**FASHION ANYTIME**

**DATABASE DESIGN DOCUMENT**

**TABLE OF CONTENTS**

1. FASHION ANYTIME PROJECT OVERVIEW ……………………………………..…………... 1
2. FASHION ANYTIME USE CASE AND FIELDS………………………………………………….1
3. INTIAL FASHION ANYTIME ERD ………………………………………………………………...7
4. FASHIO ANYTIME SPECIALIZATION-GENERALIZATION RELATIONSHIP……………….8
5. FASHION ANYTIME CONCEPTAUL ERD……………………………………………………….12
6. FASHION ANYTIME RELATIONSHIP CLASSIFICATIN AND ASSOCIATIVE......................12

MAPING

1. FASHION ANYTIME SPECIALIZATION-GENERALIZATION MAPPING……………………13
2. INITIAL DBMS PHYSICAL LEVEL ERD…………………………………………………………13
3. ADDING ATTRIBUTES TO INITIAL PHYSICAL DBMS ERD…………………………………14
   1. ATTRIBUTE DBMS ERD……………………………………………………………………..17
4. FASHION ANYTIME NORMALIZATION………………………………………………………...18

10.1 FASHION ANYTIME DATA DICTIONARY AFTER NORMALIZATION………………...18

1. FASHION ANYTIME DB……………………….…………………………………………………..20
2. FASHION ANYTIME INDEXING…………………………………………………………………..21

12.1 FOREIGN KEY INDEXING…………………………………………………………………..24

12.2 QUERY DRIVEN INDEXING…………………………………………………………………27

1. INSERTED DATA ON FASHION ANYTIME TABLES…………………………………………..28
2. FASHION ANYTIME QUESTIONS AND QUERIES………………………….………………….31
   1. GROUP1 QUEREIS………………………………………………………………………..32
   2. GROUP2 QUERIES………………………………………………………………………..33
3. FUNCTION,STORE PREOCEDURE……………………………………………………………...36
   1. RETURNS THE TOTAL NUMBER OF THE PRODUCTS
   2. CHECK ORDER IS ELIGINLE FOR FREE SHIPPING OR NOT
   3. RETURNS THE TOTAL QTY OF PRODUCT WITH A GIVEN ID
   4. CHECK ORDER STATUS SHIPPING CAN'T CANCEL
4. IMPLEMENTING TRANSACTION IN FASHION ANYTIME ……………………………………..42

16.1 CASH ON DELIVERY -- ORDER STATUS COMPLETED FROM SHIPPING STAGE WITH TRIGGER

1. MAINTAIN HISTORY TABLE WITH TRIGGER…………………………………………………..45

17.1 PRODUCT PRICE CHANGE WITH TRIGGER

17.2 CUSTOMER'S ADDRESS CHANGE WITH TRIGGER

1. SUMMARY AND REFLECTION…………………………………………………………………49

**1) Fashion anytime project overview**

* Fashion anytime is a database project to keep the inventory up to date such as what item is out of stock, billing, shipping status, purchase order and many more.
* SQL script is created for sample for sample database schemas.
* A new user can register.
* A customer can filter the product based on product details.
* If admin want to view sum total of all the billed orders for the month.

**2) Fashion anytime use cases and fields**

**User\_Info**

|  |  |
| --- | --- |
| **Field** | **What it stores** |
|  |  |
| user\_id | It stores the information about user. Each user has a unique id. |
| user\_type | It stores the information about the user type (customer/seller/admin) |

**Customer\_master**

|  |  |
| --- | --- |
| **Field** | **What it stores** |
|  |  |
| cust\_id | It stores the information about customer. Each customer has a unique id. |
| cust\_name | It stores the name of the customer. |
| cust\_add | It stores the information about customer’s address. |
| cust\_phone\_no | It stores the customer’s mobile no. |
| cust\_email\_add | Stores the customer’s email address. |

**Salesman\_master**

|  |  |
| --- | --- |
| **Field** | **What it stores** |
| seller\_id | It stores the information about seller. |
| seller\_name | It stores the name of the seller. |
| seller\_add | It stores the seller’s address. |
| seller\_phone\_no | It stores the seller’s mobile no. |
| seller\_email\_add | Stores the seller’s email address. |

**Signup use case**

There are a several components in play for this use case. However, I am focused on what the database needs to store this information. For sign up use case generally customer\_master and Salesman\_master tables are required to store the details of customer and sellers.

* The user enters their information and the account is created in the database.

The structural rules for the customer\_master and salesman\_master entity is each customer/seller has one account and each account belongs to one customer and seller.

**Category\_master**

|  |  |
| --- | --- |
| **Field** | **What it stores** |
| pro\_cat\_id | It stores the information about product category id. |
| pro\_cat\_type | It stores the information about type of the categories for the product. |

**Product\_master**

|  |  |
| --- | --- |
| **Field** | **What it stores** |
|  |  |
| prod\_id | It stores the information about product id. |
| pro\_name | It stores the name of the product. |
| color | It stores the information about the product color. |
| pro\_size | It stores the information about the size of the product. |
| qty\_on\_hand | It stores the information about product quantity. |
| price | It stores the price of the product. |
| seller\_id | It stores the information about seller. |
| pro\_cat\_id | It stores the information about category id. |

**Seller lookup Use case**

From this use case we can get the information about seller. What kind of the products sell by seller. We can derive this information from seller and product master table.

1. Seller can login on site and sell their products. Seller can sells many products and many products sell by many seller.
2. Seller can also stop selling their product when product is not available.

Structural rule for salesman and product master entity is sellers can sell many products and many products are shelled by many sellers.

**Sales\_order**

|  |  |
| --- | --- |
| **Field** | **What it stores** |
|  |  |
| order\_id | It stores the information about order id. |
| order\_date | It stores the information about the order date. |
| order\_status | It stores the information about order status. |
| cust\_id | It stores the information about customer. |
| seller\_id | It stores the information about seller id. |

**Sales\_order\_details**

|  |  |
| --- | --- |
| **Field** | **What it stores** |
|  |  |
| prod\_id | It stores the information about product id. |
| order\_id | It stores the information about order |
| qty\_ordered | It stores information about number of the quantity ordered. |
| pro\_rate | It stores the price of the product. |

**Payment\_Info**

|  |  |
| --- | --- |
| **Field** | **What it stores** |
|  |  |
| payment\_id | It stores the information about payment id. |
| payment\_type | It stores information about the type of payment. |
| payment\_date | It stores the information about payment date. |
| cust\_id | It stores the information about customer who is making payment. |
| order\_id | It stores the information about order id. |

**Purchase use case**

This use case describe the functionality about add to cart features when customer place the order. For add to cart use case mainly customer, product, sales order ,cart and payment info database are required to store the information.

1. The person login on site and make a purchase.
2. The application fetches all purchases matching the criteria from the database and user can select their purchase as per their choice. Customer can place many orders for respected product as per their availability.
3. When person select product and perform add to cart operation each purchase is associated with respected customer’s account.

**Structural rule for these entity are:**

1. There is no guest account is available on site so before placing the order customer compulsory need to create an account.
2. Customer may place zero or many orders.

**Product\_price\_history**

|  |  |
| --- | --- |
| **Field** | **What it stores** |
|  |  |
| price\_chng\_id | It stores the information about product’s price change id. |
| prod\_id | It stores the information about product id. |
| pro\_old\_price | It stores the information about old price of the product. |
| pro\_new\_price | It stores the information about new price of the product. |
| pro\_price\_change\_date | It stores the date when price of the respected product is changed. |

**Customer\_master\_history**

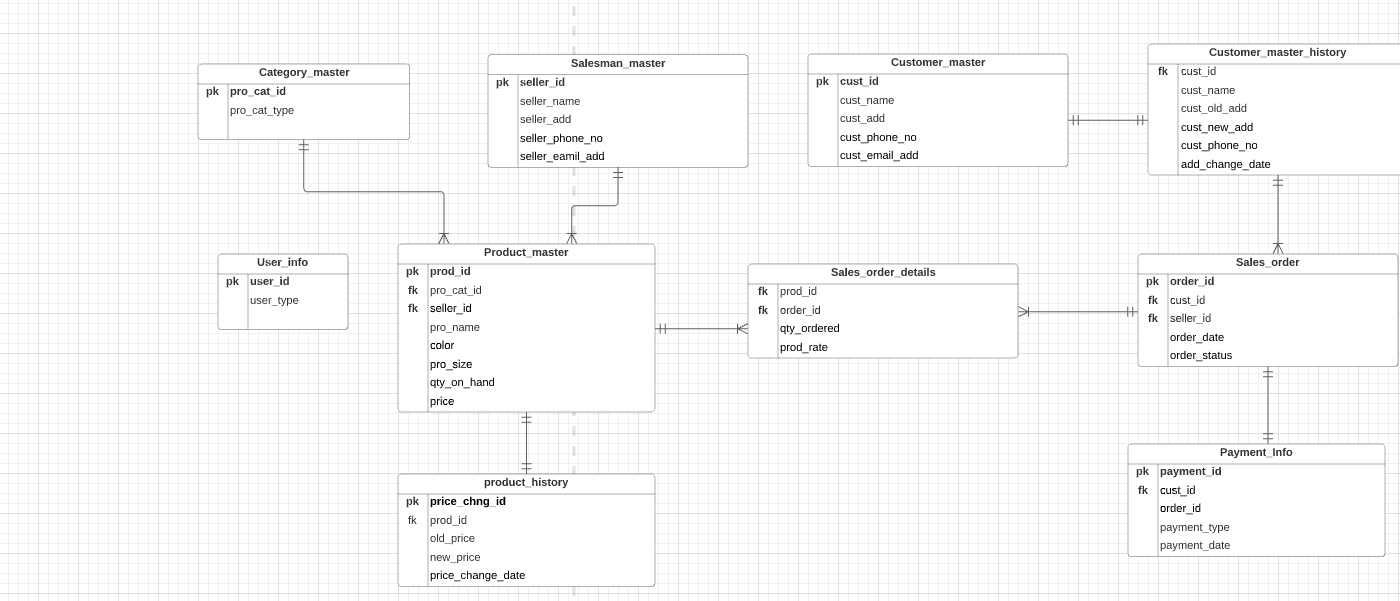
|  |  |
| --- | --- |
| **Field** | **What it stores** |
|  |  |
| cust\_id | It stores the information about the customer’s id. |
| cust\_name | It stores the information about the customer’s name. |
| cust\_old\_add | It stores the information about old address of the customer. |
| cust\_new\_add | It stores the information about new address of the customer. |
| cust\_phone\_no | It stores the customer’s mobile number. |
| add\_change\_date | It stores the date when address of the respected customer is changed. |

Here is the summary of the structural rules for Iteration2:

1. Each customer/seller has one account and each account belongs to one customer and seller.
2. A category may assign to many products.
3. Product is assign to zero or many orders.
4. Sellers can sell many products and many products are shelled by many sellers.
5. Customer may place zero or many orders.
6. Customer makes payment for their respected order.

Based on the above rules, below is ERD using crow’s foot style. The structural rules and the ERD covers the main entities of my database; customer, salesman, product master, cart or payment, history details.

1. **Initial fashion anytime ERD**



**4) Fashion anytime specialization and generalization relationships**

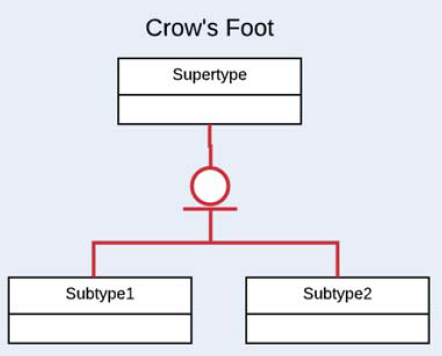
**GENERALIZATION**

* Generalization is the process of extracting common properties from a set of entities and create a generalized entity from it.
* It is a bottom-up approach in which two or more entities can be combined to form a higher level entity if they have some attributes in common.
* Subclasses are combine to make a super class.
* Generalization is used to emphasize the similarities among lower-level entity set and to hide difference in the schema.

**SPECIALIZATION**

* It is opposite of generalization.
* In specialization, an entity is broken down into sub-entities based on their characteristics.
* Specialization is a top-down approach where higher level entity is specialized into two or more lower level entities.
* Specialization is used to identify the subset of an entity set that shares some distinguishing characteristics.

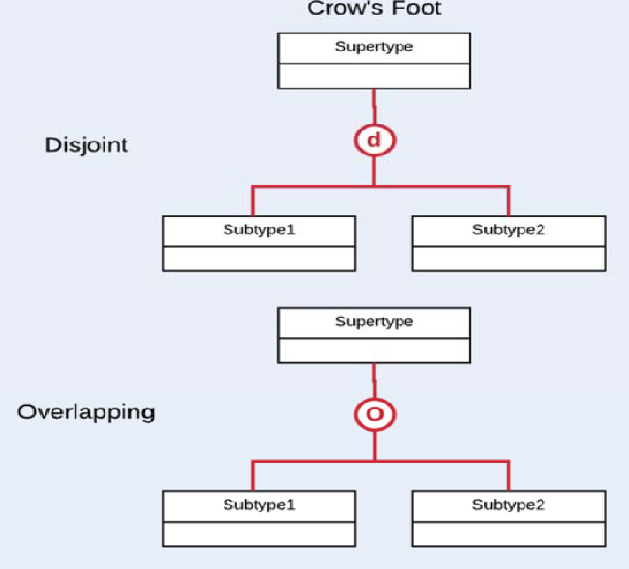
**Diagrammatic Representation of specialization – generalization existence**



**Constraints**

1. **Disjoint constraints: -** A super type instance may simultaneously be a member of two or more subtypes. It describes the relationship between members of the subclasses and **constraint** indicates whether it is possible for a member of a super class to be a member of one, or more than one, subclass.
2. **Overlap constraints:** - Itdetermines whether or not two subclasses can contain the same entity.

**Diagrammatic representation of constraints**

****

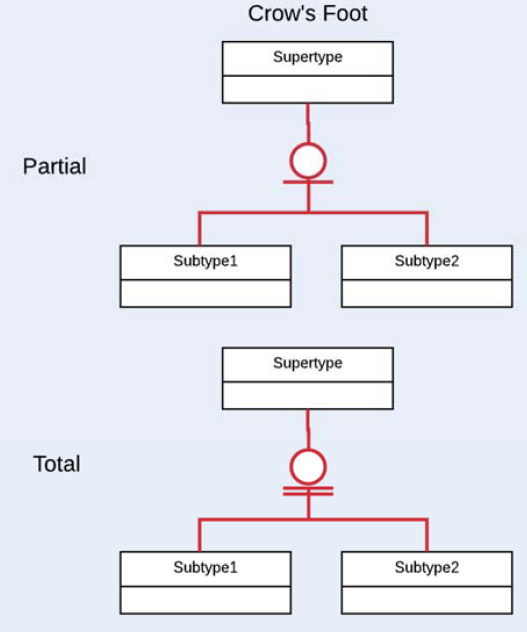
**Completeness Constraints**

The completeness constraints specify whether each entity super type occurrence must also be a member of at least one subtype. The completeness constraint can be partial or total.

1. **Partial completeness: -** Partial completeness means that not every super type occurrence is a member of sub type.

**b) Total completeness:** **-** Total completeness means that every super type occurrence must be at least member of one sub type.

**Diagrammatic Representation of Completeness**

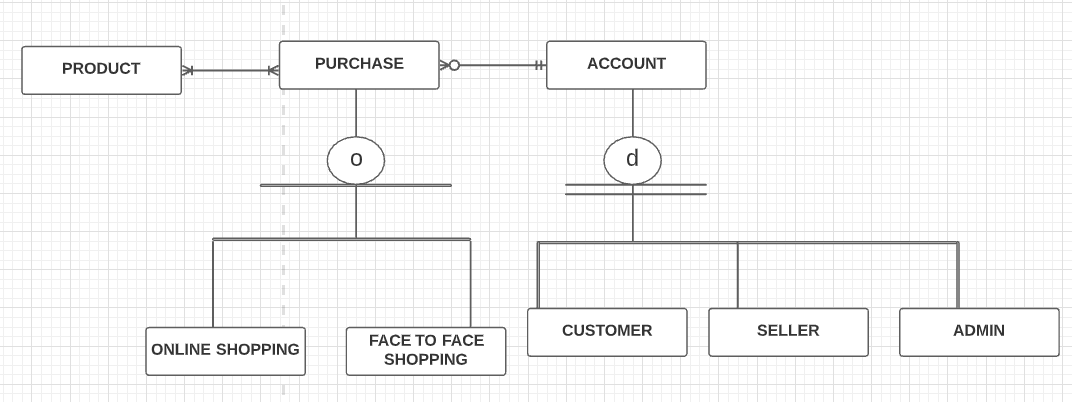


1. A customer can purchase product online or can visit to store and make their purchase.

Using Crow’s Foot, we use the “O” to indicate that the relationship is overlapping, since the purchase can be online or face to face or both ( people visit the store and some items they don’t find from the store for that respected item if it is available online then, they can place the order ) or none.

