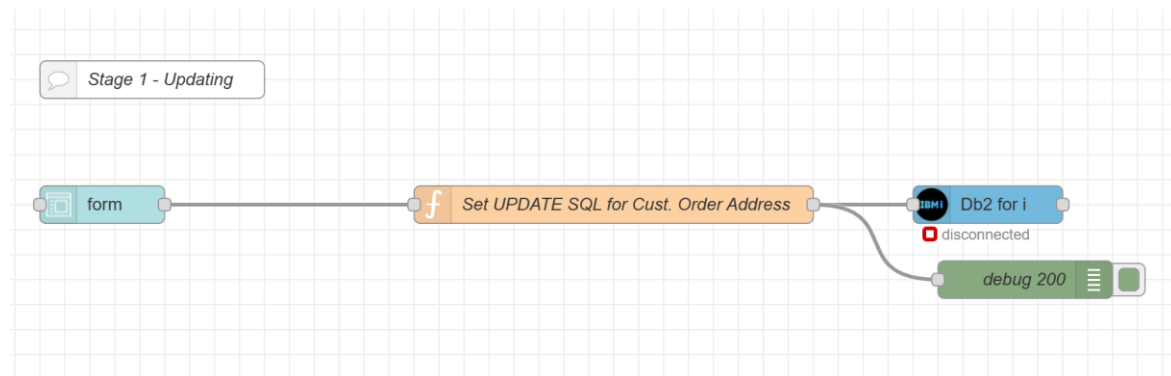
The SQL will literally take what was entered on the Forms Node and write it to the correct Sales Order Address row in the table (S105224).

But wait a minute. If we go ahead with this form and, say, we just enter the value for the Post Code in the Form Node, we run the risk of overwriting all the other columns in the table with Blanks/Nulls.

Worse, as we have no idea which fields the user will populate, we can start to really dig a hole for ourselves. Let's see what it looks like: -

Drag in a Db2 Node. It should default to the properties we have been using, so there will not be any need to alter it. It is just waiting for our instruction, in this case, an SQL statement.

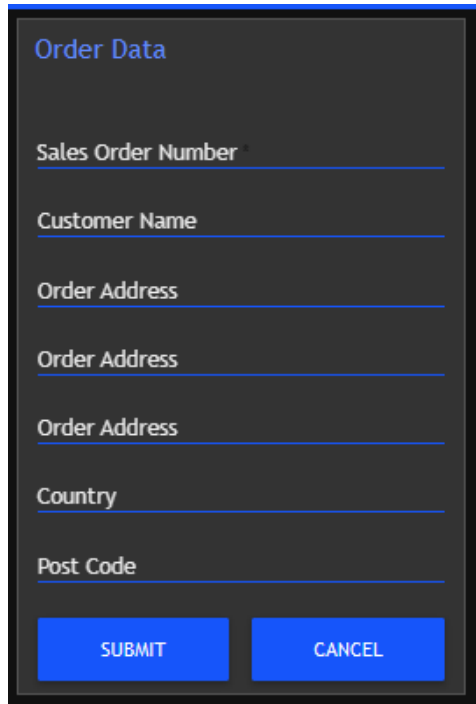
Drag in a Debug Node and set it to show the complete msg. object. Your Flow should look like this: -





Deploy the Flow, and then go to the UI – Table Output Tab.

The form we have created using the Form Node will look a bit like this: -



At this point, this is just an experiment where we can just observe the effect of what we have created. We will repair this one when we move on to how to set this up more professionally.

So, just key: -

**410314** into the Post Code field only.

Then hit **Submit**.

...It should stop you and force you to fill in the 'Sales Order Number' field. Accept this and then key S105224 into the Sales Order Number field. Submit this again.

