**Report**

DBMS-2

End term project

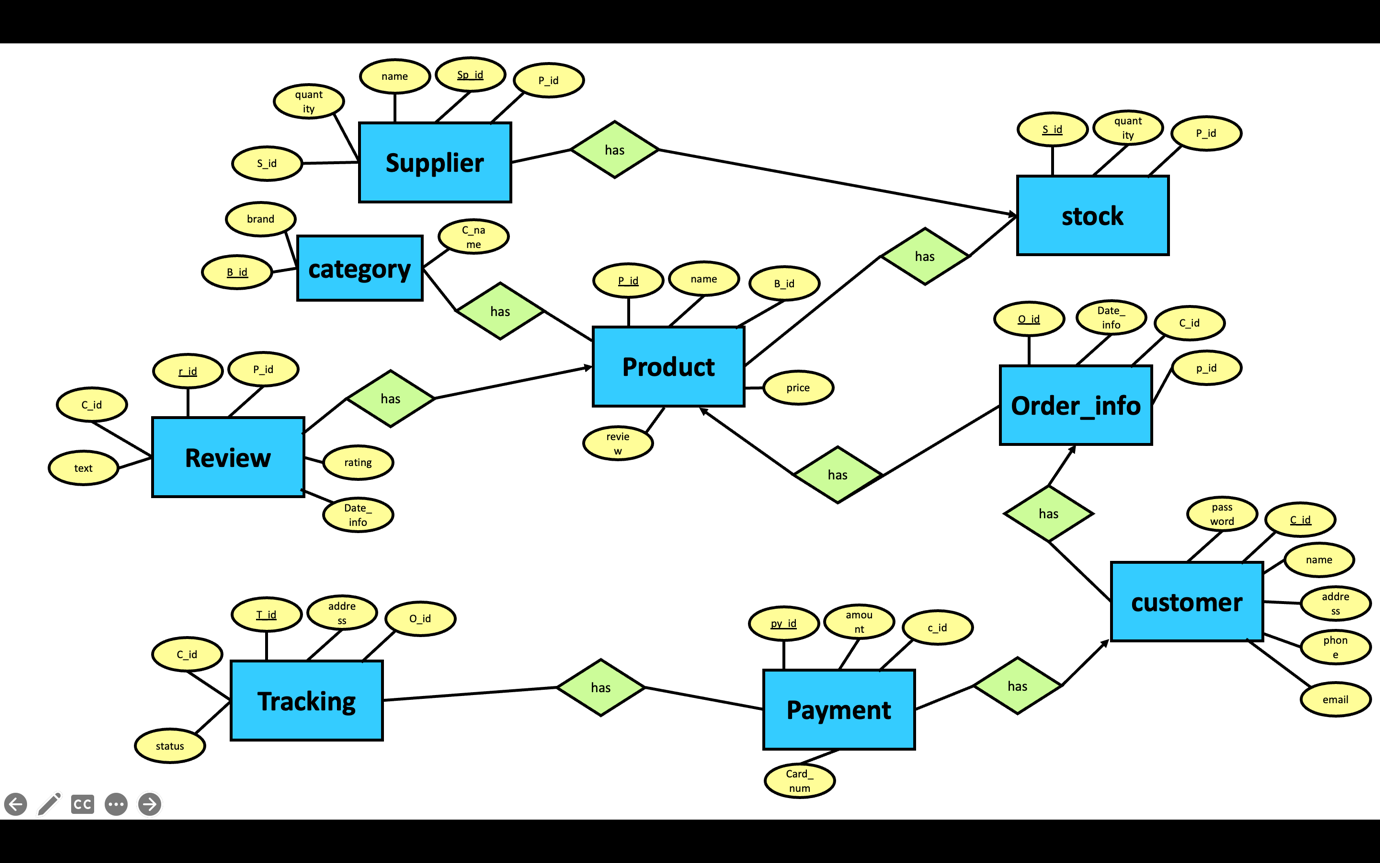
04-N

* Yermakhan Daniar 210103468
* Karassayev Dias 210103363
* Abdullov Adilet 210103435
* Mergenbay Alibek 210103330