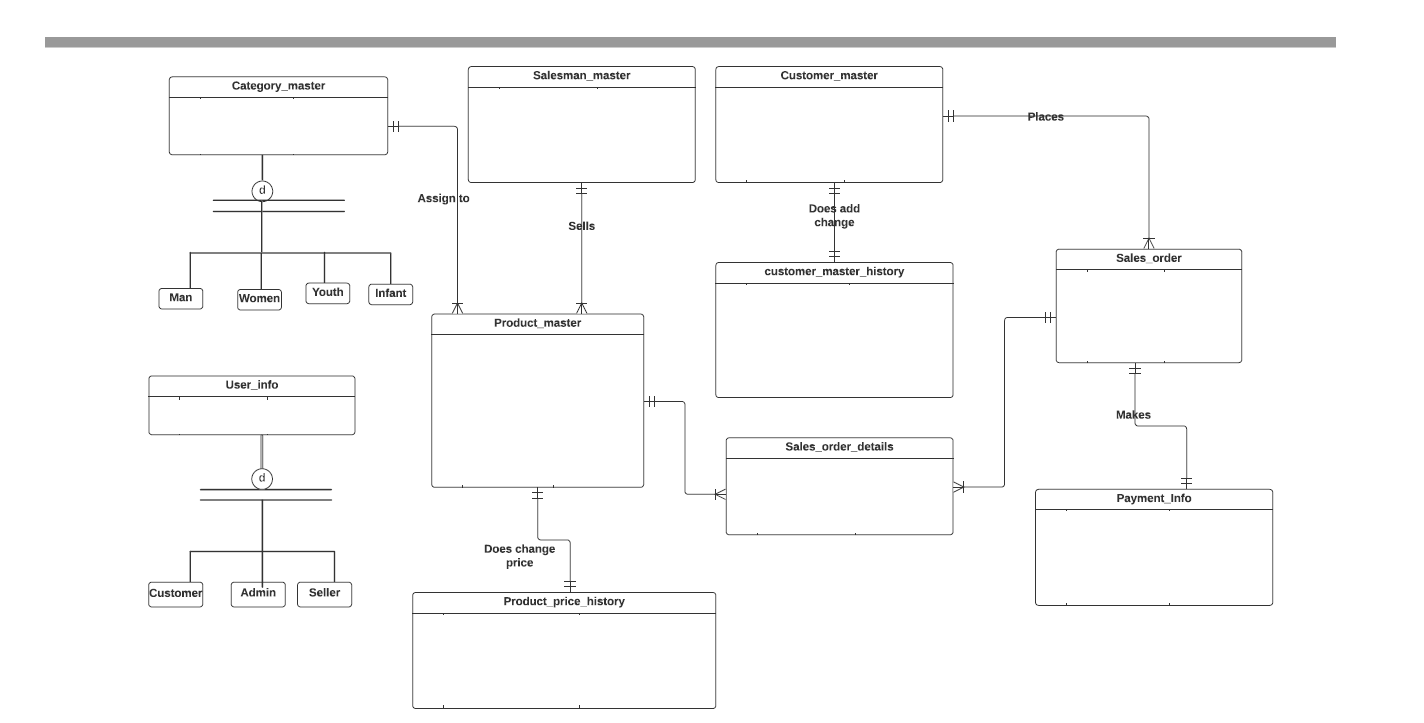
1. A user is a customer or seller.

Relationship is complete and disjoint: - Users can be either a customer or a seller. The relationship and complete and disjoint since there are only going to be these two types of users and the users cannot be both types.



Above relationships are added to my below initial version of ERD and created my conceptual ERD.

**5)Fashion anytime conceptual ERD**

****

**6)Fashion anytime Relationship classification and Associative mapping**

Conceptual ERD is used to identify the relationships. The associative relationships in my conceptual ERD are:

1. Category\_master/product\_master :- (1:M) A category may assign to many products.
2. Salesman\_master/product\_master :- ( M:N) Sellers can sell many products and many products are shelled by many sellers.
3. Customer\_master/Sales\_order : - (1:M) Each customer can have or place many sales orders.
4. Product\_master/Sales\_order\_details/Sales\_order : - (M:N) An **Order** can have multiple **Products, a Product** can be ordered in multiple **Orders. A bridge entity** ,**Sales\_order\_details** will have the foreign key to both product\_master and Sales\_order entities.
5. Sales\_order/Payment\_info : - (1:1) Each customer makes payment for their respected order.
6. Product\_master/Product\_price\_history – (1:1) Each product price will have exactly one product price history. ( Product price is going to change in some interval of time period.)

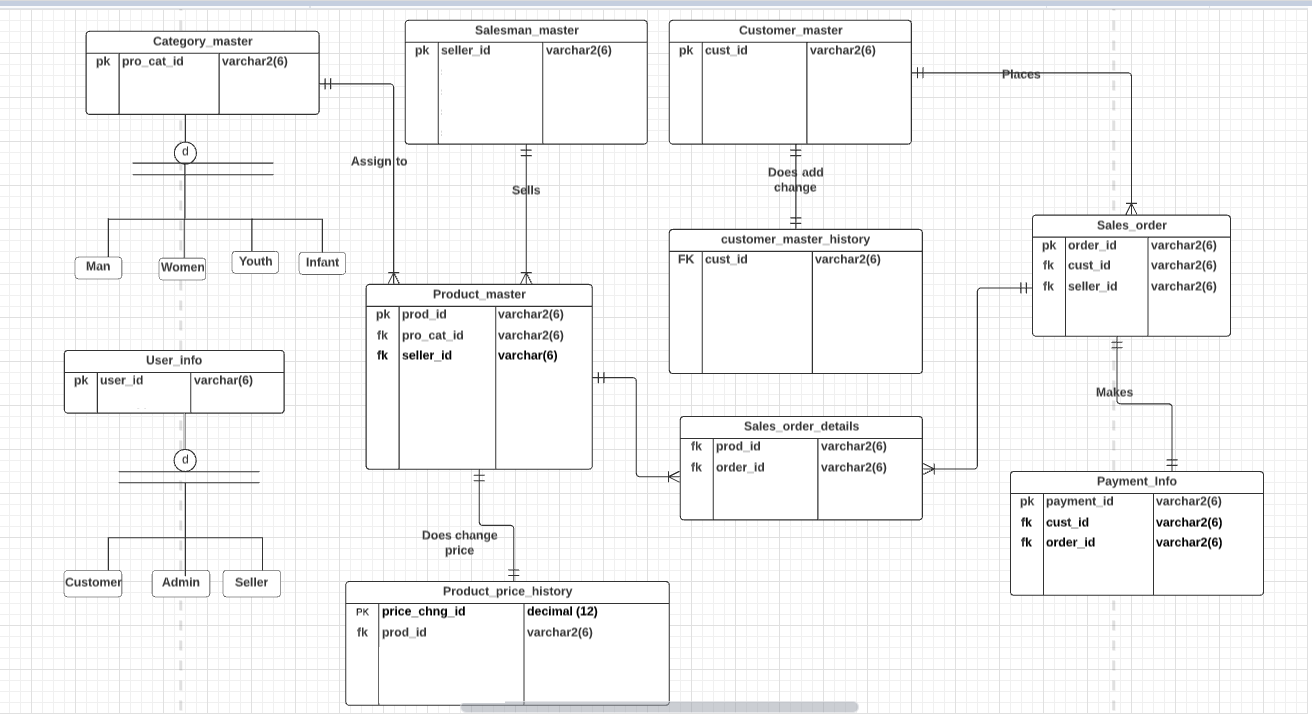
**7)Fashion anytime Specialization-Generalization Mapping**

There is specialization-generalization mapping in my conceptual ERD, for category entity. We use double bars to indicate the relationship is totally complete, since the list is exhaustive; a product must fit into one of these categories.

**Below is the DBMS physical ERD with the relationships mapped into them.**

1. The additional entities under Category\_master are: (Man,Women,Youth,infant). Pro\_cat\_id is a primary key which reference the foreign key of product\_master entity.
2. Salesman master has a primary key seller\_id which is reference the foreign key in sales order entity.
3. Sales\_order has a order\_id , primary key which is reference the foreign key in sales order details entity.
4. Sales order details has a cart \_id , primary key which is reference the foreign key in payment\_info entity.

**8) INITIAL DBMS PHYSICAL LEVEL ERD**

****

* Following are the major usages supported by the database:

1. A new user can register on the application.
2. A seller/customer can update their details.
3. A customer can filter the product based on product details.
4. A customer can add or delete a product from the cart.

* The structural rules and the conceptual ERD covers the main entities of my database; customer\_master, salesman\_master, product\_master, sales\_order, sales\_order\_details, payment\_info,product\_price\_history and the relationship between them.
* The design reflects the hierarchy of category of product – man,women,youth,infant.The design also captures a hierarchy of online shopping . Now, I have my entities identified with primary keys, foreign keys and few attributes. Also, I have my relationships defined between entities following the business rules. Now this design can be used for my further iterations.

**9)Adding attributes to Initial Physical DBMS ERD**

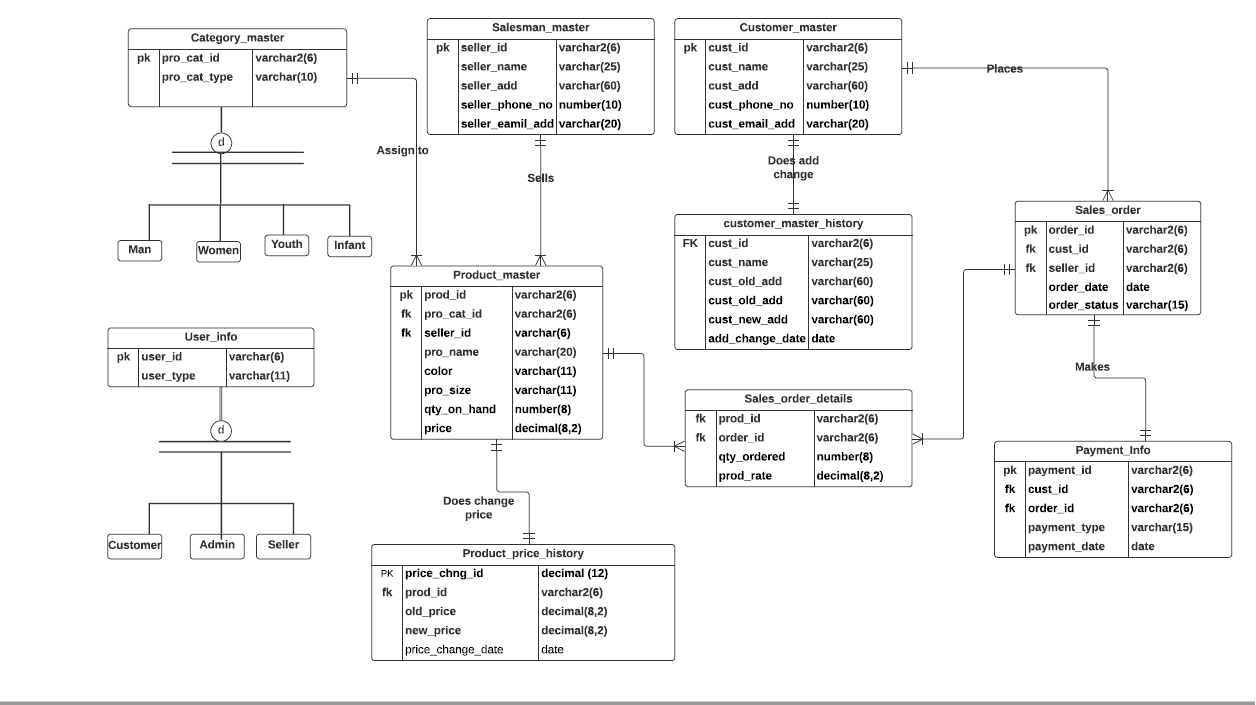
**Fashion anytime database attributes:-** In previous iterations we identified attribute for Initial level Physical DBMS ERD. For your reference above I draw the initial level physical DBMS ERD which identifies the initial level attribute and now I am going to enhance the initial level DBMS ERD with adding more attributes on the entities.

Below I have captured the attributes for each of the entities as per the use cases and the structural database rules.

|  |  |  |  |
| --- | --- | --- | --- |
| **TABLE** | **ATTRIBUTE** | **DATATYPE** | **REASON** |
|  |  |  |  |
| User | user\_type | Varchar(11) | App will show different UI based on the user logged in. This field stores data on kind of user such as customer, seller, admin etc.I allow upto 11 characters. |
| Customer\_master | cust\_name | Varchar(25) | Every customer has a name which acts like the identification for the customer.I allow upto 25 characters. |
| Customer\_master | cust\_addr | Varchar(60) | Every customer has a address.I allow upto 60 characters. |
| Customer\_master | cust\_phone\_no | number(10) | Every customer has a phone number. I allow up to 10 digits. |
| Customer\_master | cust\_email\_add | Varchar(20) | Every customer has a email address. I allow up to 20 characters. |
| Salesman\_master | seller\_name | Varchar(25) | Every seller has a name which acts like the identification for the customer |
| Salesman\_master | seller\_add | Varchar(60) | Every seller has a address.I allow upto 60 characters. |
| Salesman\_master | seller\_phone\_no | Number(10) | Every seller has a phone number. I allow up to 10 digits. |
| Salesman\_master | seller\_email\_add | Varchar(20) | It stores the information about the seller’s email address.I allow up to 20 characters. |
| Category\_master | pro\_cat\_type | Varchar(10) | This field stores the information about types of the product category. I allow up to 10 characters. |
| Product\_master | pro\_name | Varchar(20) | It stores the information about the product name. I allow up to 20 characters. |
| Product\_master | color | Varchar(11) | It stores the information about the color of the product. I allow up to 11 characters. |
| Product\_master | pro\_size | Varchar(11) | It stores the information about the size of the product. I allow up to 11 characters. |
| Product\_master | quantity\_on\_hand | Number(8) | It stores the information about the product quantity. I allow for up to 8  digits. I also allow for the standard two decimal points. |
| Product\_master | price | Decimal(8,2) | It stores the information about the price of the product. I allow for up to 8 digits and standard two decimal points. |
| Sales order | order\_date | date | This field stores the information about the date when customer place the order. |
| Sales order | order\_status | Varchar(15) | This field stores the information about the order status. I allow up to the 15 characters. |
| Sales order details | qty\_ordered | Number(8) | This field stores the information about the number of the quantity to be ordered for respected product. I allow up to 8 digits. |
| Sales order details | pro\_rate | Decimal(8,2) | It stores the information about the product rate. I allow for up to 8 digits and Standard two decimal points. |
| Payment\_info | payment\_type | Varchar(15) | It is necessary to show payment is done by credit/debit/visa or other type of payment mode. |
| Payment\_info | payment\_date | date | A payment has done on a specific date. |
| Product\_history | old\_price | Decimal(8,2) | It stores the information about the old price of the respected product. I allow for up to 8 digits and Standard two decimal points. |
| Product\_history | new\_price | Decimal(8,2) | It stores the information about the new price of the respected product. I allow for up to 8 digits and Standard two decimal points. |
| Product\_history | price\_change\_date | date | A price has change on a specific date. |
| Customer\_master\_history | cust\_name | varchar2(25) | Every customer has a name which acts like the identification for the customer.I allow upto 25 characters. |
| Customer\_master\_history | cust\_old\_add | varchar2(60) | It stores the information about the old address of the respected customer. I allow for up to 60 characters. |
| Customer\_master\_history | cust\_new\_add | Varchar2(60) | It stores the information about the new address of the respected customer. I allow for up to 60 characters |
| Customer\_master\_history | cust\_phone\_no | Number(10) | Every customer has a phone number. I allow up to 10 digits. |
| Customer\_master\_history | add\_change\_date | date | An address has change on a specific date |

The initial DBMS physical ERD has been enhanced to include the identified attributes and their data types.

**9.1) physical DBMS ERD with attributes**



**10)Fashion Anytime Database Normalization**

After review of entities, attributes and the relationship between them, I do not see any partial or transitive dependencies in any of the entity. Also, there is no redundancy observed. So, the current design is in the third normal form already. At this point, I do not have changes in my design as there was no need for normalizing the entities.

**10.1) Data Dictionary After Normalization**

1. **User\_info**

|  |  |
| --- | --- |
| **Field** | **What it stores** |
| UserId | It stores the information about user. Each user has a unique id. |
| UserType | This field stores data on kind of user such as customer, seller etc. |

But as far as the database is concerned, this entity is not having any relationship with other entities. But this entity will be part of my database even in iterations.