So, now if we go back to the Databases tab (The Order Output created in Lesson 7) and look at this order (S105224 – it should still be in the Inject Node in Lesson 7), we can see the damage we have done... (Go in to Lesson 7, hit the Inject then return to the UI Databases Tab to see the referred data)

ENTER

Sales Order Number  
S105224

SUBMIT

CANCEL

Order Head

ODCUNM	ODCUA1	ODCUA2	ODCUA3	ODCUA4	ODPONO
					410314

OAORNO	OAORDT	OAOREF	OAYREF
S105224	20100817	Loek Prinsen	

OBORNO	OBFACI	OBWHLO	OBITNO	OBITDS	OBTEDS	OBSAPR
S105224	N2C	N10	9355776	SYS.CBA415-SR60 ...		5904.000000
S105224	N2C	N10	9355776	SYS.CBA415-SR60 ...		5904.000000
S105224	N2C	N10	9355776	SYS.CBA415-SR60 ...		5904.000000
S105224	N2C	N10	9355776	SYS.CBA415-SR60 ...		5904.000000
S105224	N2C	N10	9355776	SYS.CBA415-SR60 ...		5904.000000
S105224	N2C	N10	9355776	SYS.CBA415-SR60 ...		5904.000000
S105224	N2C	N10	9355777	SYS.CBA525-SR60 ...		5998.000000
S105224	N2C	N10	9355777	SYS.CBA525-SR60 ...		5998.000000
S105224	N2C	N10	9355778	SYS.CBA730-SR60 ...		6888.000000
S105224	N2C	N10	9355778	SYS.CBA730-SR60 ...		6888.000000
S105224	N2C	N10	9355776	SYS.CBA415-SR60 ...		5904.000000
S105224	N2C	N10	9355776	SYS.CBA415-SR60 ...		5904.000000
S105224	N2C	N10	9355777	SYS.CBA525-SR60 ...		5998.000000
S105224	N2C	N10	9355777	SYS.CBA525-SR60 ...		5998.000000
S105224	N2C	N10	99PR	Mounting frame	PRICE ADJUSTMENT	466.000000
S105224	N2C	N10	99VR	FREIGHT/EXPRESS ...		2000.000000

Well, the update certainly worked! But in updating the Post Code, we have inadvertently overwritten a lot of other information... with blank!

So, how do we get around this? With some code, of course.

Let's go back to our Function Node: -

```
1  let order = msg.payload.FODORNO
2  let cuname = msg.payload.FODCUNM
3  let addr1 = msg.payload.FODCUA1
4  let addr2 = msg.payload.FODCUA2
5  let addr3 = msg.payload.FODCUA3
6  let addr4 = msg.payload.FODCUA4
7  let pono = msg.payload.FODPONO
8
9
10 let payloadout = `update iugred.cusordadd SET ODCUNM = '${cuname}',
11   ODCUA1 = '${addr1}',
12   ODCUA2 = '${addr2}',
13   ODCUA3 = '${addr3}',
14   ODCUA4 = '${addr4}',
15   ODPONO = '${pono}'
16   WHERE ODORNO = '${order}' `
17
18 msg.payload = payloadout
19
20 return msg;
```

It has done everything we asked of it... but we didn't ask correctly.

As the information comes in, we need to see if there is actually an entry in the field and if not, we need to preserve what is already there. There are many ways of achieving this. We will explore just one.

First, let's repair the damage we have done by entering correctly the information. In the form on the UI, enter the following: -

Form Name	Entry
Sales Order Number	S105224
Customer Name	SACCAP INDUSTRIAL VALVES MANUFACT
Order Address	112/114 Avenue de Vendôme
Order Address	Blois
Order Address	
Country	FRANCE
Post Code	410314

Take a look again at the order using the Databasex tab, You will see that all the order information is now restored. (Remember to refresh, Inject).

We are going to get SQL to do the work here.

If a user wanted to just change the post code, they would just want to put in the Sales Order Number (mandatory) and then just put in the Post Code. It would be up to us to detect this and make sure that the SQL statement did not update the other columns or we would end up being in a worse state than when we started. Just like in our test above.

Additionally, it would be unfair - and particularly risky bad practice - to force the user to enter ALL of the information for all OF THE COLUMNS we are interested in, each time just one column needed updating.

So, let's see what SQL can do for us here.