**Introduction to the system**

This is a report for dbms-2 end term project which is about database for e-commerce. Our project is based on online grocery store. It contains 9 entities which follow the BCHF rules and have one-to-one and one-to-many structures. The database gives opportunity to store information about products and customers. Customers can track the status of delivery. Information about order and payment area also stored in database. Customer also can add reviews with rating to products. Also, it checks for availability of product by tracking the number of products left in stock and refresh the stock when delivery arrives.

**ER diagram**



In our project we have 9 entities with attributes designed for each table, each table has its own primary key (underlined text in attributes). Tables are connected with one-to-one and many-to-one connections. All tables were created in oracle apex and filled with mock data using mockaroo. Also, some tables have foreign key relation with other tables. For example: table product has a p\_id as primary key and b\_id as foreign key referencing the category table’s primary key b\_id.

**Normal Forms**

Table supplier follow 1nf, because it has unique column name and atomic values in each record. Follows 2nf, because all non-key attributes depend on full primary key s\_id. Follows 3nf, because there is no transitive dependency all non-key attributes depend only on primary key. It is in BCNF because sp\_id is super key.{p\_id, name, quantity, c\_id -> sp\_id}

Table stock also follow 1nf because it has unique column name and atomic values in each record. For 2nf and 3nf there no partial dependency and transitive dependency, because quantity and s\_id depend only on full primary key which is s\_id, also it is in BCNF because s\_id is super key. {p\_id, quantity, -> s\_id}

Table product also follow all normal forms, each column has unique names and there are only atomic values in table rows. There no transitive or partial dependency and p\_id is super key. {b\_id, name, price, review -> p\_id}

Table order\_info has unique column names and atomic values for 1nf, no partial dependency for 2nf and no transitive dependency for 3nf and super key is o\_id for BCNF. {p\_id, date\_info, s\_id -> 0\_id}

Table category is created for removing partial dependency from product table, we can access brand name, category name by b\_id from category and product. It also follows 1nf, 2nf, 3nf, and BCNF b\_id is super key. {brand, c\_name -> b\_id}

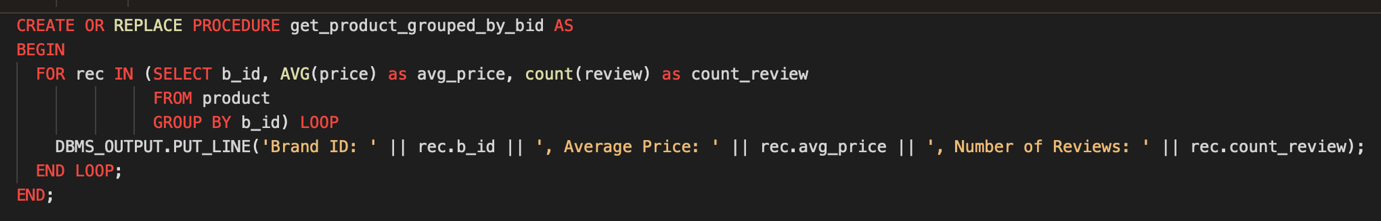
Table customer follows all rules of normal forms, because all non-key attributes depend fully on key attribute which is c\_id. C\_id is also a super key. {name, password, email, phone, address -> c\_id}

Table payment has only unique column names and only atomic values for each tuple. No partial or transitive dependency because all non-key attributes depend only of full key which is super key as well py\_id. {c\_id, card\_num, amount, s\_id -> py\_id}

Table tracking follow the 1nf form because it has uniques column names and atomic values in each tuple. {c\_id, status, address, o\_id -> t\_id}