Structural database rule for this use case is:- A user is a customer or seller.In iteration3 when we talked about relationship disjoint at that time already viewed graphical representation about this structural rule.

1. **Customer\_master**

|  |  |  |
| --- | --- | --- |
| **Field** | **Data Type** | **Constraints** |
|  |  |  |
| cust\_id | Varchar2(6) | Primary key , Not null |
| cust\_name | Varchar(25) | Not Null |
| cust\_address | Varchar(60) | Not Null |
| cust\_phone\_no | Number(10) | Not Null |
| cust\_email\_address | Varchar(20) |  |

1. **Salesman\_master**

|  |  |  |
| --- | --- | --- |
| **Field** | **Data Type** | **Constraints** |
| seller\_id | Varchar2(6) | Primary key, Not Null |
| seller\_name | Varchar(25) | Not Null |
| seller\_add | Varchar(60) | Not Null |
| seller\_phone\_no | number(10) | Not Null |
| seller\_email\_add | Varchar(20) |  |

1. **Category\_master**

|  |  |  |
| --- | --- | --- |
| **Field** | **Data Type** | **Constraints** |
|  |  |  |
| pro\_cate\_id | Varchar2(6) | Primary key, Not Null |
| pro\_cat\_type | Varchar(10) | Not Null |

1. **Product\_master**

|  |  |  |
| --- | --- | --- |
| **Field** | **Data Type** | **Constraints** |
|  |  |  |
| prod\_id | Varchar2(6) | Primary key, Not Null |
| Pro\_cat\_id | Varchar2(6) | Foreign key |
| Seller\_id | Varchar2(6) | Foreign key |
| pro\_name | Varchar(20) | Not Null |
| color | Varchar(11) |  |
| pro\_size | Varchar(11) |  |
| qty\_on\_hand | Number(8) | Not Null |
| price | Decimal(8,2) | Not Null |

1. **Sales\_order**

|  |  |  |
| --- | --- | --- |
| **Field** | **Data Type** | **Constraints** |
|  |  |  |
| order\_id | Varchar2(6) | Primary key , Not null |
| order\_date | date | Not Null |
| order\_status | Varchar(15) | Not Null |
| cust\_id | Varchar2(6) | Foreign key, Not Null |
| seller\_id | Varchar2(6) | Foreign key, Not Null |

1. **Sales\_order\_details**

|  |  |  |
| --- | --- | --- |
| **Field** | **Data Type** | **Constraints** |
|  |  |  |
| prod\_id | Varchar2(6) | FK, Not null |
| order\_id | Varchar2(6) | FK, Not null |
| qty\_ordered | Number(8) | Not Null |
| pro\_rate | Decimal(8,2) | Not Null |
| Prod\_id,order\_id |  | Composite pk |

1. **Payment\_info**

|  |  |  |
| --- | --- | --- |
| **Field** | **Data Type** | **Constraints** |
|  |  |  |
| payment\_id | varchar2(6) | PK |
| payment\_type | varchar(15) | Not null |
| payment\_date | date | Not null |
| cust\_id | varchar2(6) | Fk , Not null |
| order\_id | varchar2(6) | Fk , Not null |

1. **Product\_history**

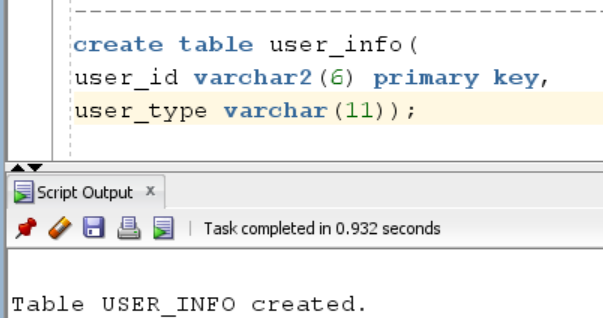
|  |  |  |
| --- | --- | --- |
| **Field** | **Data Type** | **Constraints** |
|  |  |  |
| price\_chng\_id | decimal(12) | Pk, Not Null |
| prod\_id | varchar2(6) | Not Null |
| pro\_old\_price | decimal(8,2) | Not Null |
| pro\_new\_price | decimal(8,2) | Not Null |
| pro\_price\_change\_date | date | Not Null |

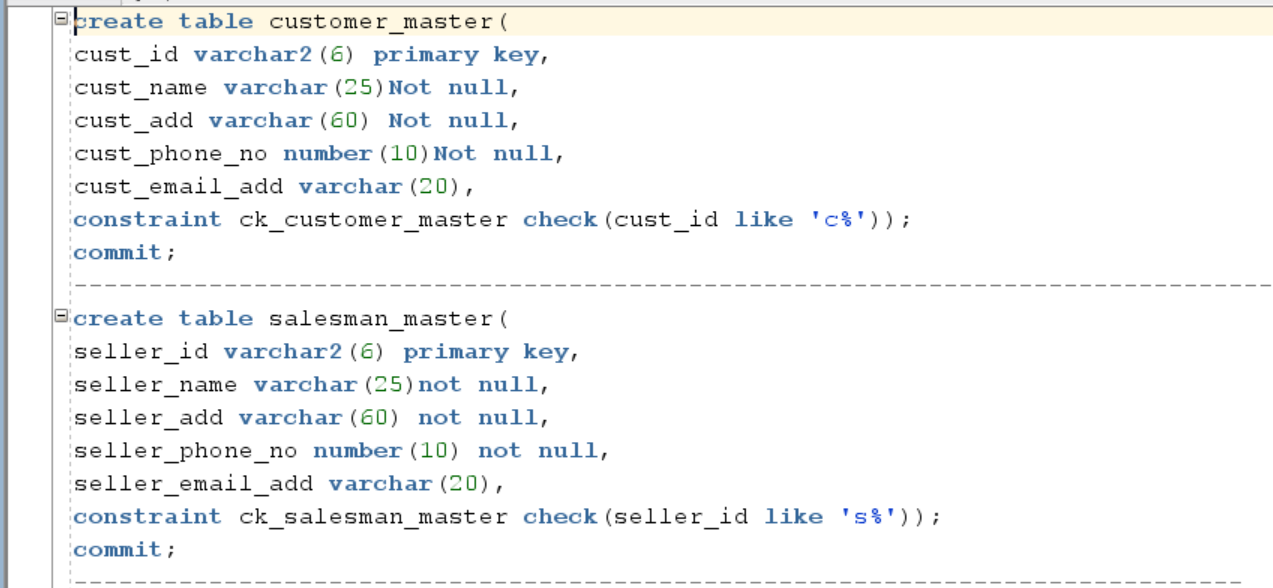
1. **Customer\_master\_history**

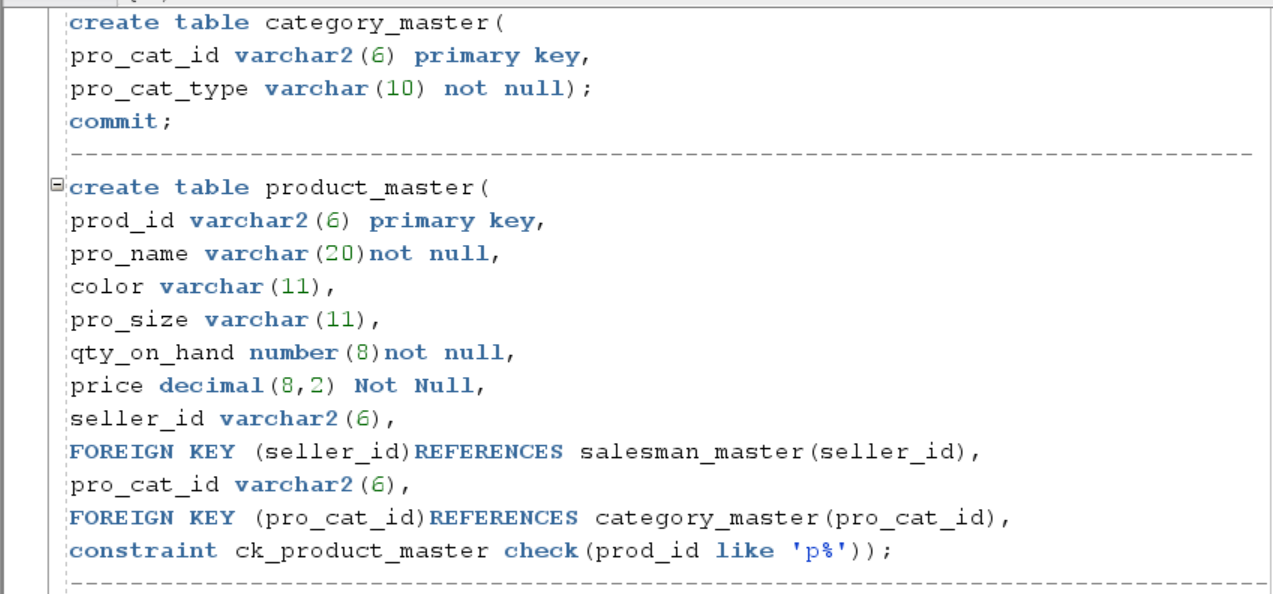
|  |  |  |
| --- | --- | --- |
| **Field** | **Data Type** | **Constraints** |
|  |  |  |
| cust\_id | varchar2(6) | Not Null |
| cust\_name | varchar(25) | Not Null |
| cust\_old\_add | varchar(60) | Not Null |
| cust\_new\_add | varchar(60) | Not Null |
| cust\_phone\_no | number(10) | Not Null |
| add\_change\_date | date | Not Null |

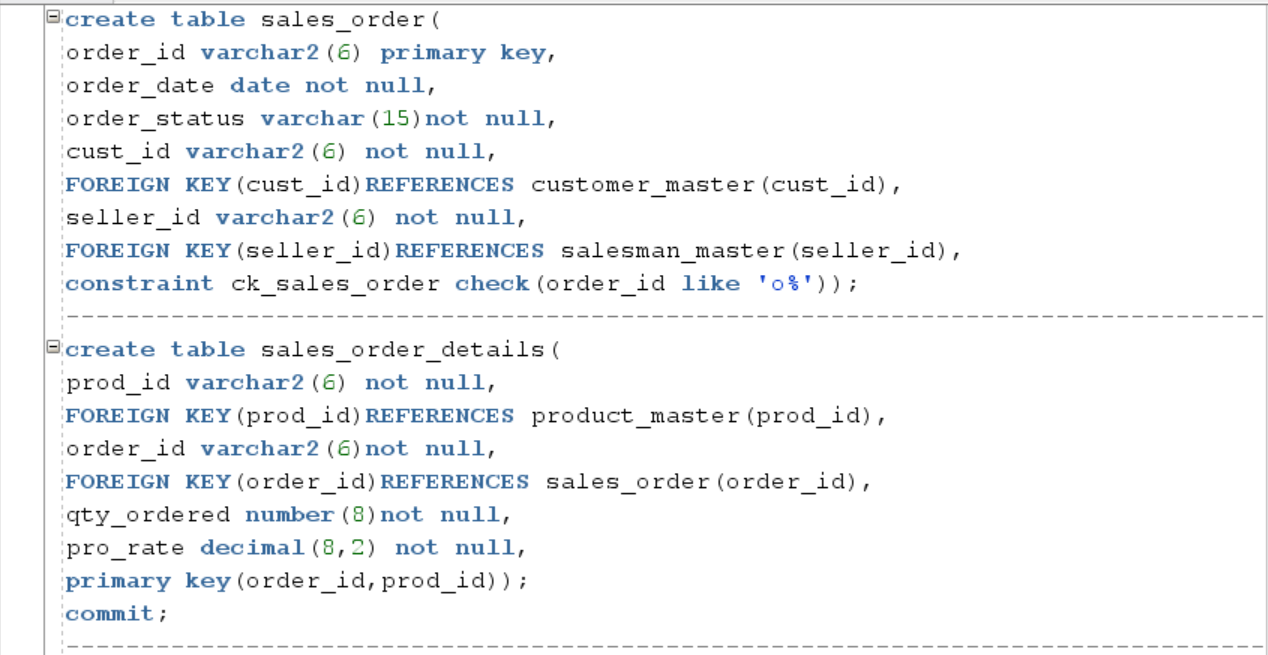
**11)Fashion anytime DB**

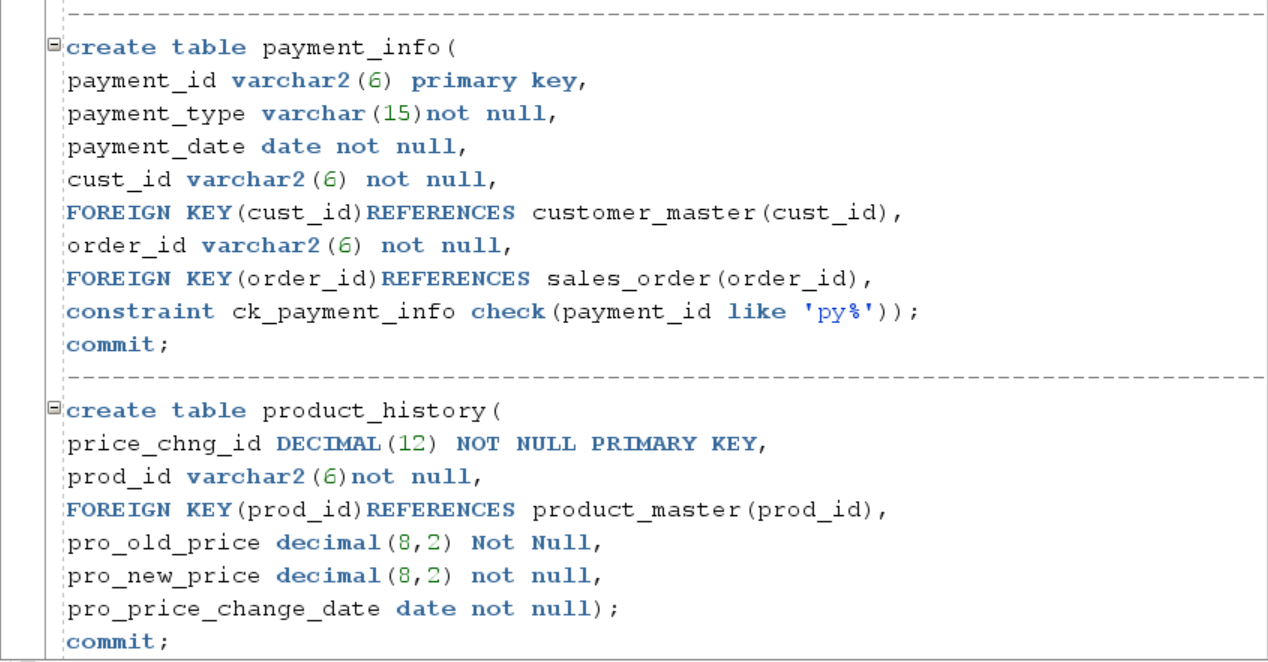
Tables are created with constraints in Oracle.