In the previous method, we asked SQL to 'Set' each column, without any checks: -

We need to enhance the statement to check the value coming in. For this we will introduce a 'CASE' element into the SET part of the statement: -

This is what is happening. Let's take the Customer Number field – ODCUNM

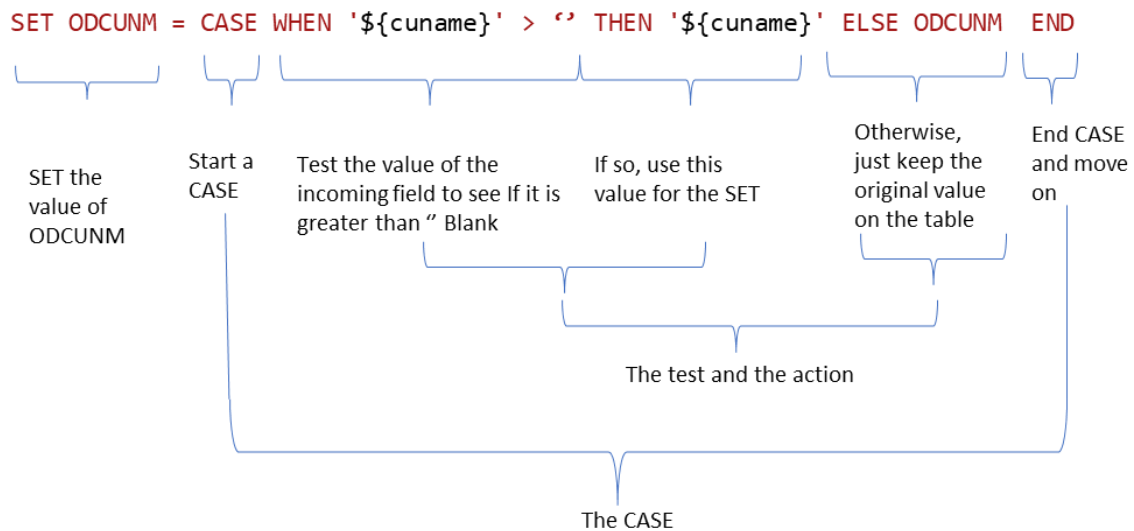
Now, instead of just setting it to the incoming value - **cuname**

```
SET ODCUNM = '${cuname}'
```

We want SQL to evaluate what is coming in and ONLY update it if the value is greater than '' (Blank)

```
SET ODCUNM = CASE WHEN '${cuname}' > '' THEN '${cuname}' ELSE ODCUNM END
```

Here is the make-up of this part of the statement: -



So, if we do this for every field then it will not matter if the user keys in one, two or all the fields, the row in the table will be updated accordingly.

So let's go ahead and do this. Change the build of the SQL statement to include the new CASE values and give it a test. The Form Node does not need to change. It should look like this: -

```

1  // Bring in the payload and place into variables for this process
2
3  let order = msg.payload.FODORNO
4  let cuname = msg.payload.FODCUNM
5  let addr1 = msg.payload.FODCUA1
6  let addr2 = msg.payload.FODCUA2
7  let addr3 = msg.payload.FODCUA3
8  let addr4 = msg.payload.FODCUA4
9  let pono = msg.payload.FODPONO
10
11
12 // Build the SQL statement, only SET the column if the value from the Forms Node is NOT null/blank
13
14 let payloadout = `update iugred.cusordadd set
15   ODCUNM = CASE WHEN '${cuname}' > '' THEN '${cuname}' ELSE ODCUNM END,
16   ODCUA1 = CASE WHEN '${addr1}' > '' THEN '${addr1}' ELSE ODCUA1 END,
17   ODCUA2 = CASE WHEN '${addr2}' > '' THEN '${addr2}' ELSE ODCUA2 END,
18   ODCUA3 = CASE WHEN '${addr3}' > '' THEN '${addr3}' ELSE ODCUA3 END,
19   ODCUA4 = CASE WHEN '${addr4}' > '' THEN '${addr4}' ELSE ODCUA4 END,
20   ODPONO = CASE WHEN '${pono}' > '' THEN '${pono}' ELSE ODPONO END
21   WHERE ODORNO = '${order}' `
22
23
24
25 // Store the SQL statement in the msg.payload , when '${addr1}' > '' then set ODADR1 = '${addr1}';
26
27 msg.payload = payloadout
28
29 return msg;

```

Pay careful attention to where items like commas are used. Remember, because you are building an SQL Statement in a String, the Function Node cannot do even basic checks on the SQL statement itself.