**Coding part**

**- Procedure which does group**

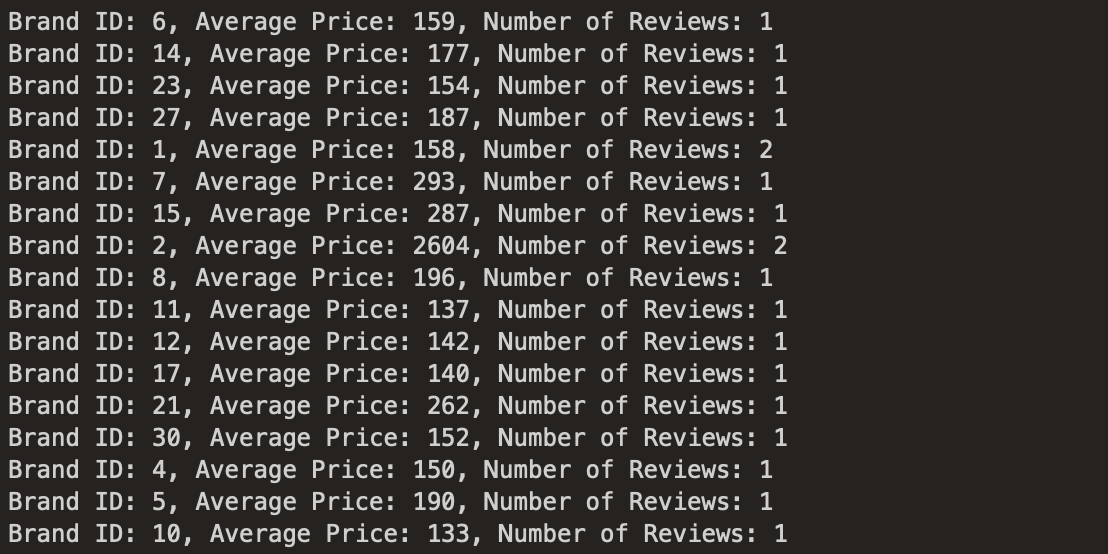
****

Procedure named get\_product\_grouper \_by\_bid performs group operation inside the procedure. It groups by b\_id attribute in product table. It outputs average price and number reviews for each brand id. To output all rows for loop is used , it prints every b\_id and its average price and number of reviews



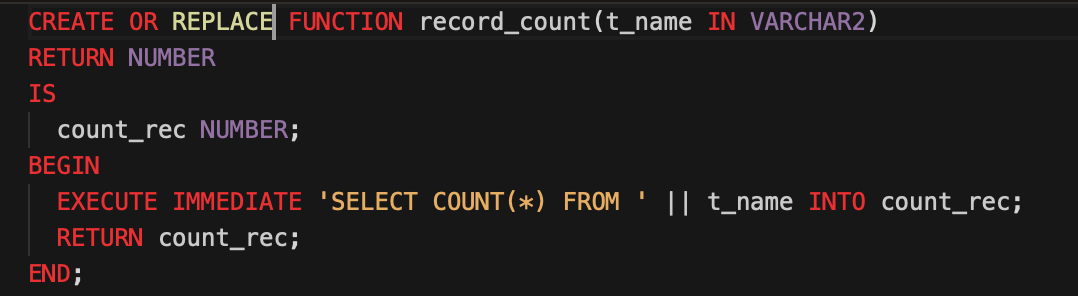
To execute this procedure we use this structure, in this structure we just call the name of procedure.

Ouput:

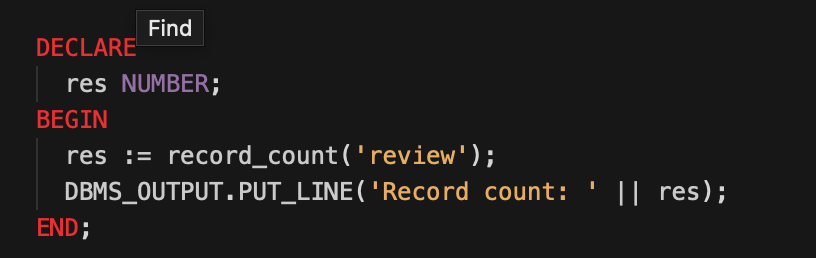


And so on.