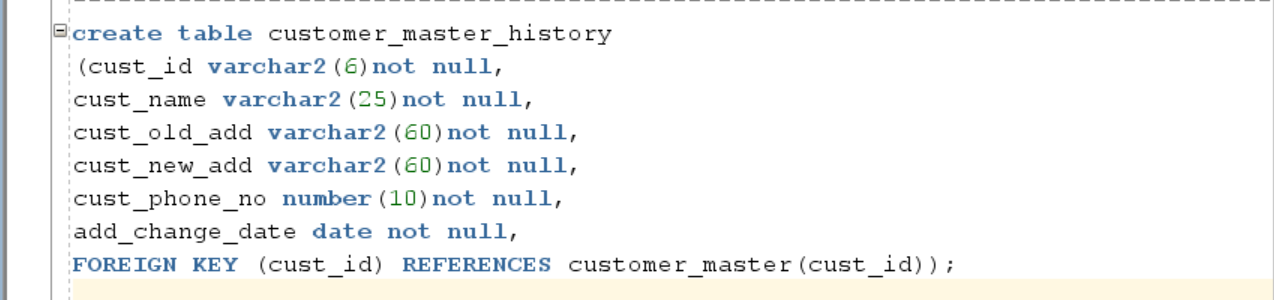




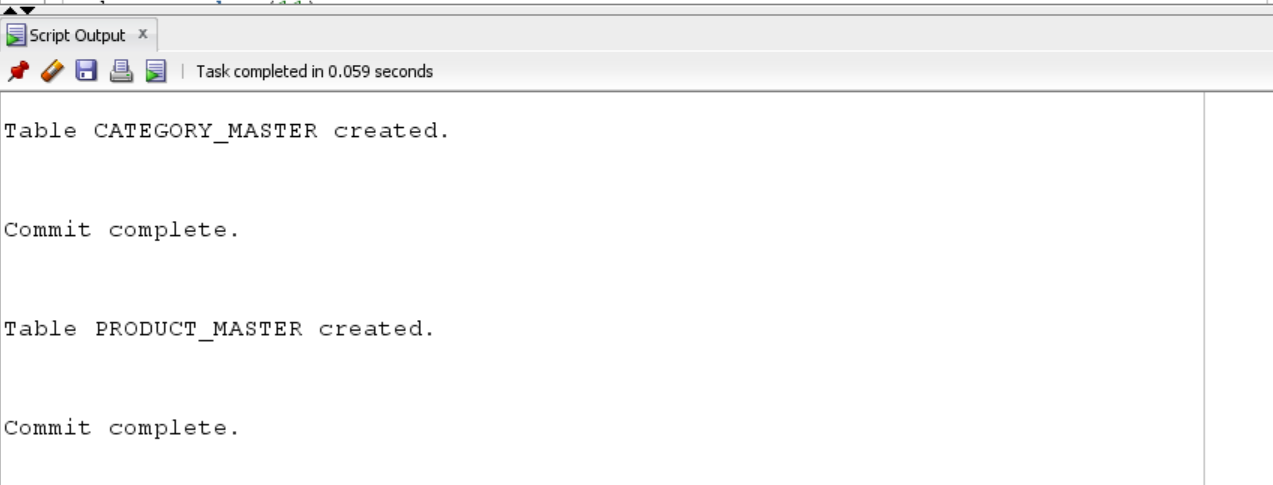


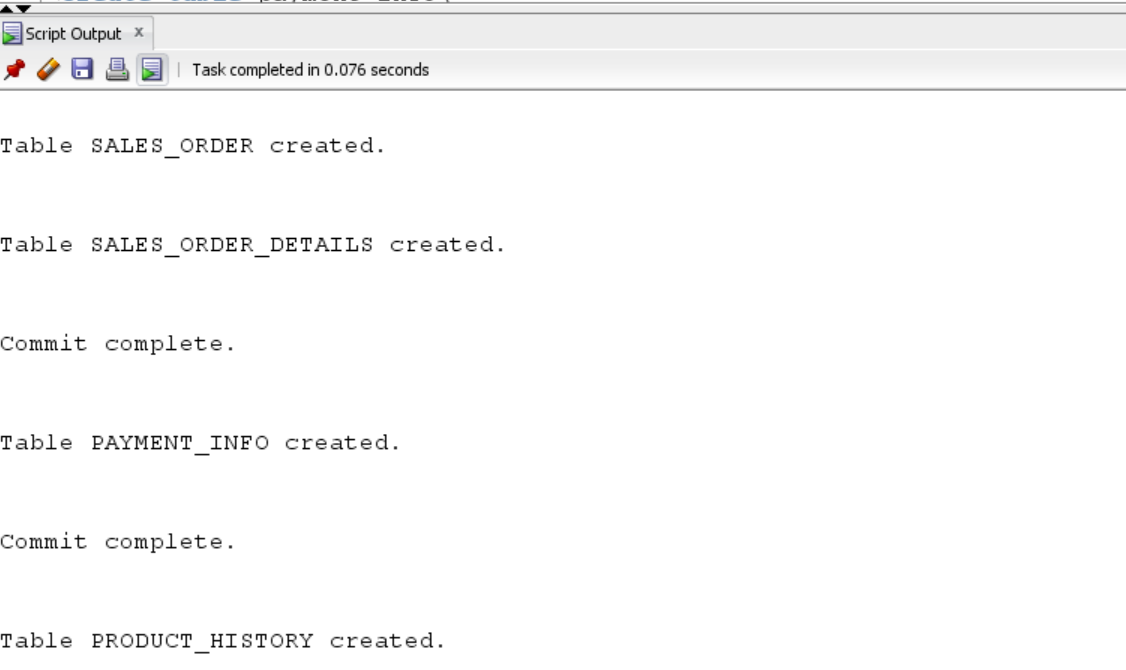


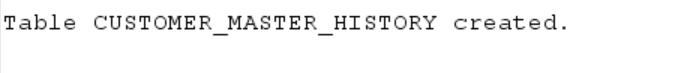












**12) Fashion anytime indexing**

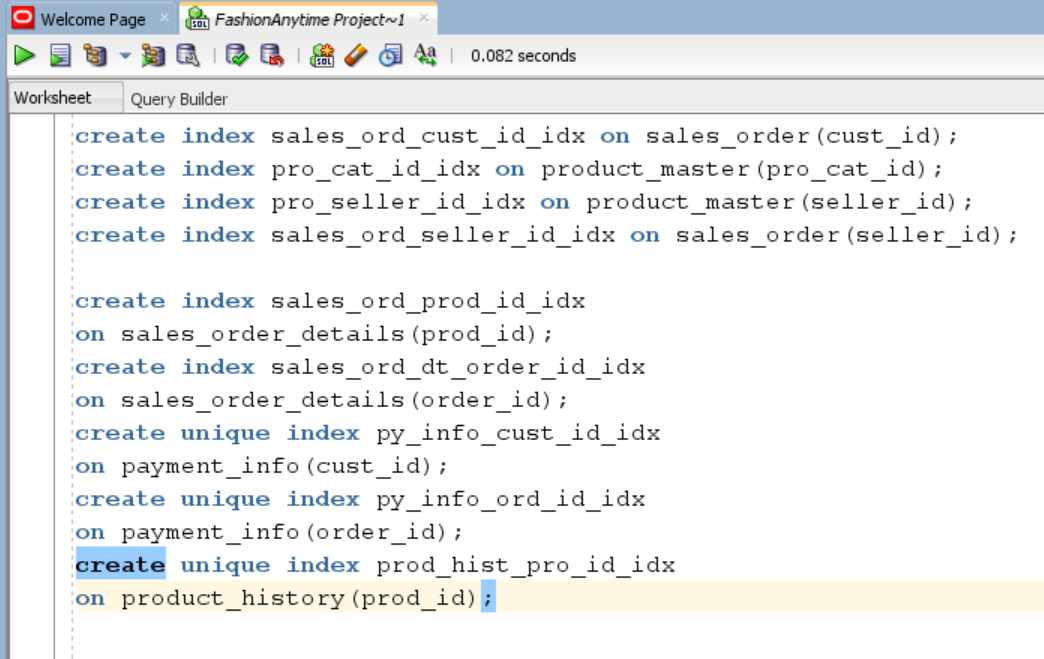
The primary keys of the database as below.

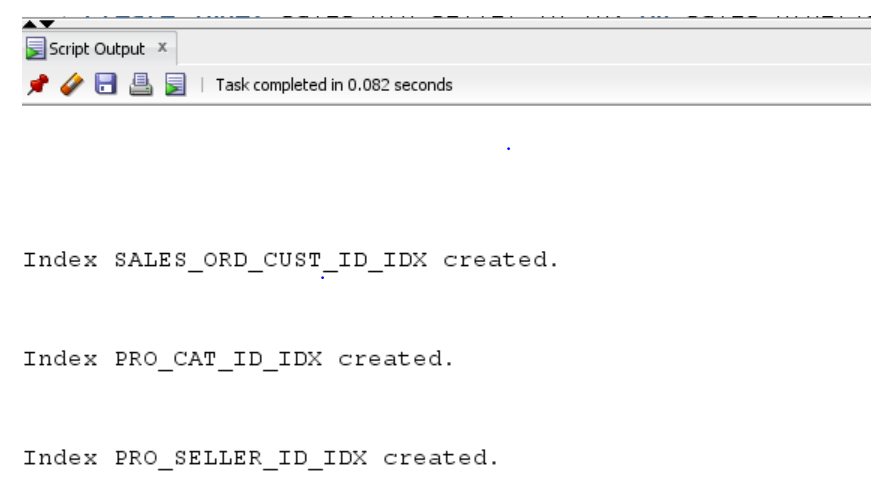
1. Customer\_master.cust\_id
2. Saleaman\_master.seller\_id
3. Category\_master.pro\_cat\_id
4. Product\_master.pro\_id
5. Sales\_order.order\_id
6. Payment\_info.payment\_id

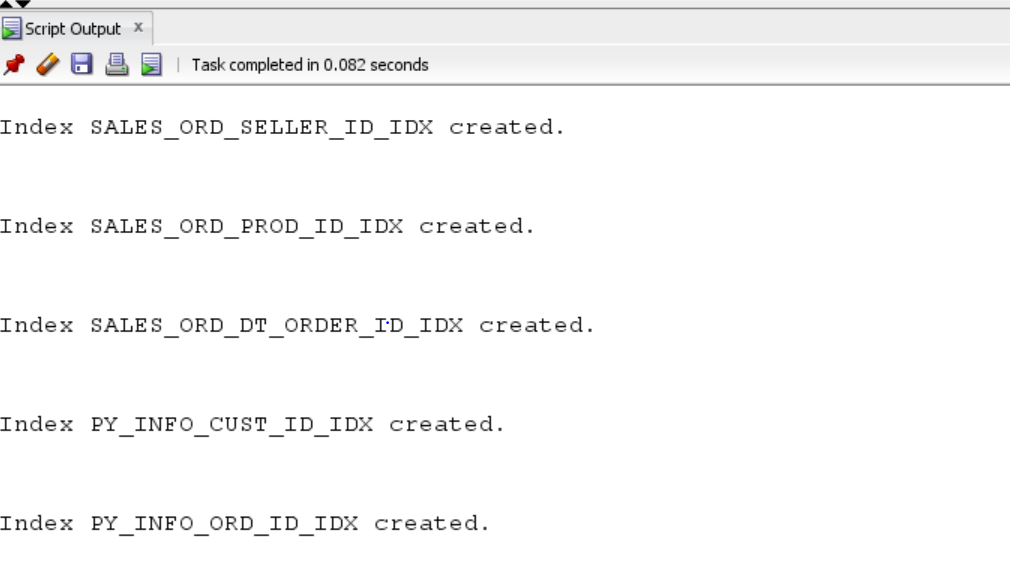
**Foreign keys : -** We index foreign key columns is because foreign keys will almost always be used in the WHERE clauses of SQL queries that perform joins between the referencing tables and the referenced tables. Below is a table identifying each foreign key column.

**Adding foreign key indexes**

|  |  |
| --- | --- |
| **Foreign key column** | **Description** |
| Product\_master.pro\_cat\_id | This is foreign key in the product master table reference the Category\_master table. The index is non-unique since product can belong to different category. |
| Product\_master.seller\_id | This is foreign key in the product master table reference the Salesman\_master table. The index is non-unique since many products can be sell by many seller. |
| Sales order.cust\_id | This is foreign key in the Sales\_order table reference the Customer\_master table. The index is non-unique since there are many customers place the orders. |
| Sales order.seller\_id | This is foreign key in the Sales\_order table reference the Salesman\_master table. The index is non-unique since there are many sellers sell the products. |
| Sales order details.prod\_id | This is foreign key in the Sales\_order details table reference the prodcut\_master table. The index is non-unique. |
| Sales order details.order\_id | This is foreign key in the Sales\_order details table reference the Sales order table.The index is non-unique since order can be produce by many customers. |
| Payment\_info.cust\_id | This is foreign key in the payment\_info table reference the customer table. The index is unique since each customer will have a unique order id for respected order they made. |
| Payment\_info.order\_id | This is foreign key in the payment\_info table reference the Sales order table.The index is unique since respected payment done by every customer have a unique customer id. |
| Product\_history.prod\_id | This is foreign key in the product\_history table reference the Product master table.The index is unique since respected product will have at a time one entry for price change. |
| Customer\_master\_history.cust\_id | This is foreign key in the Customer\_master\_history table reference the customer master table.The index is unique since at a time one entry for address change. |





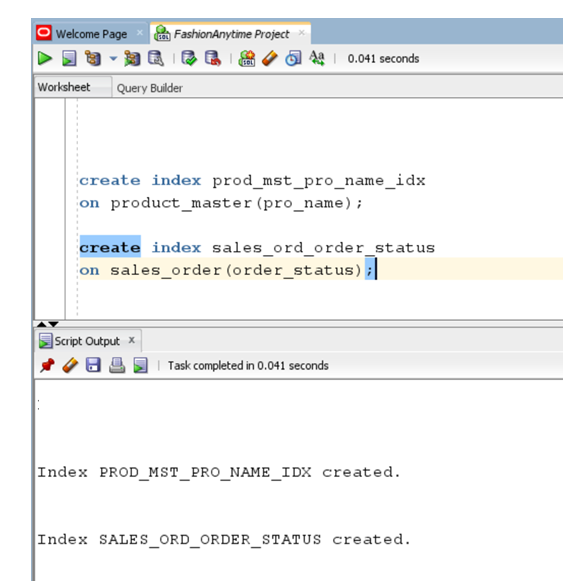




**Query driven indexes**

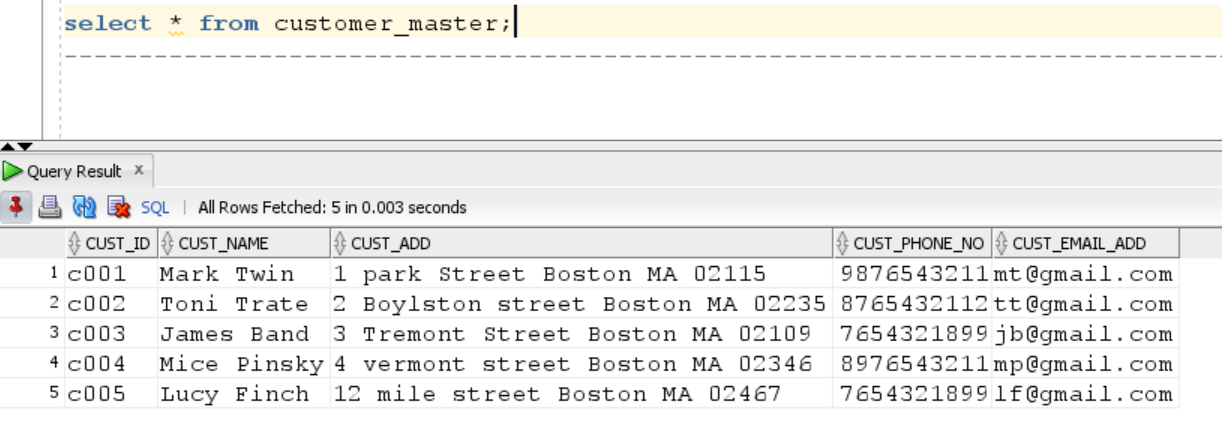
Columns that are considered neither primary nor foreign key columns must be evaluated on a case‐by case basis according to more complex criteria. Indexing is helpful to improve database performance.

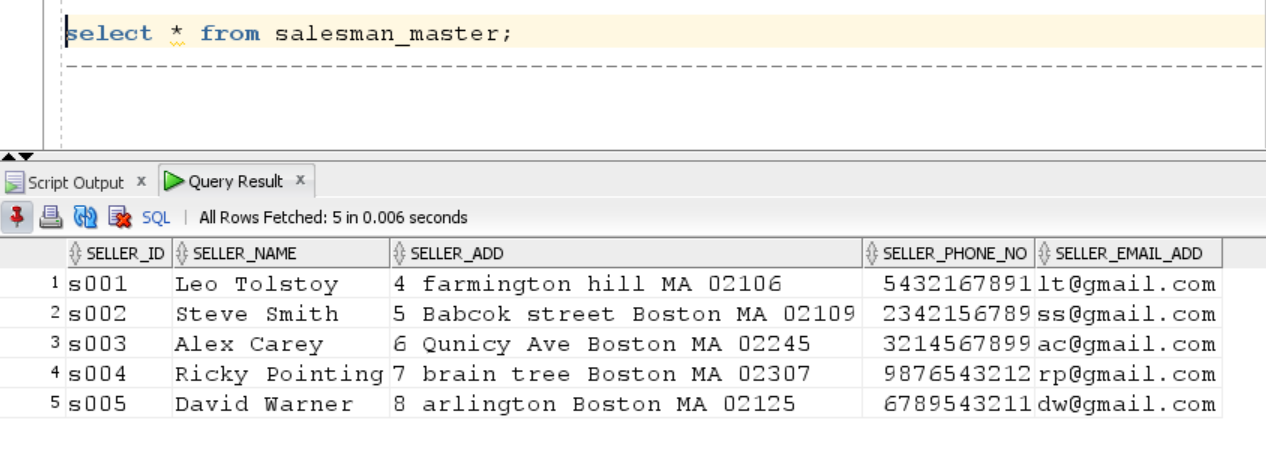
* Product\_master.product\_name :-This column will be frequently used to get Customer wants to see filtered products on the basis of size, color, etc. Product name is not unique.Same product sell by multiple sellers.
* Sales\_order.order\_status :- This column will be very frequently used to get a order status of the product whether it is in process, cancelled, delayed. This is not unique.There can be more than one product with same status name.

