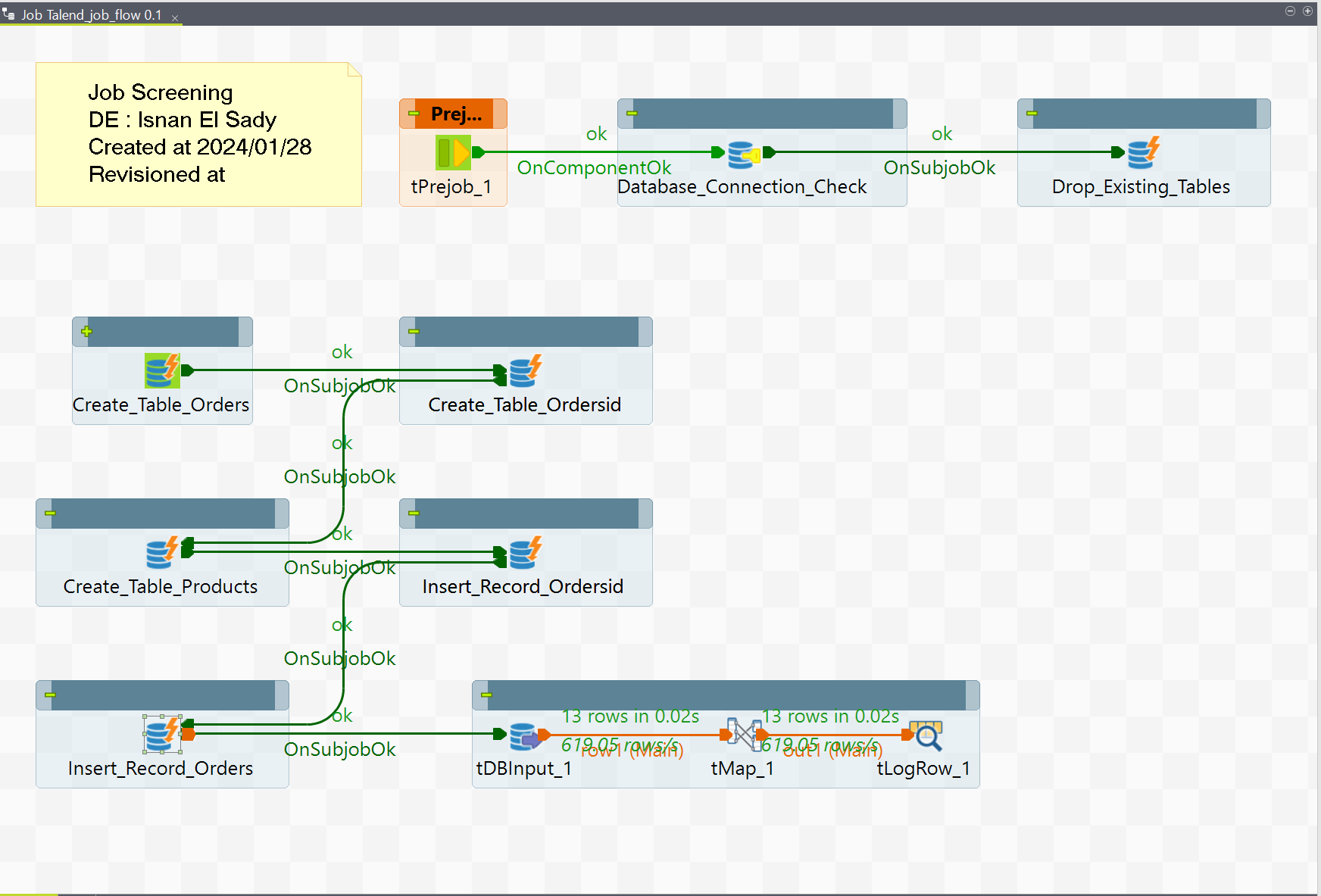
**Technical Document**

**Project in Charge : Isnan El Sady**

Tools / Requirements include :