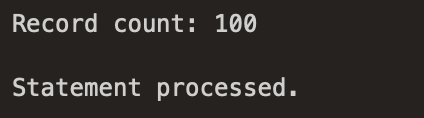
**- Function which counts the number of records**



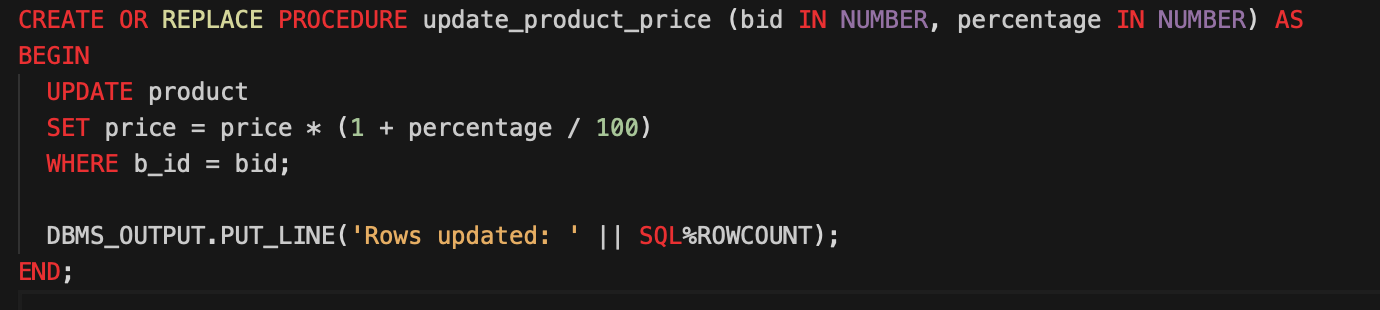
Record count function is function that count number of records in provided table. Functions accepts t\_name table name as input type VARCHAR2 and return number. After declaring the count\_rec, we do execute immediate command to make the sql dynamic, because if it will remain static t\_name will not be readed and will give error (table not found). Inside the sql part it performs basic count command, and function replaces the t\_name with provided table name. All of this data is stored inside count\_rec and then returned;



To execute this function, we used this piece of written code. In this example table ‘review’ is passed. Any existing table name can be passed and the function returns:



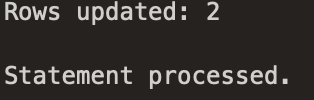
**- Procedure which uses SQL%ROWCOUNT to determine the number of rows affected**