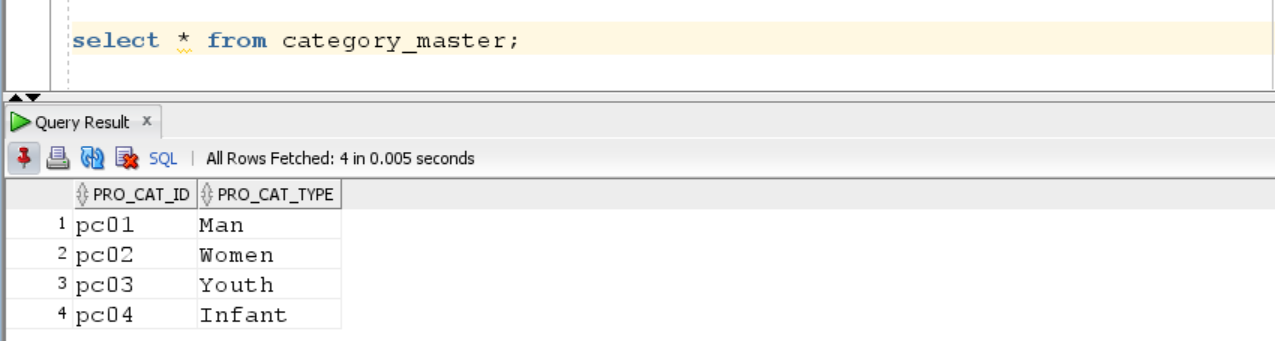


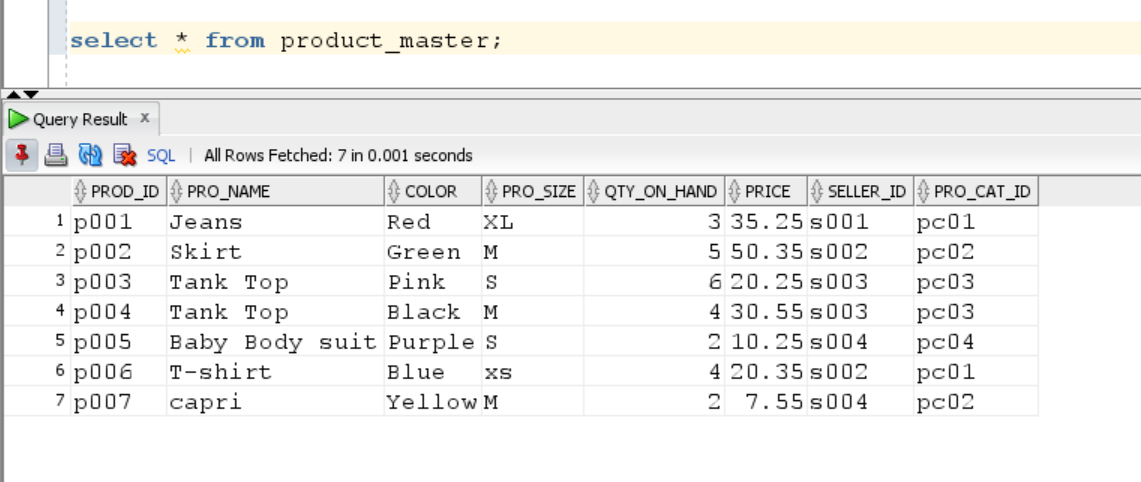
* The main entities of my database; customer\_master, salesman\_master, product\_master, sales\_order, sales\_order\_details, payment\_info,product\_price\_history
* Now, I have my entities identified with primary keys, foreign keys and few attributes. Also, I have my relationships defined between entities. Now this design can be used for my further iterations.
* All the attributes for each entity are identified and the final DBMS Physical ERD for my design is now complete. My tables are not having any partial or transitive dependencies. Verified if my tables are in normalized form and since it was already in 3NF, I proceeded with next steps.
* The create table SQL scripts follows the specification from the DBMS physical ERD. Foreign Key indexes and some important query driven indexes have been created to help speed up access the database.

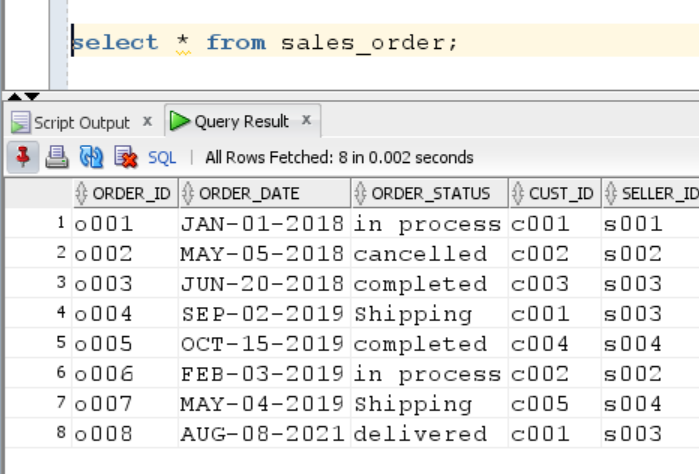
**13 ) INSERTED DATA ON FASHION ANYTIME TABLES**

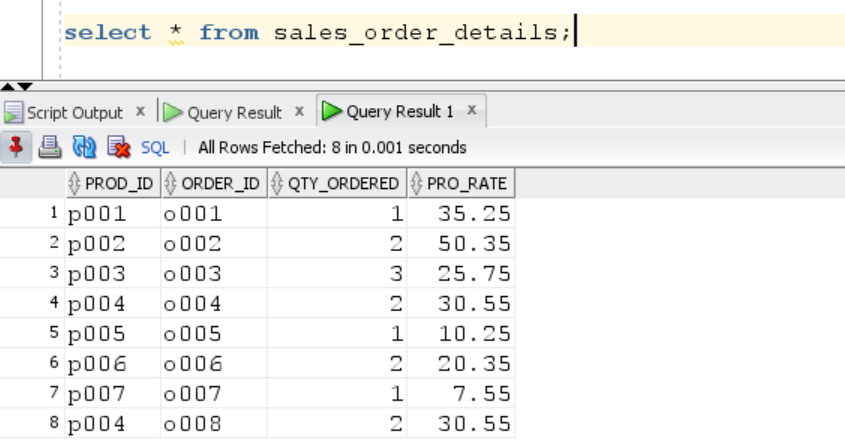
****

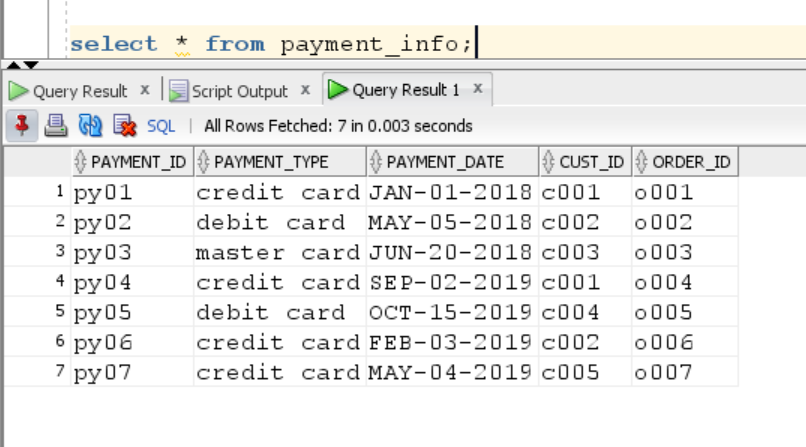
****

****





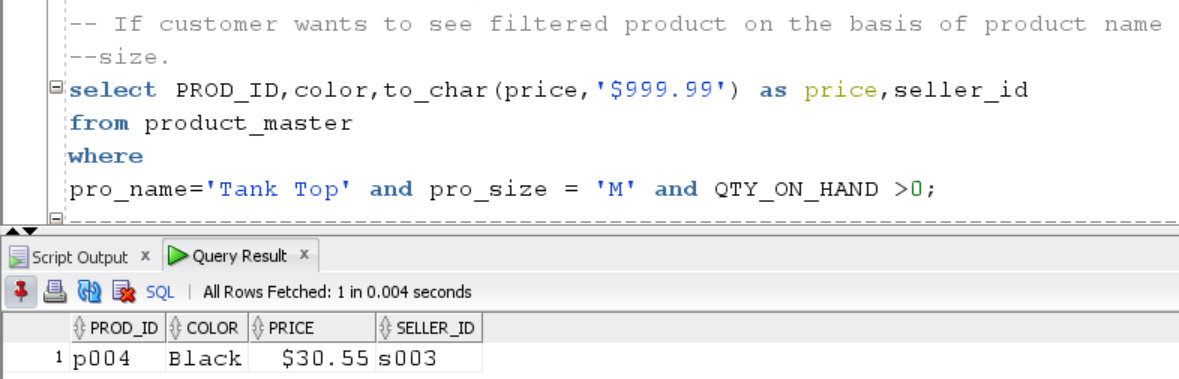


****

**14) Fashion anytime questions and queries**

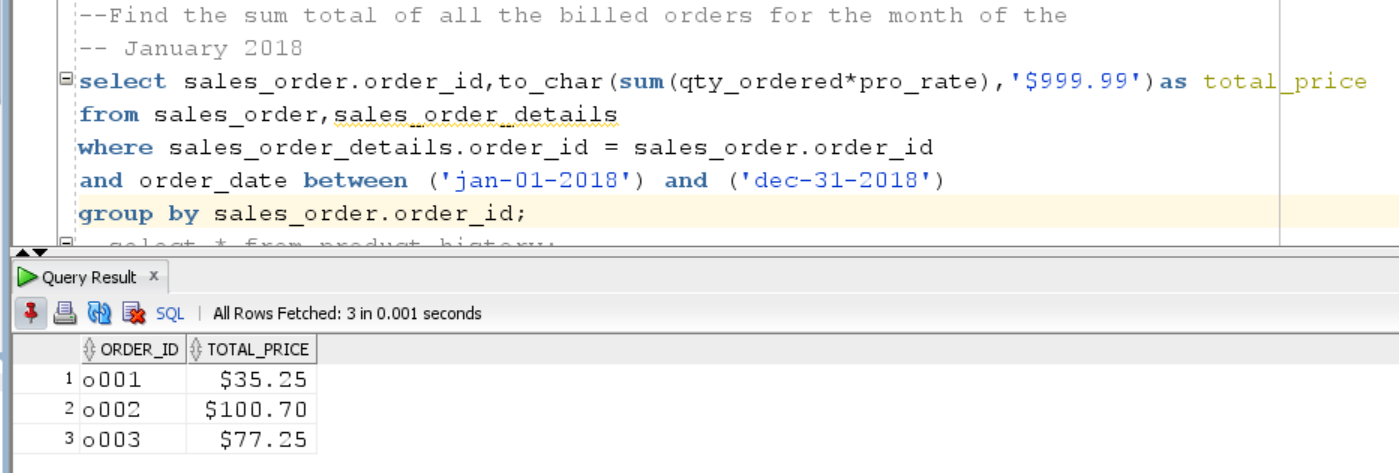
When customer wants to search product with different filter option and remove the item from the cart. Admin can view check the total of all the billed orders for the month of the January 2018.Admin can check respected order is placed by any customer or not. When user placed order over 35$ they will be eligible for free shipping. Once order in shipping stage user can not cancel it. Below queries supports the above questions.

**Query 1)If customer wants to see filtered product on the basis of product name, size.**

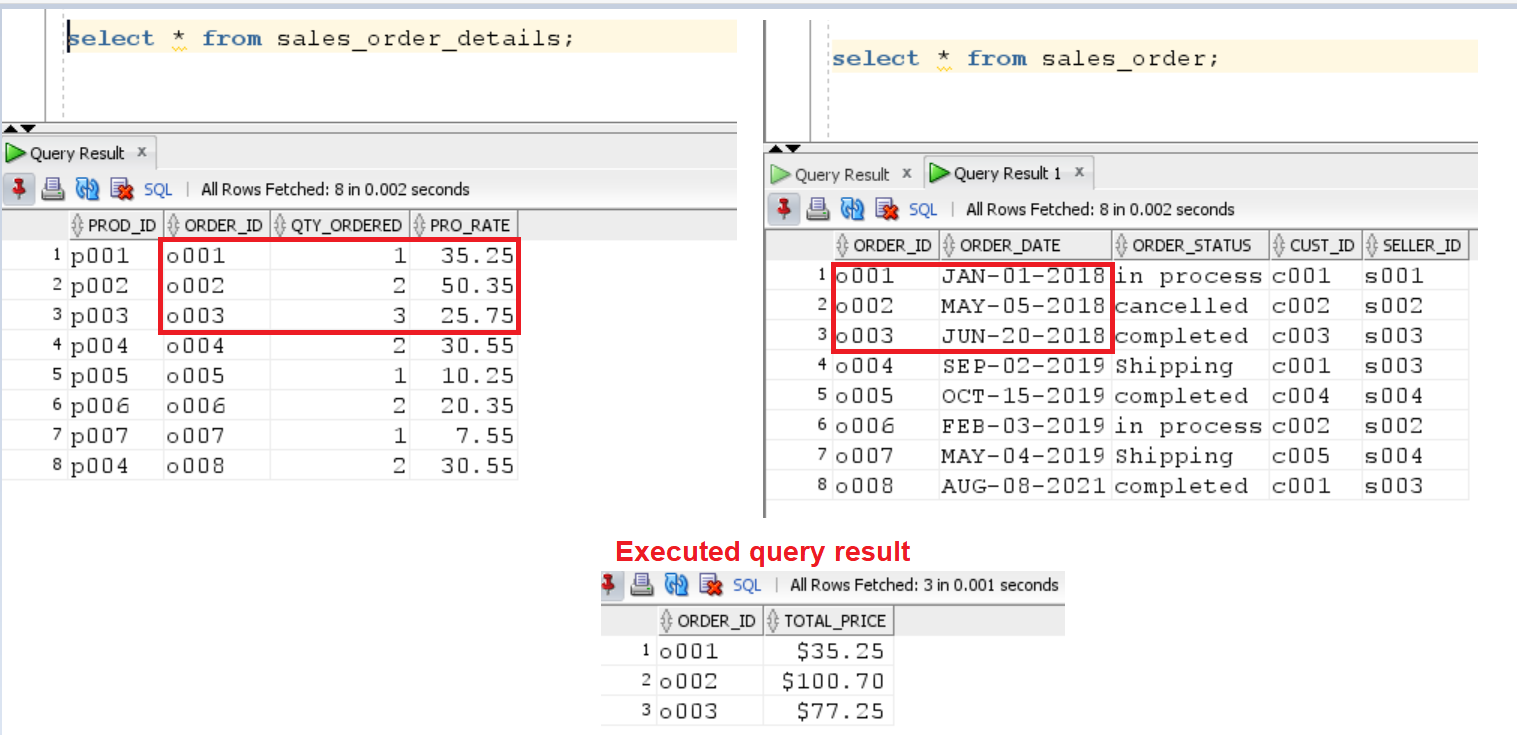


**Group 1 Query As per iteration-5**

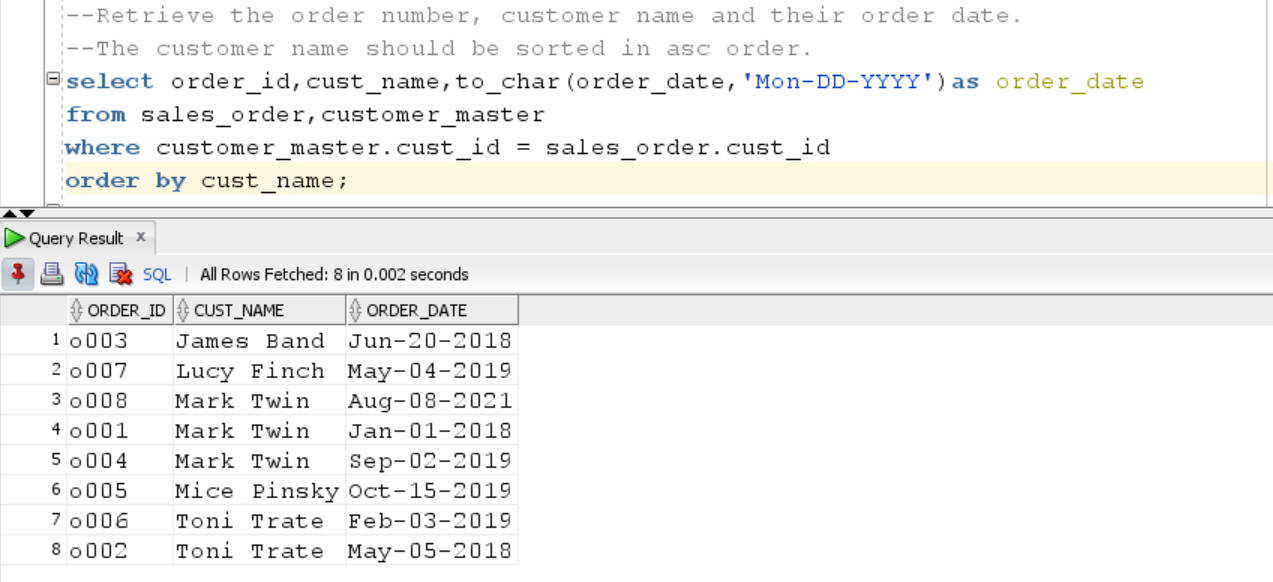
**Query 2) If admin want to view sum total of all the billed orders for the month of the January 2018 .**