1. Talend Open Studio Data Integration version 8.0
2. PostgreSQL Database version 14.0
3. pgAdmin version 4.0

**Data Flow Overview**

****

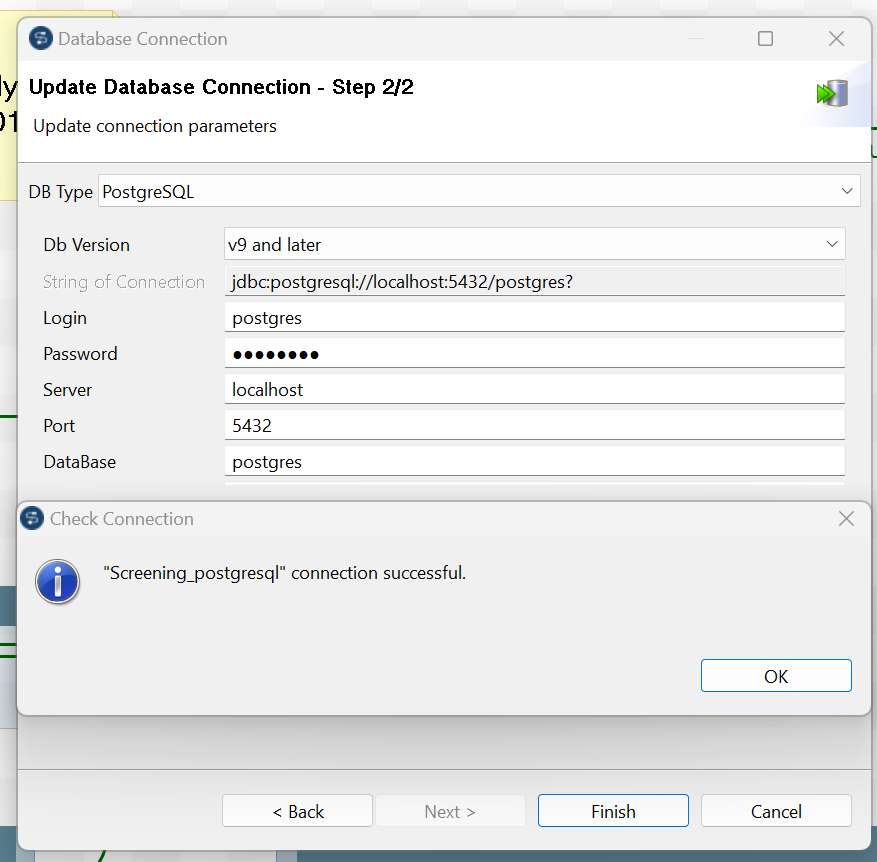
There are essentially 4 Stages of the data flow, that is include

1. Dropping tables making sure none overlaps with any data we’ll be working with.
2. Creating the tables.
3. Inserting the records on to tables.
4. Using Input component to query the data we need and output it with studio built-in log in table format.

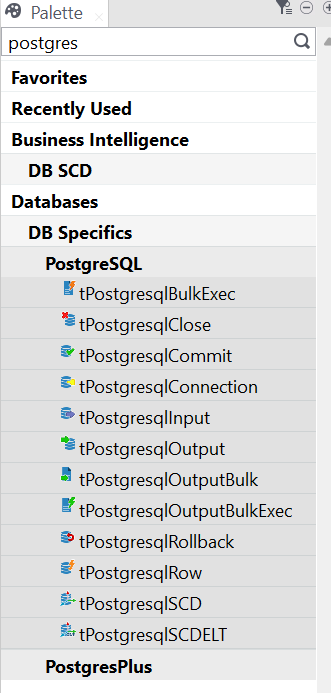
**Step by Step Job Designing**

There following will specifically guide trough the process of designing the flow from scratch to finish.