*N.B. If the text you are entering does NOT follow the colour scheme as in the example above, it may be because you are not using the BACKTICKS correctly!!*

Remember to **Deploy** it.

Now key in the Sales Order Number S105224 and, say, the Post Code. When you refresh the data and look again, the Post Code will have changed but all the other information making up the record is still intact.

Try it now.

#### Footnote to consider

Many of you will have noticed a flaw even in the enhanced SQL Update statement we have used.

Yes, if a user wanted to blank out a field, say, changing an address where the new Address only had 2 or three lines instead of the previous 4 line address, changing the address line to blank ("") would have no effect.

Clearly this can be catered for as well. Do a little research with Mr Google and see if you can enhance the SQL Statement to accommodate this.

b) Stage 2 - Inserting Rows to a Table.

Where we want to, say, add a new supplier or, add a new order delivery address to a master table, or simply when we are adding new transactions to a transaction table, we want to Insert rows to add to the rows already there.

So, to save creating a new form and a new Flow from scratch, let's copy the current Update Flow to a point further down this Flow palette.

Once copied, change the Form Node to be on the same Tab as before but in a new name group - INSERT.

Add a new Element :-

Phone	ODPHNO	FODPHNO
-------	--------	---------

We can dispense with the Inject Node as we have a perfectly good form to use.

Now, we are going to work on the Function Node and change this to create an INSERT SQL statement rather than an UPDATE SQL Statement.

Open the Function Node. The SQL here is far more simple. Here is the normal SQL Syntax: -

```
INSERT INTO iugred.cusordadd (ODORNO, ODCUSN, ODADR1, ODADR2, ODADR3, ODADR4, ODPONO, ODPHNO)
VALUES (order, cuname, addr1, addr2, addr3, addr4, pono, phone)
```

Note: We now have to put in the new Order Number as we are introducing a new row, and we are making sure that the final column (Phone, which we haven't use as yet) is also being populated.

However, as we are going to be constructing a String with the substitution values from the Form Node, we will see it like this: -

```
let payloadout = `insert into iugred_nn.cusordadd (ODORNO, ODCUSN, ODADR1, ODADR2, ODADR3, ODADR4, ODPONO, ODPHNO)
VALUES ('${order}', '${cuname}', '${addr1}', '${addr2}', '${addr3}', '${addr4}', '${pono}', '${phone}')`
```

So, replace the SQL statement in the Function Node with this code and we'll give it a go.

Remember to Deploy, then go to the UI Table Output. In the INSERT form, fill in a name and address for a new row – Sales Order **S199123**.

Check if it is there by using the Databasex enquiry table same Sales Order Number as the one you just entered here – S199123.

You will only get the Order Address information as we have not added records to the Order Header and Order Detail file yet.

So now, as well as reaching into a major IBM DB2 Database, you have added information to the Database and modified data in the existing database.

What we learned about reading the database earlier has been very helpful here so that we can see if our additions and modifications have been successful.

Have a go at adding a few new rows using the INSERT DB form (Use different Sales Order Numbers for each new one), change a few rows – even the ones you just entered – using the UPDATE DB form, and check they are OK using the enquiry form and tables on the Databasesex tab.

You might find it easier now to modify what we have done in Lesson 7 and actually add a new Form where you can key in the Sales Order Number on the same page (TAB) as the Sales Order Tables. This would negate you needing to keep changing the Inject Node contents to get a new order to show. Try it, it does help.

### Another footnote for you to consider

There are very many things that can be wrong with Data being put in to a Database, and these all usually need some attention from the person developing the entry/amend programs and facilities.

One here is very important, but you may have only seen a Node-RED error in the Debug panel. That issue is duplication.