****

**Below is additional screen-shots of the verified result.**

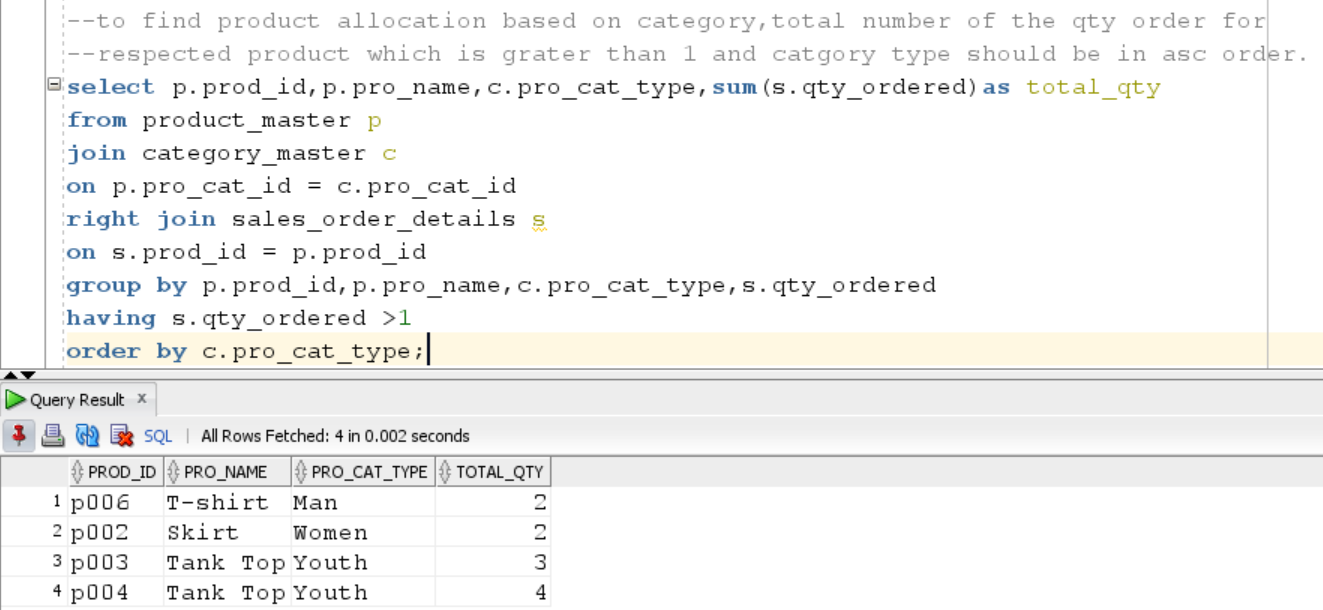
****

**Query 3)Admin want to Retrieve the order number, customer name and their order date. The customer name should be sorted in asc order.**

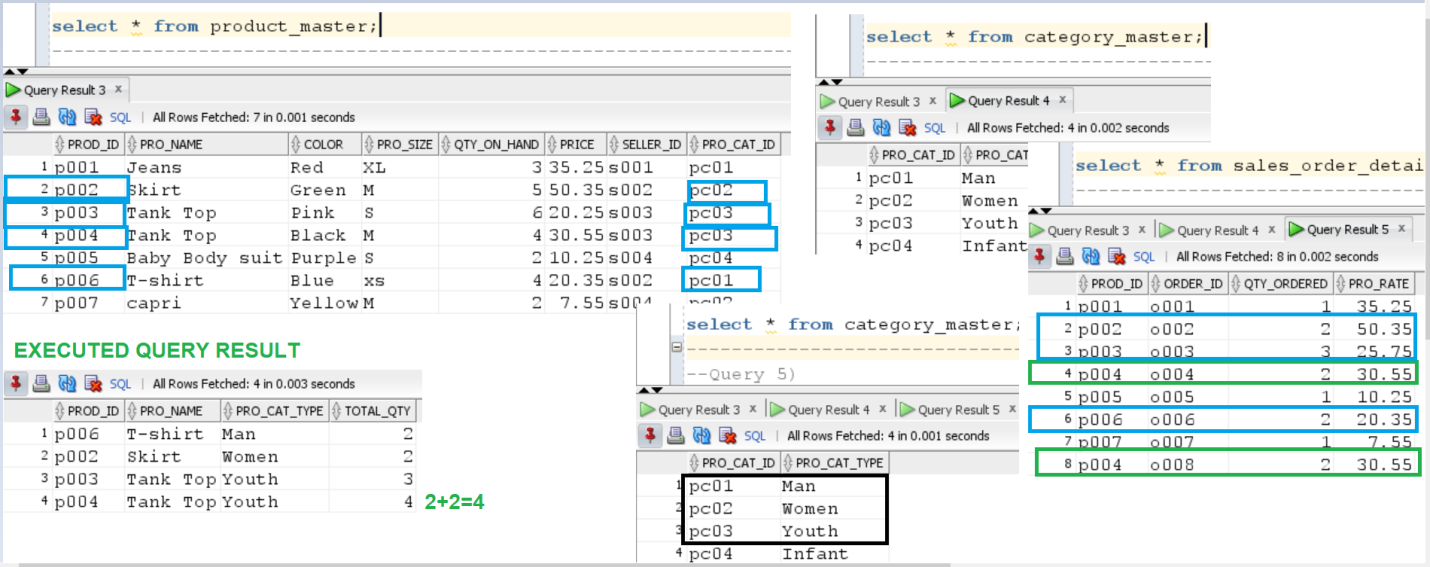
****

**Group 2 Query As per iteration-5**

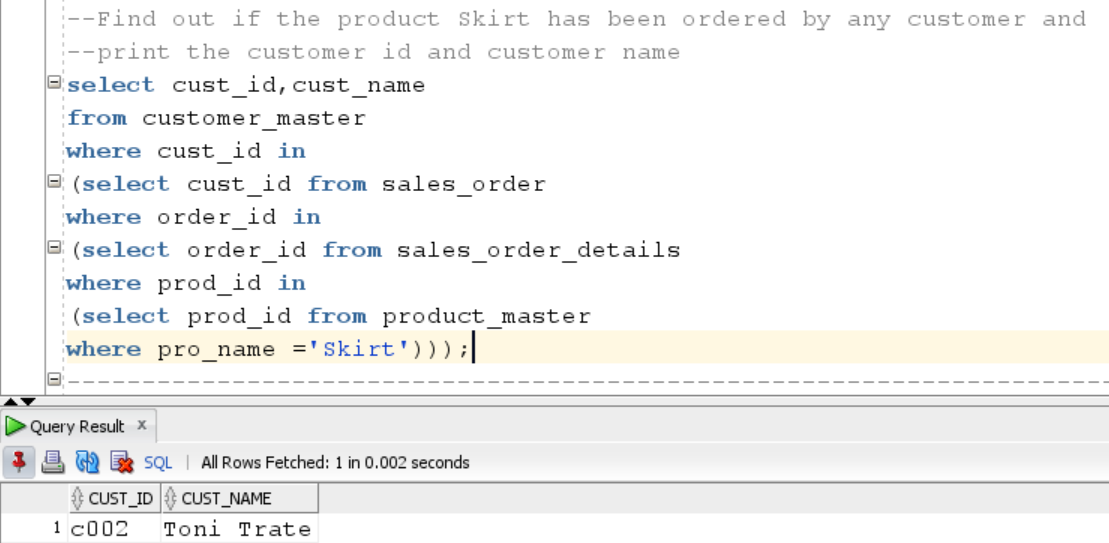
**Query 4) To find out product and their category type. Display category type in asc order and total number of the qty is ordered for respected product.**

****

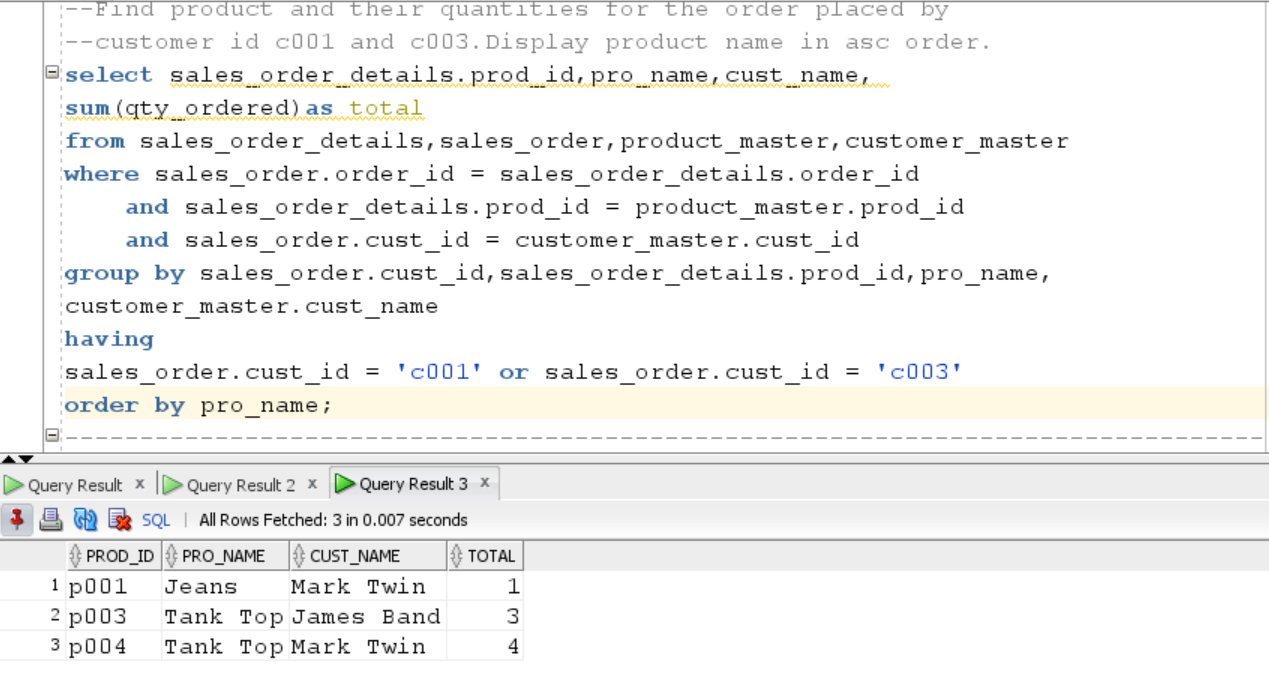
**Additional screen-shots of verified above query result**

****

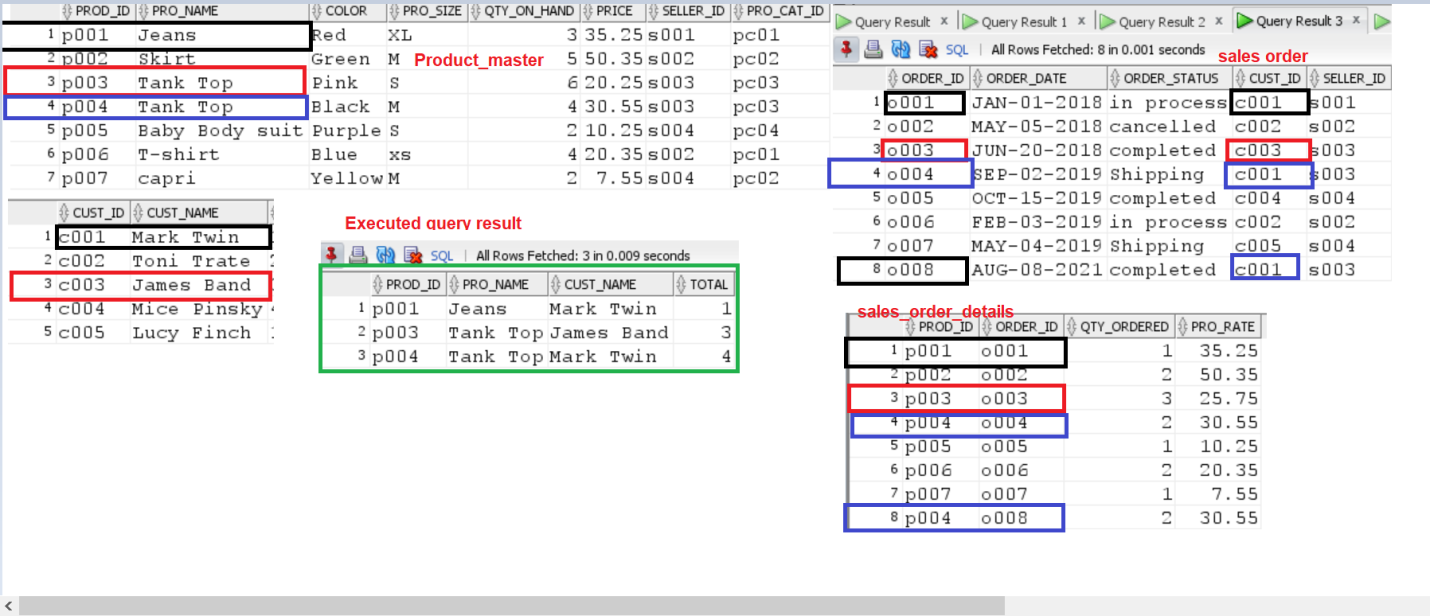
**Query 5)**  Find out if the product skirt has been ordered by any customer and print the customer id and customer name. (SUB QUERY)



**Query 6) Find product and their quantities for the order placed by customer id c001 and c003 and product name should be display in the ascending order.**