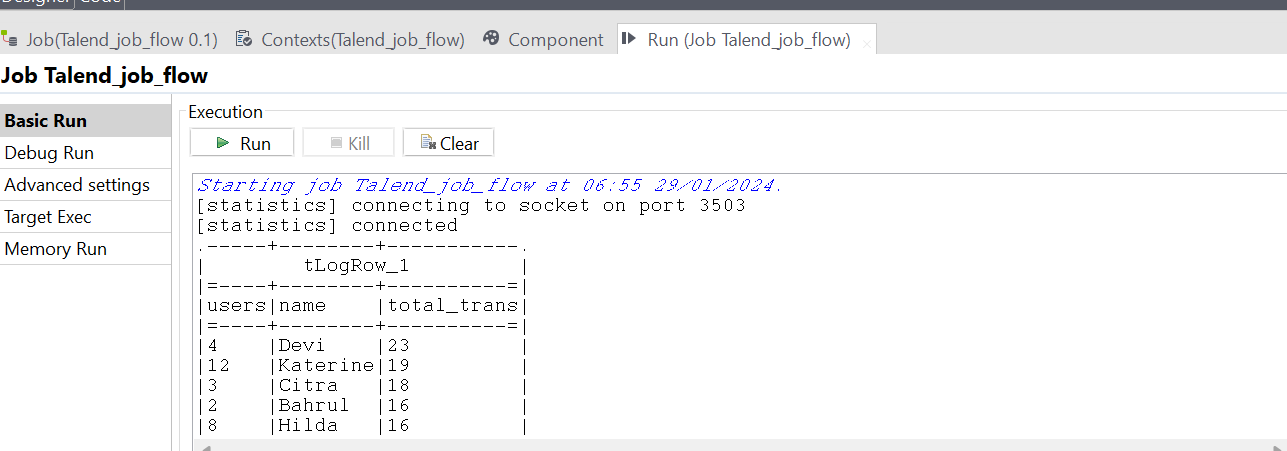
1. Establishing a database connection, we’ll want to establish a connection with the destinated database, in studio, just have to navigate trough the left hand side part of the window, search **Metadata**, drop it down, you’ll see **DB Connection***,* right click and *create new connection*. Define the connection names, and other parameters. Once all that filled, click on *Test Connection*, a prompt should come out if there’s nothing wrong with the connection.



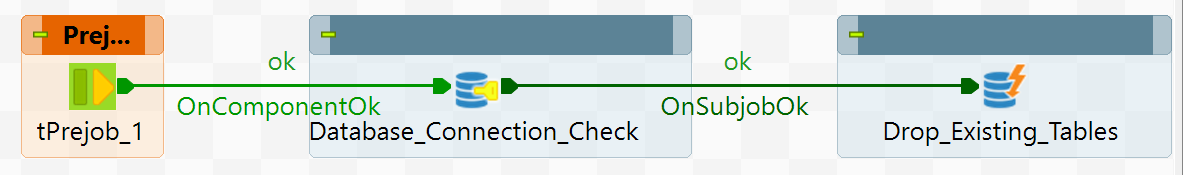
1. Second is we want to make a *canvas*, essentially a template in which well be designing the flow at or Job template. To do this, navigate through the same left hand side windows, search for **Job Design**, right-click and create job, or if you want to be more organized you can make a folder on top a job. Define the names of job, others are optional, should there be no error a blank *canvas* appear.
2. The blank *canvas* is the place where we’ll put the component for our job design. Component can either be put on to the canvas design by, first, dragging the from the **palette** , located In the right hand side of windows or second, simply just straight forward type the component you need in directly the blank canvas.



1. Should you have ready job to be executed, In the designer, you’ll find another windows at the bottom, search for tab labeled – **Run Job** , or simply press F5. The log will inform you should there are any error, if not the screen below represents a successfully run job.