If we Insert Sales Order 199124 into our database with an new address and contact details, that's fine, and we have a unique address for that order. But we don't want another one on the table as that would then at least confuse everyone, and potentially cause a great deal of downstream problems with our wider Database.

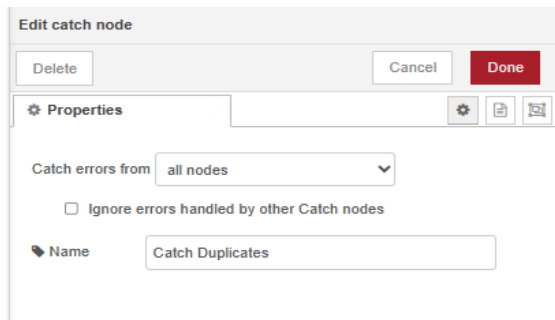
We have prevented this by making the ODORNO (Sales Order Number) column 'Unique'. This means that the DB2 Database Management System in the background just will not let there be two rows with the same Sales Order Number on them.

So how do you tell without looking at the Debug section in the Flow? All you see is that the entry never makes it on to the database because whenever you look at the Database (Using the enquiry form on the Databasesex tab), you see the old record.

Node-red can help here. We need to catch the error so... we'll use a Catch Node.

In your Flow, drag in a Catch Node and open it: -





**Edit catch node**

Delete Cancel Done

Properties

Catch errors from **all nodes**

☐ Ignore errors handled by other Catch nodes

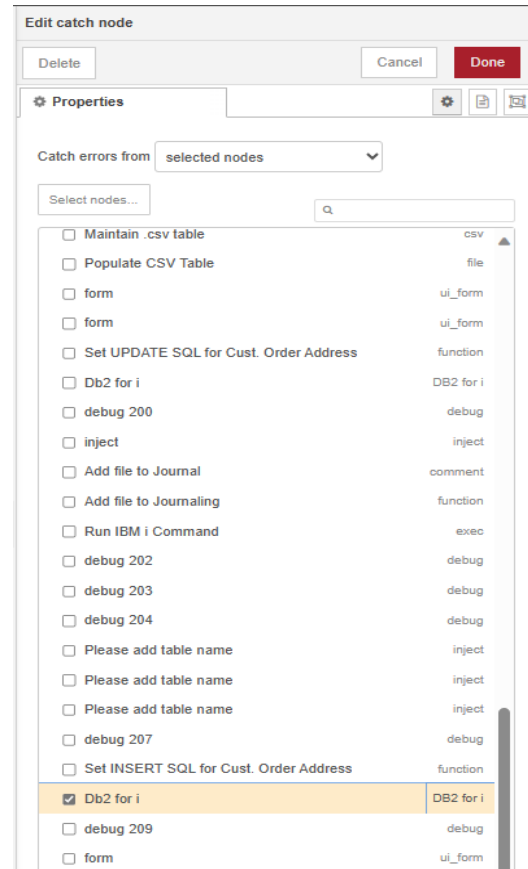
Name **Catch Duplicates**

It is set to catch errors from all nodes to start with, but that might become tedious. Change this to 'Selected nodes' and then select our DB2 node from the drop-down list.

You will see that there are many Db2 for i Nodes listed. This is because we have been lazy and not given each one a name... However, it is the one just below the function box that we have named 'Set INSERT SQL for Cust. Order Addresses'.

Select that one. It will be near the bottom of the list. Give it a name.

Now, drag in a Notification Node (that actually may have shown up when you searched for the Catch Node). Open it: -



**Edit catch node**

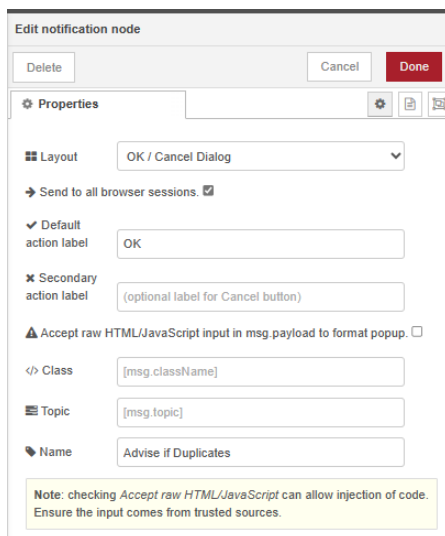
Delete Cancel Done

Properties

Catch errors from **selected nodes**

Select nodes...