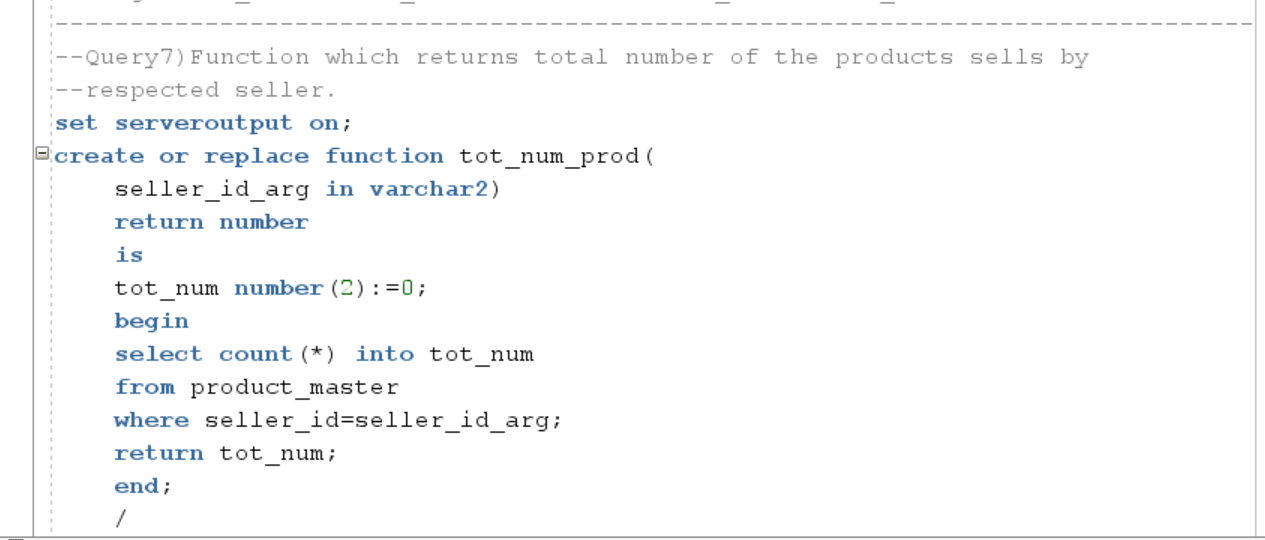
****

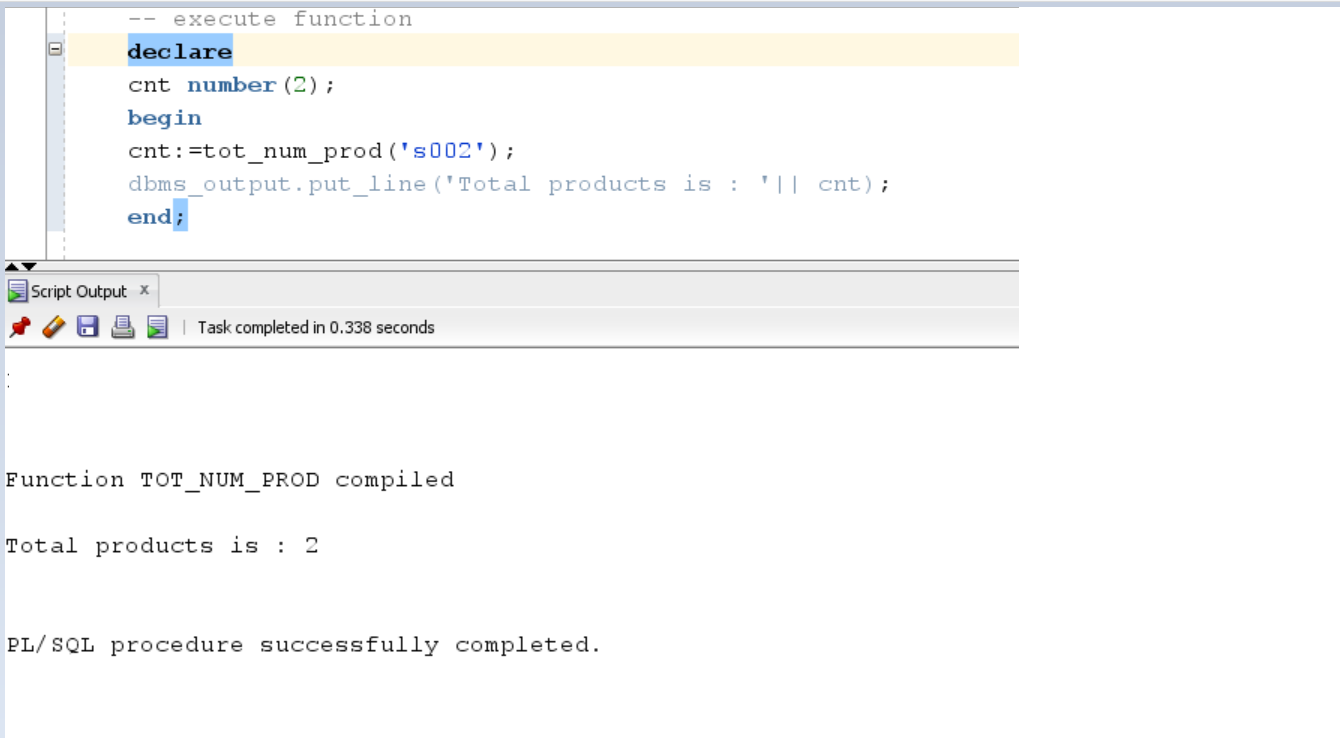
**Additional screen-shots of verified above query result**

****

**15) Function, Store procedures**

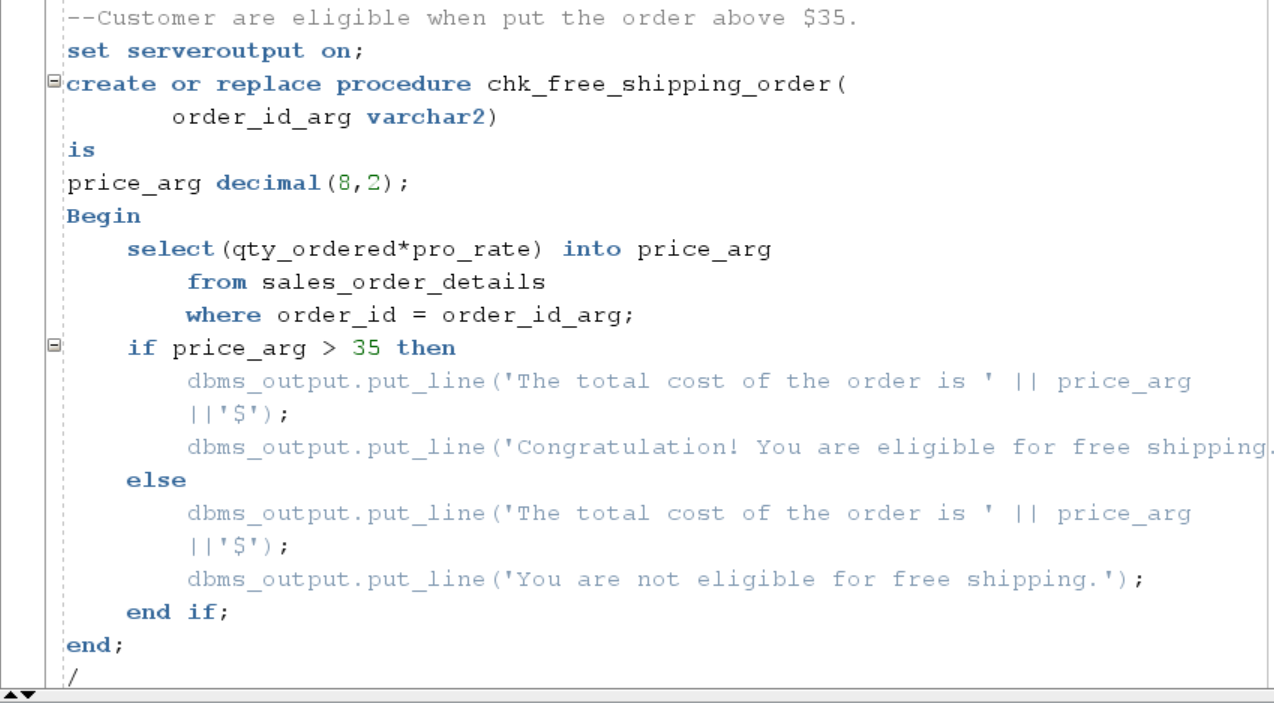
**Query 7) Tot\_num \_prod function :-** It is used to check total number of the products return by the respected sellers.

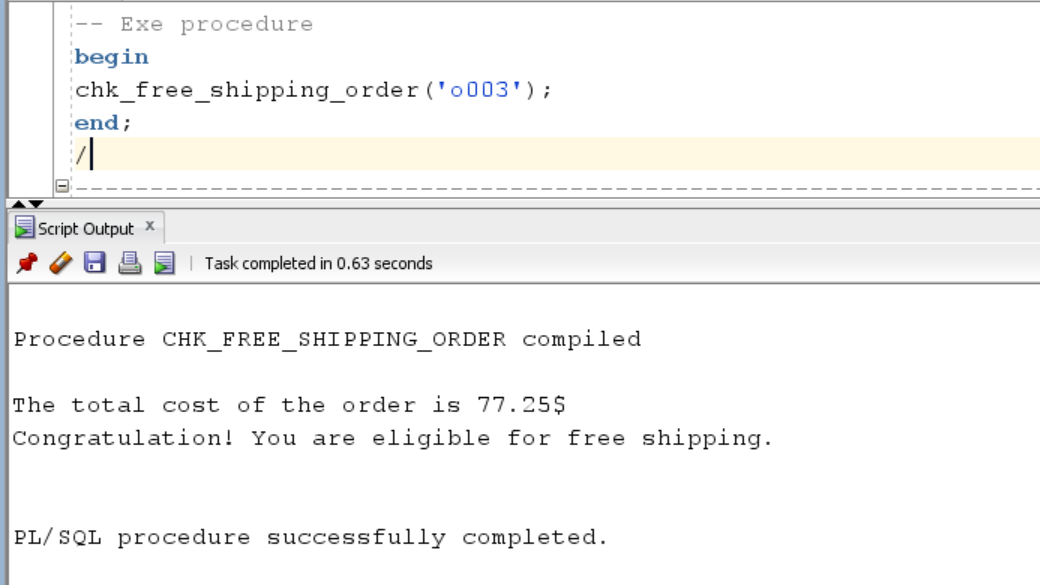
****

****

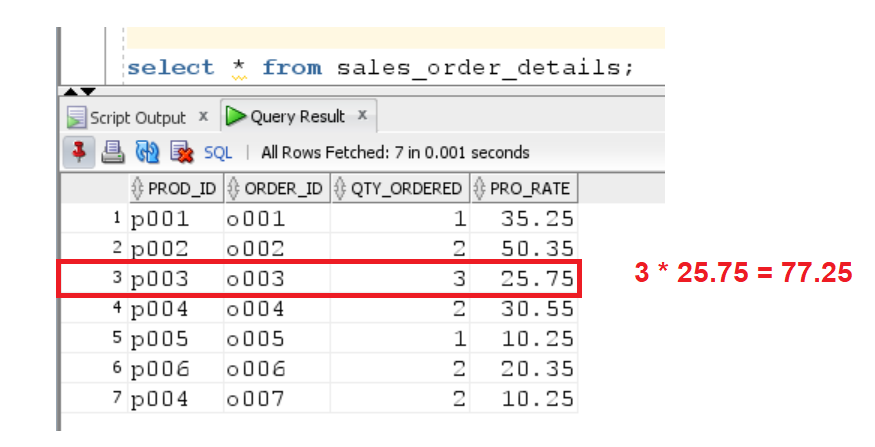
**Query 8) Check free shipping order procedure**

As per the business rule when customer placed the order above 35$ they are eligible for frees shipping. This procedure helpful to check whether respected order is eligible for free shipping or not.

****

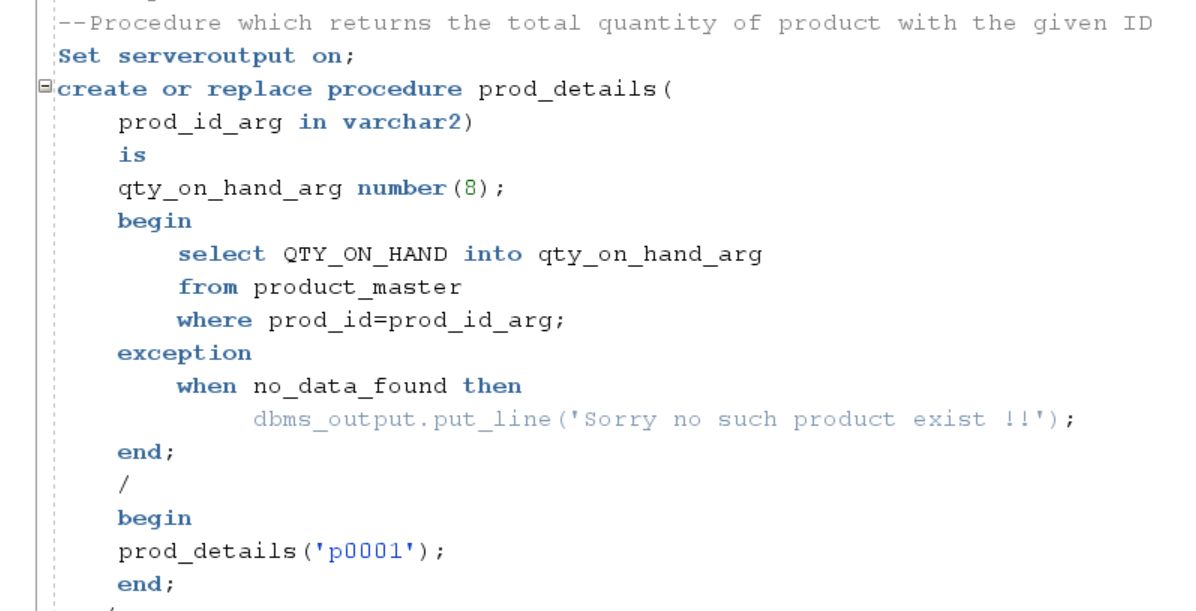


**Verify cost of the order with sales\_order details tables**

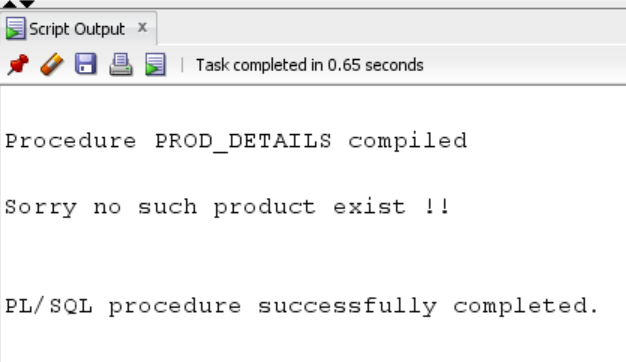
****

**Query 9)**

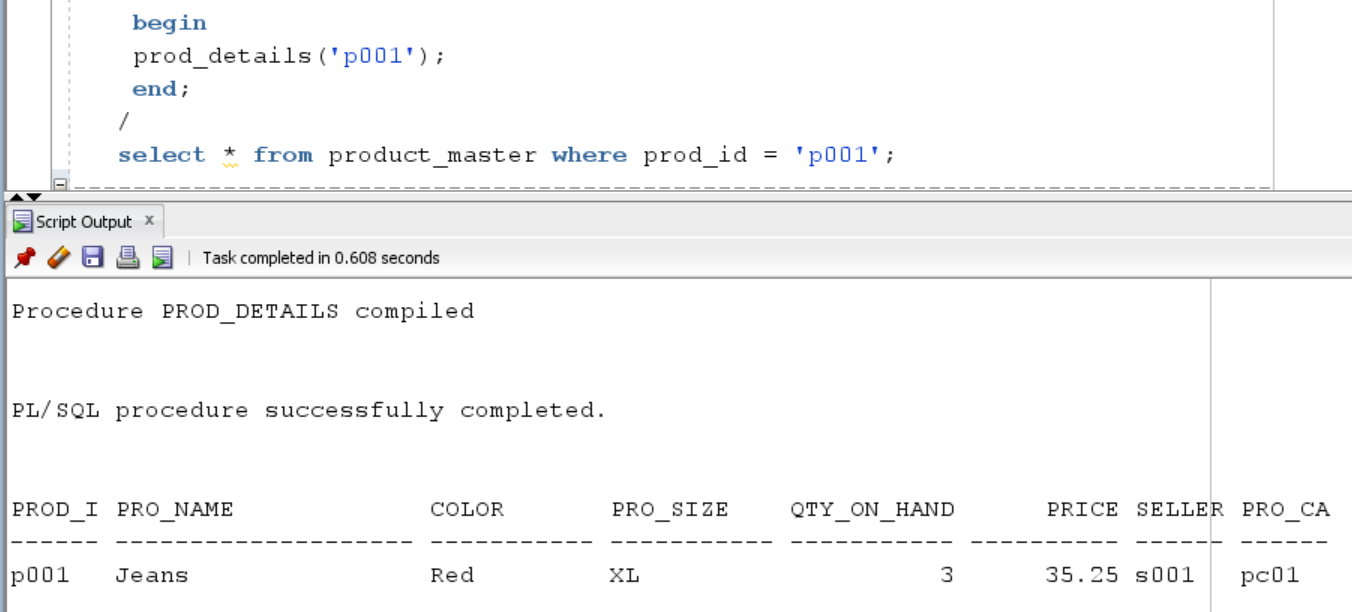
**Prod\_details procedure : -** It is used to check the total quantity of product with the given ID