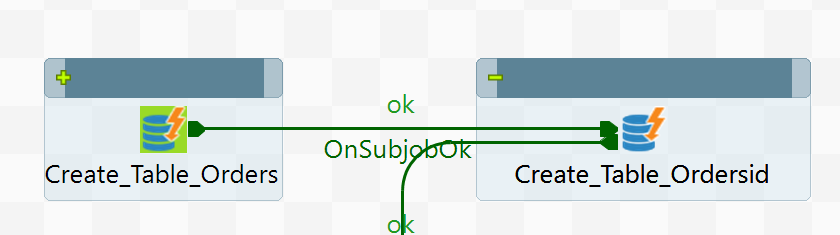


**Flow Explanation**



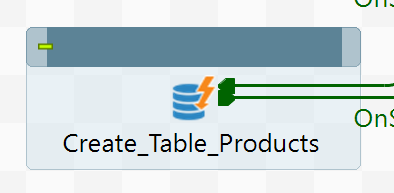
Prejob essentially preparatory flow executed before anything else. There are 2 components here. First, *Database\_Connection\_Checking* is to ensure the connection established is successful in studio, second is procedural list of queries to be executed by the component. The list of queries is to drop any table making sure no data overlaps with data we’ll be working on. Below is the query.

|  |
| --- |
| DROP TABLE IF EXISTS public."ORDERS";  DROP TABLE IF EXISTS public."ORDERSID";  DROP TABLE IF EXISTS public."PRODUCTSDETAILS"; |

****

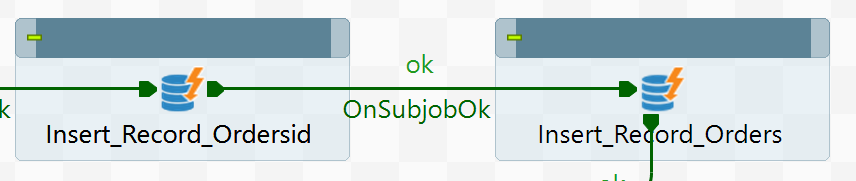
First stage of the main flow, is to create the tables, 2 of which we’ll be working on. There are Orders and OrdersId. The former holds data related to detailed transaction of certain customer, like the product, how many the bought, and the total of their purchase, meanwhile the latter holds detailed info about the customer, their name, unique store id gender, and address. DDL queries for these tables are as below.

|  |
| --- |
| CREATE TABLE IF NOT EXISTS public."ORDERS"  (  id integer NOT NULL,  createdat date NOT NULL,  users integer NOT NULL,  productname character(50) COLLATE pg\_catalog."default",  price double precision,  quantity double precision,  totalprice double precision,  CONSTRAINT "ORDERS\_pkey" PRIMARY KEY (id)  );  CREATE TABLE IF NOT EXISTS public."ORDERSID"  (  id integer NOT NULL,  name character varying(50) COLLATE pg\_catalog."default" NOT NULL,  gender character varying(20) COLLATE pg\_catalog."default" NOT NULL,  address character varying(50) COLLATE pg\_catalog."default",  CONSTRAINT "ORDERSID\_pkey" PRIMARY KEY (id)  ); |

****

Next step is to create supporting table, ProductsDetail. This table Holds information for the name of product being sold and the listed price tag. This is essentially an extra step and a necessary one, detailed about this, relate to the next step of the flow. DDL query and record ingestion of the table is as below.

|  |
| --- |
| CREATE TABLE IF NOT EXISTS public."PRODUCTSDETAILS" (  NAME varchar(50) ,  price float8);  INSERT INTO public."PRODUCTSDETAILS"  values ('The Ordinary The Bright Set' , 8.65);  INSERT INTO public."PRODUCTSDETAILS"  values ('Goodnight Wrinkle Night Cream' , 8.99);  INSERT INTO public."PRODUCTSDETAILS"  values ('Dorion Renaud Charcoal Detox Mask' , 12.49);  INSERT INTO public."PRODUCTSDETAILS"  values ('Mask: Korean Face Mask' , 8.99);  INSERT INTO public."PRODUCTSDETAILS"  values ('Facial Fuel Energizing Face Wash' , 10.99);  INSERT INTO public."PRODUCTSDETAILS"  values ('Tower 28 SOS Skincare Set' , 14.99);  INSERT INTO public."PRODUCTSDETAILS"  values ('Le Domaine Skincare The Cream' , 11.99);  INSERT INTO public."PRODUCTSDETAILS"  values ('Chanel Boy de Chanel Lip Balm' , 9.99);  INSERT INTO public."PRODUCTSDETAILS"  values ('Summer Fridays Jet Lag Mask Moistur' , 10.99);  INSERT INTO public."PRODUCTSDETAILS"  values ('Grown Alchemist Balancing Toner' , 7.99);    INSERT INTO public."PRODUCTSDETAILS"  values ('Skyn Iceland Hydro Cool Firming Eye Gels' , 8.75);    INSERT INTO public."PRODUCTSDETAILS"  values ('Invisible Daily SPF Broad Spectrum' , 10.49);  INSERT INTO public."PRODUCTSDETAILS"  values ('Humanrace Lotus Enzyme Exfoliator' , 8.99);    INSERT INTO public."PRODUCTSDETAILS"  values ('Paula Complex Cleansing Balm' , 13.99);    INSERT INTO public."PRODUCTSDETAILS"  values ('Sturm Darker Skin Tones' , 7.99); |