<input type="checkbox"/>	Maintain .csv table	csv
<input type="checkbox"/>	Populate CSV Table	file
<input type="checkbox"/>	form	ui_form
<input type="checkbox"/>	form	ui_form
<input type="checkbox"/>	Set UPDATE SQL for Cust. Order Address	function
<input type="checkbox"/>	Db2 for i	DB2 for i
<input type="checkbox"/>	debug 200	debug
<input type="checkbox"/>	inject	inject
<input type="checkbox"/>	Add file to Journal	comment
<input type="checkbox"/>	Add file to Journaling	function
<input type="checkbox"/>	Run IBM i Command	exec
<input type="checkbox"/>	debug 202	debug
<input type="checkbox"/>	debug 203	debug
<input type="checkbox"/>	debug 204	debug
<input type="checkbox"/>	Please add table name	inject
<input type="checkbox"/>	Please add table name	inject
<input type="checkbox"/>	Please add table name	inject
<input type="checkbox"/>	debug 207	debug
<input type="checkbox"/>	Set INSERT SQL for Cust. Order Address	function
<input checked="" type="checkbox"/>	Db2 for i	DB2 for i
<input type="checkbox"/>	debug 209	debug
<input type="checkbox"/>	form	ui_form



**Edit notification node**

Delete Cancel Done

Properties

Layout **OK / Cancel Dialog**

Send to all browser sessions. ☒

Default action label **OK**

Secondary action label (optional label for Cancel button)

☐ Accept raw HTML/JavaScript input in msg.payload to format popup.

Class **[msg.className]**

Topic **[msg.topic]**

Name **Advise if Duplicates**

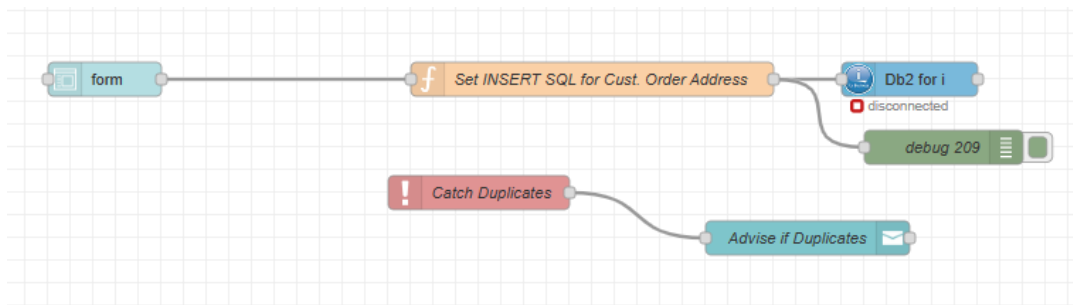
Note: checking Accept raw HTML/JavaScript can allow injection of code. Ensure the input comes from trusted sources.

You'll see that there a selection of Layouts. Let's choose the OK/ Cancel Dialogue option.

This will display the error and offer up an 'OK' button to click to clear the error notification. This is good because it means we have to recognise the issue before we can continue.

Link the Notification Node to the Catch Node... but the Catch Node does NOT need to be integrated into the Flow!

Your Flow should look like this: -



Now, try and put in a duplicate Sales Order. You should get a notification looking like this: -

Add a Row

Sales Order Number

Customer Name

Order Address

Order Address

Order Address

Country

Post Code

SQLSTATE=23505 SQLCODE=-803 Duplicate key value specified.

OK

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](https://github.com/JoshRyanEOG/i-UG_Education_Node-Red), 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](https://github.com/JoshRyanEOG/i-UG_Education_Node-Red)