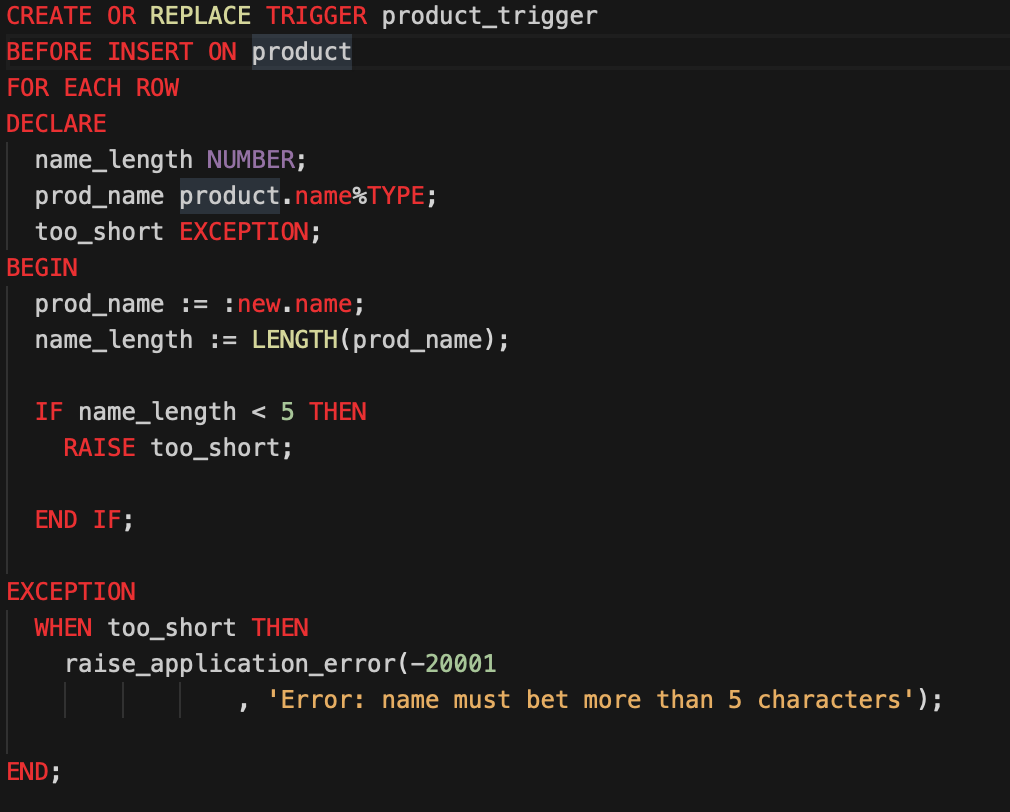
Update\_product\_price procedure return number of affected rows after the change of price of given brand. Procedure accepts the brand id and percentage that the price should be increased. At the end it prints the affected rows by SQL%ROWCOUNT command.



To execute this procedure we used this code, where we set the b\_id as 1 and percentage as 5 and it printed:

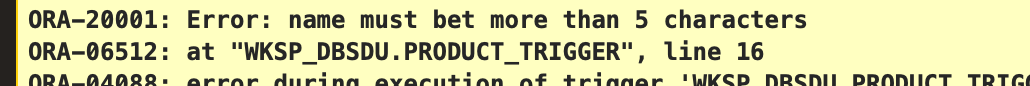


**- Add user-defined exception which disallows to enter title of item (e.g. book) to be less than 5 characters**

****

This user defined exception is implemented using trigger named product\_trigger, main function is before insertion it checks for length of name attribute and if name length is less than 5 it will raise user defined exception named too\_short. If exception is raised it will stop the insert and raise error -20001 and print the given message.

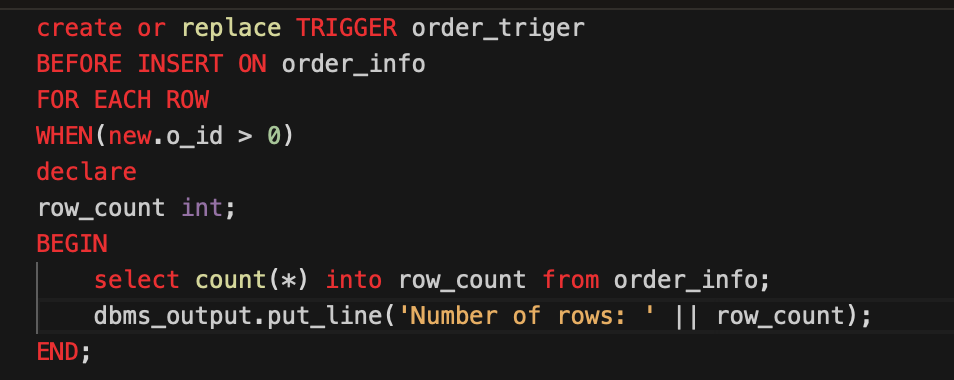
To test this exception we can use this query and it will output:



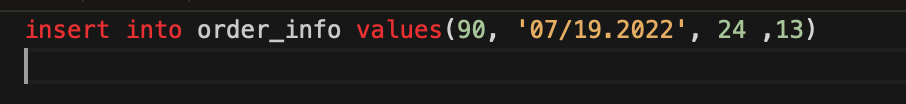
This error will stop the insert and output this text.