Next inserting records on both to OrdersId and Orders tables. Record ingestion of the former table is similar process to above. Define the name of the customer, their unique id , gender and address. The query is as shown below.

|  |
| --- |
| INSERT INTO public."ORDERSID"  values (1  ,'Abdul' , 'Pria' , 'Surabaya');  INSERT INTO public."ORDERSID"  values ((SELECT MAX("id") + 1 FROM public."ORDERSID")  ,'Bahrul' , 'Pria' , 'Sidoarjo');  INSERT INTO public."ORDERSID"  values ((SELECT MAX("id") + 1 FROM public."ORDERSID")  ,'Citra' , 'Wanita' , 'Yogyakarta');  INSERT INTO public."ORDERSID"  values ((SELECT MAX("id") + 1 FROM public."ORDERSID")  ,'Devi' , 'Wanita' , 'Bandung');  INSERT INTO public."ORDERSID"  values ((SELECT MAX("id") + 1 FROM public."ORDERSID")  ,'Ersyad' , 'Pria' , 'Malang');    INSERT INTO public."ORDERSID"  values ((SELECT MAX("id") + 1 FROM public."ORDERSID")  ,'Fajri' , 'Pria' , 'Jakarta');  INSERT INTO public."ORDERSID"  values ((SELECT MAX("id") + 1 FROM public."ORDERSID")  ,'Ghani' , 'Pria' , 'Jakarta');  INSERT INTO public."ORDERSID"  values ((SELECT MAX("id") + 1 FROM public."ORDERSID")  ,'Hilda' , 'Wanita' , 'Solo');    INSERT INTO public."ORDERSID"  values ((SELECT MAX("id") + 1 FROM public."ORDERSID")  ,'Indah' , 'Wanita' , 'Malang');    INSERT INTO public."ORDERSID"  values ((SELECT MAX("id") + 1 FROM public."ORDERSID")  ,'Jehan' , 'Wanita' , 'Palembang');    INSERT INTO public."ORDERSID"  values ((SELECT MAX("id") + 1 FROM public."ORDERSID")  ,'Lulu' , 'Wanita' , 'Bogor');    INSERT INTO public."ORDERSID"  values ((SELECT MAX("id") + 1 FROM public."ORDERSID")  ,'Katerine' , 'Wanita' , 'Bekasi');  INSERT INTO public."ORDERSID"  values ((SELECT MAX("id") + 1 FROM public."ORDERSID")  ,'Mustafa' , 'Pria' , 'Pasuruan'); |

Record ingestion for the latter table is different. Here, there are 4 procedural queries / steps to be done, so that record ingestion of the table can be fully completed. All of 4 procedural use the looping function. We’ll be divide these 4 into Minor 1, Minor 2 , Minor 3 and Minor 4 step. Minor 1 essentially the first one. It is to ingest as many records to the table as how many we defined the number in the query. Data ingested includes, the date record been ingested to table here function current\_date is used. Then, the random id number of customer generated using randomly generated number function, however, we’ll limit to number only registered in the ordersid table. Lastly, it’s the number of item purchased also using random generated number function. Query is as below.

|  |
| --- |
| DO $$  DECLARE v\_amount INT;  BEGIN  v\_amount := 1;  WHILE v\_amount < 201 LOOP  INSERT INTO public."ORDERS"  values (v\_amount,  (CURRENT\_DATE::date) , (SELECT floor(random() \* 13 + 1)::int) , NULL, NULL,  (SELECT floor(random() \* 20 + 1)::int) , NULL);    v\_amount := v\_amount + 1;  end loop;  end$$; |

Since Minor 1 took care of records amount for the table, date-ingested , id and purchase number. Minor 2 is to generate randomly name of product based product name registered on the supporting table, ProducsDetail using the usual select query from destinated table in addition to random() function following it. Minor 2 will iterate exactly over a number or records / rows that has been ingested to the Orders table by Minor 1 ensuring that none records for the columns of product name will be NULL. Query is as shown below.

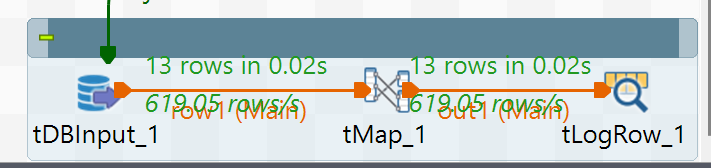
|  |
| --- |
| DO $$  DECLARE v\_product varchar(50);  DECLARE v\_amount INT;  DECLARE f record;  DECLARE v\_cursor1 CURSOR FOR SELECT DISTINCT "id" from public."ORDERS";  BEGIN  FOR f in v\_cursor1 loop  SELECT f."id" into v\_amount;  SELECT "name" FROM public."PRODUCTSDETAILS" ORDER BY RANDOM() LIMIT 1 into v\_product;  UPDATE public."ORDERS"  SET productname = v\_product  WHERE "id" = v\_amount;  end loop;  end$$; |

Minor 2 responsible for ingesting name of product column from table ProductDetail, then Minor 3 is responsible for adding the price tag for that specific product. Minor 3 also uses lopping function to iterate exactly over a number of product registered in ProductDetail table, following it by Update query by referencing the price from that said table. Query is as shown below.

|  |
| --- |
| DO $$  DECLARE v\_product varchar(50);  DECLARE f record;  DECLARE v\_cursor2 CURSOR FOR SELECT DISTINCT productname from public."ORDERS";  BEGIN  FOR f in v\_cursor2 loop  SELECT f."productname" into v\_product;  UPDATE public."ORDERS"  SET price = (SELECT prs."price" FROM public."PRODUCTSDETAILS" prs WHERE prs."name" = v\_product)  WHERE productname = v\_product;  end loop;  end$$; |

Wrapping up the procedural query for Orders table, lastly it is Minor 4. It is responsible to fill the total charge / amount of money to be payed by customer for a given product and price tag. Similar to 3 steps above, it also use looping function. It iterate exactly over a number of records ingested from Minor 1 above ensuring that the total price is exactly how much customer has spend for a given product at a certain amount. Query is as below.

|  |
| --- |
| DO $$  DECLARE v\_price NUMERIC;  DECLARE v\_quantity NUMERIC;  DECLARE v\_total NUMERIC;  DECLARE f record;  DECLARE v\_cursor2 CURSOR FOR SELECT DISTINCT price , quantity FROM public."ORDERS";  BEGIN  FOR f in v\_cursor2 loop  SELECT f.price into v\_price;  SELECT f.quantity into v\_quantity;  SELECT v\_price \* v\_quantity into v\_total;  UPDATE public."ORDERS"  SET totalprice = v\_total  WHERE price = v\_price and quantity = v\_quantity;  end loop;  end$$; |



The Orders and OrdersId are completed, we can now query them to display which customers have purchased from the store multiple times. Here as the name suggest, we use input component to fill in the query that we want. Tmap is essentially to transfer the rows / records from the input component to output. Here we used logrow component, a built-in log component which can also read data into multiple format, including table database. Queryand resulting output are shown below.

|  |
| --- |
| select A.users , B."name" , count(A.users) total\_trans  FROM public."ORDERS" A  LEFT JOIN public."ORDERSID" B ON A.users = B.id  GROUP BY A.users , B."name"  ORDER BY count(A.users) DESC; |