**- Create a trigger before insert on any entity which will show the current number of rows in the table**

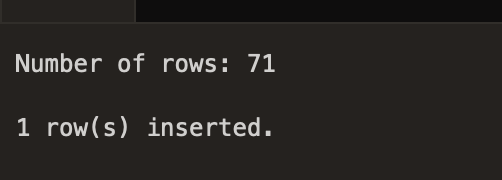
**1)**

****

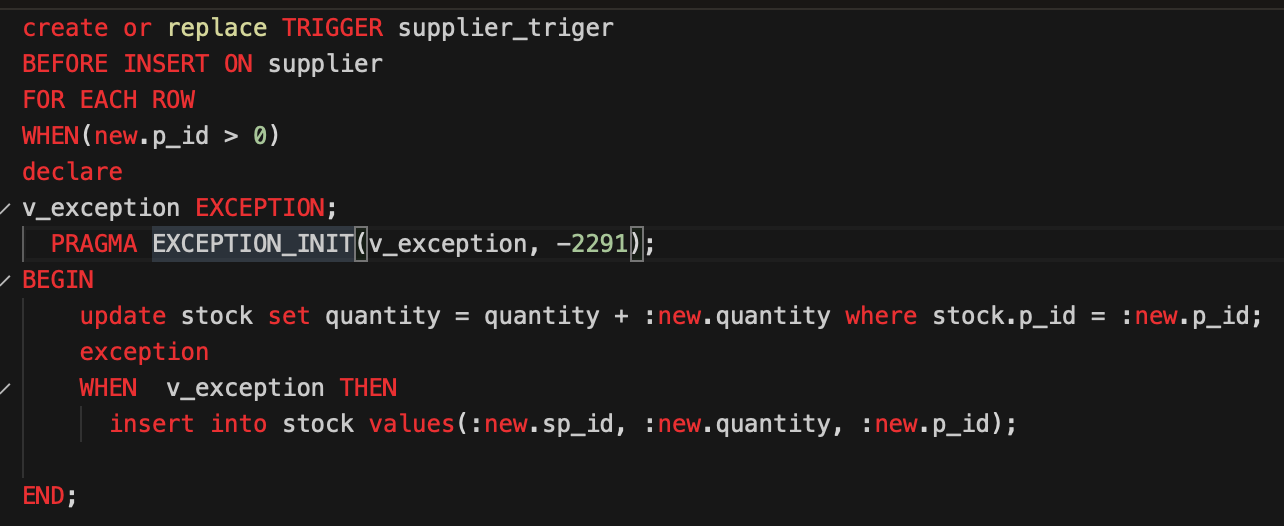
This trigger count number of rows before insertion on table order\_info, row\_count as number data type is declared is declaration part. Then in the main part we perform select query with count command that store ouput in row\_count and then row\_count is printed at the end.



To check the trigger we can enter this test data to table order\_info and output will be:

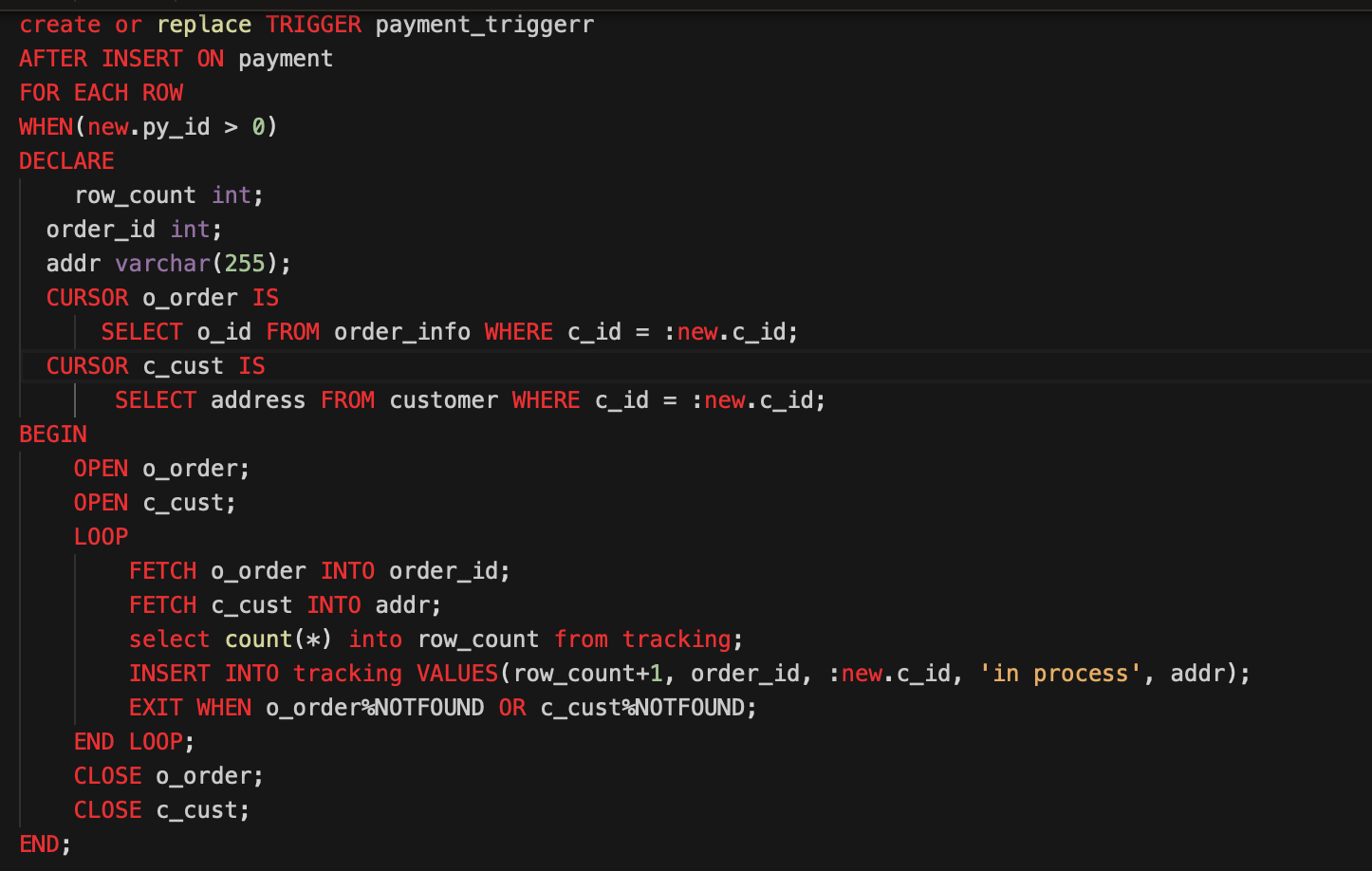


**2)**



This trigger updates the quantity in stock in data was inserted into supplier. Before insertion trigger checks for product is stock and if it exists it will update it and if it does not exist the user defined execution will triggered that will insert product to stock. 

It can be triggered with this query which insert new supply to supplier table and P\_id is id of products that is not in stock table. So, the exception will be triggered and it will add it to stock table.

**3)** 

This trigger works after insertion into the table. It should insert new data into the Tracking table. Since after the customer pays for the goods, a new tracking data should be created. In declaration part we have row\_count, order\_id as int data type , addr as varchar with 255 limit and two cursor which are used in trigger. Open cursor\_name is used to execute and enable cursor and then data from cursor is fetched to order\_id and addr. And then data is inserted to tracking table. Close cursor\_name is used to stop execution of cursor. To trigger this trigger we need to insert data to payment.

**The end**