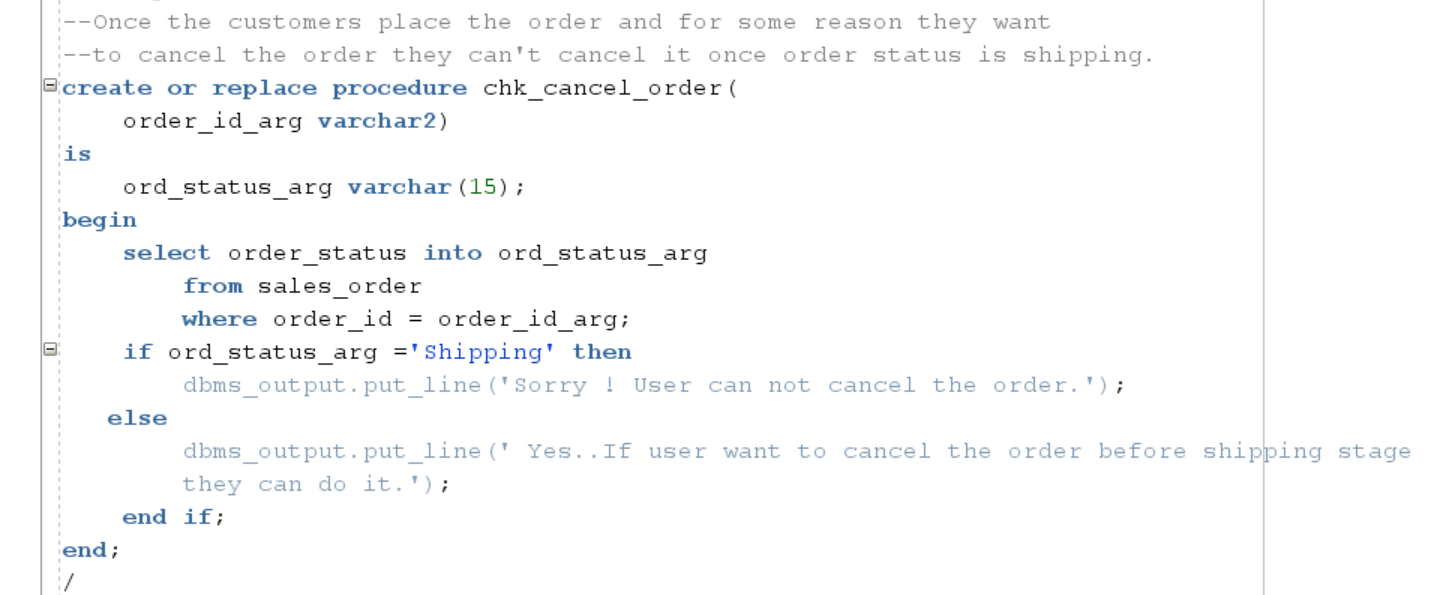
**When data does not exist in the table it throws the exception verify with passing product\_id which is not available in the database.**

****

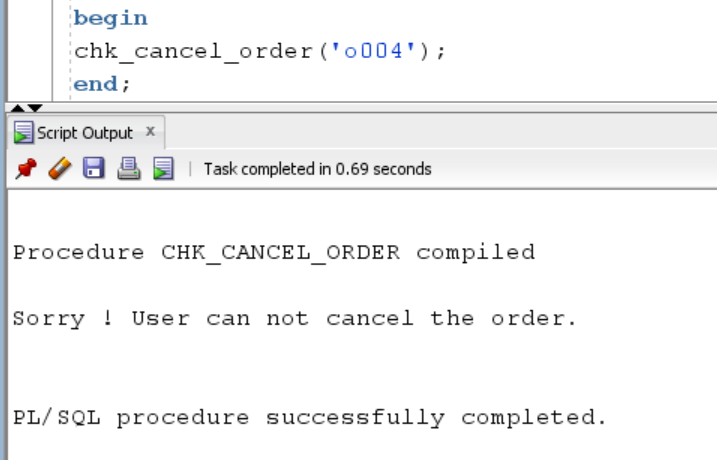
**When data exist in the table it returns the record.**

****

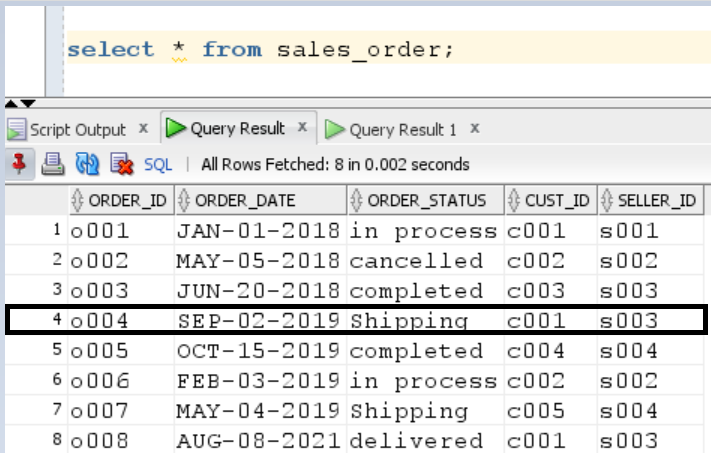
**Query 10) Chk\_cancel\_order :-** As per the business rule once order in the shipping stage customer can’t cancel the order. This procedure is used to check once the order status is shipping then it does not allow to user to cancel the order.

****

**Execute procedure**

****

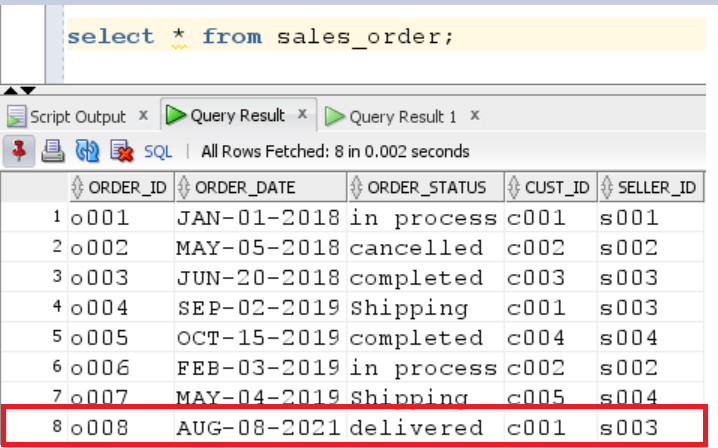
**Verify with sales\_order table where order\_id is o004 and status is shipping so can’t cancel the order.**

****

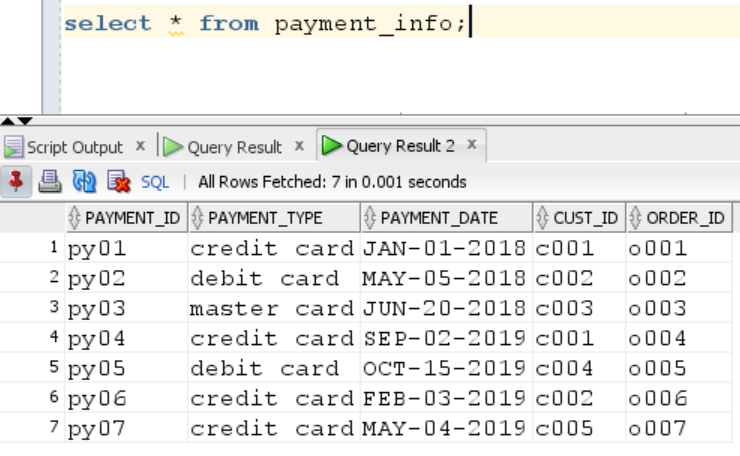
**16) FASHION ANYTIME DATABASE TRIGGERS**

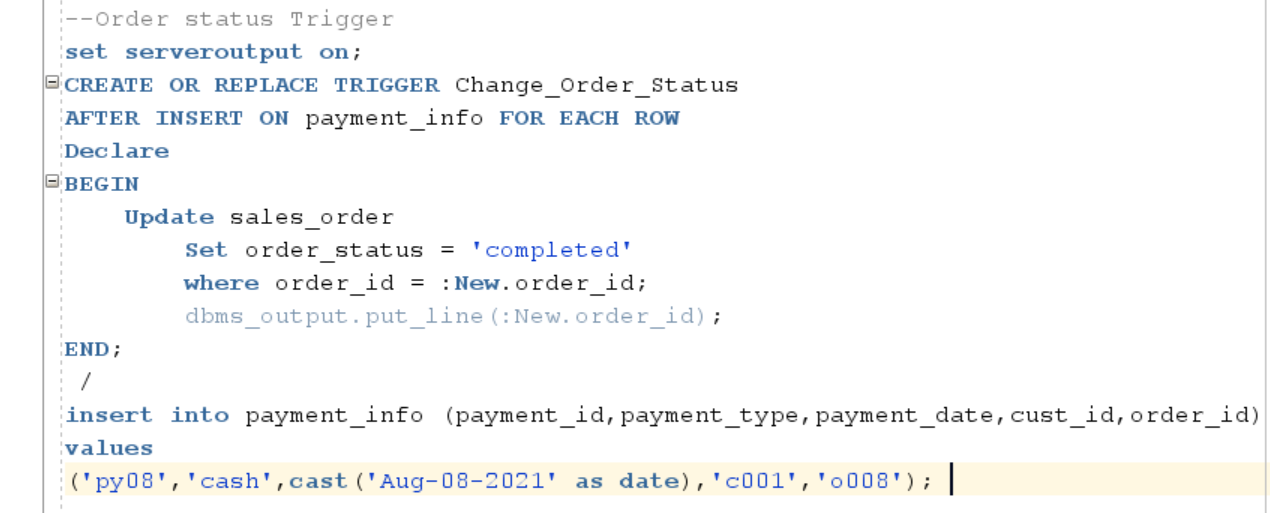
**Change\_Order status Trigger:-** When user perform payment activity for respected order status is automatically change into shipping to completed. Assume that payment type is cash-on delivery.

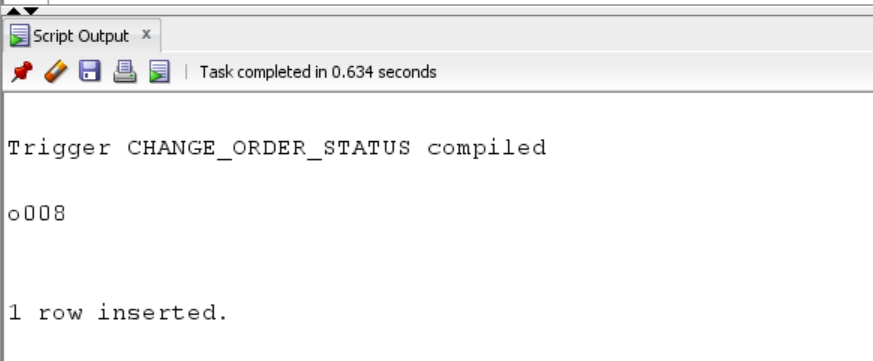
**Screen-shots before trigger is executed.**

****

**Order id ‘o008’ highlighted with red color status is delivered when trigger is executed and insert payment query is fired order status is automatically completed. ( CASH ON DELIVERY )**

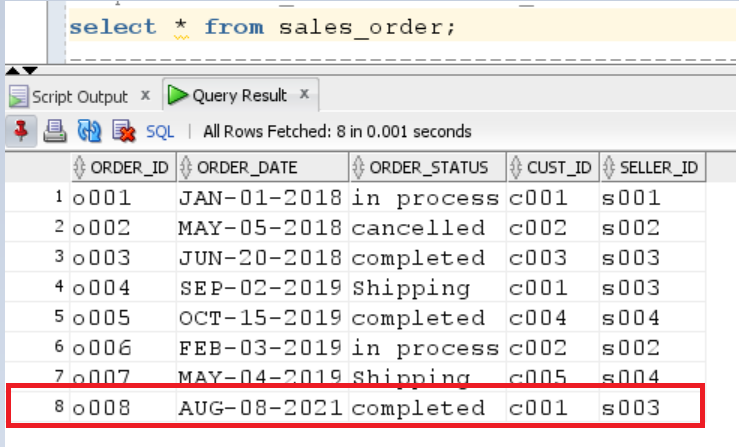
****

****

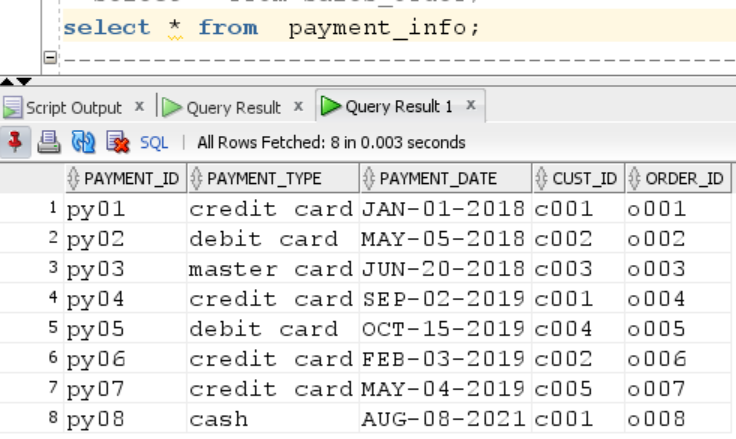
****

**Screen-shots after trigger is executed**

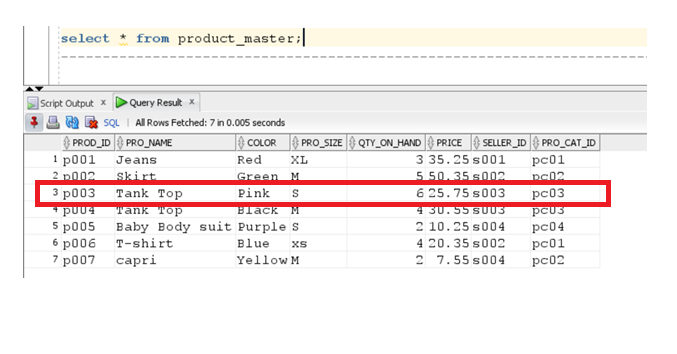
As we show before trigger is executed order id ‘o008’ has an order status is delivered once trigger is fired status is automatically completed.

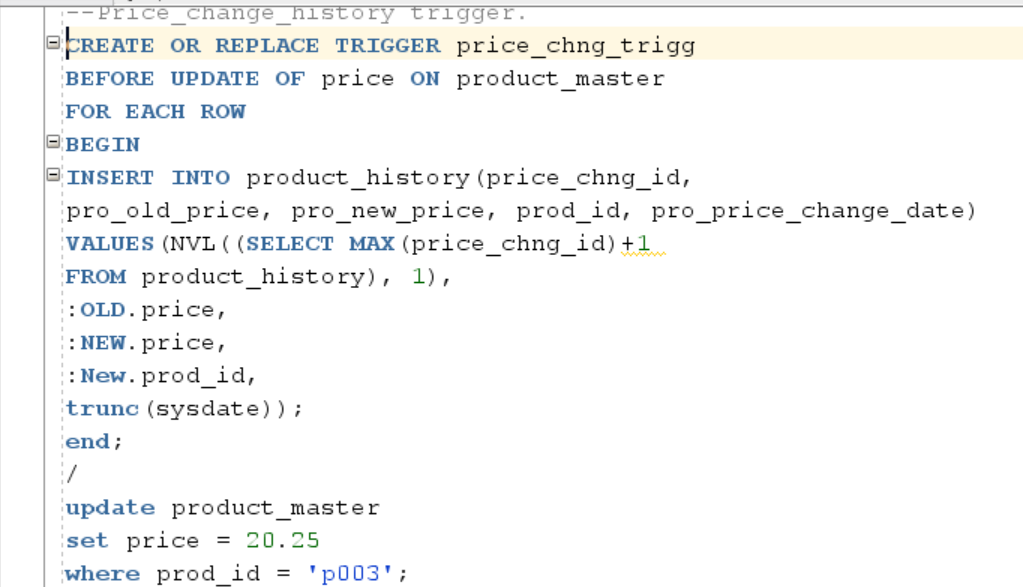
****

**When order status trigger is fired insert payment data (payment\_id ‘py08’) is inserted inside payment\_info table.**

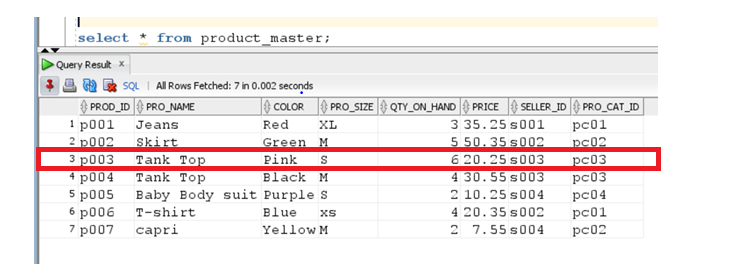
****

**17) Maintain history tables with triggers.**

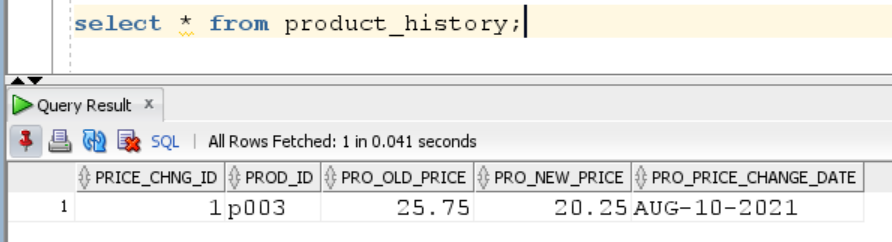
**Query12) Price\_chngg\_trigg :** - When seller change the product’s price. Then history table is created to track of the respected product’s old and new price with effective date.

****

**After trigger is executed verify product id p003 price is change in product\_master table.**

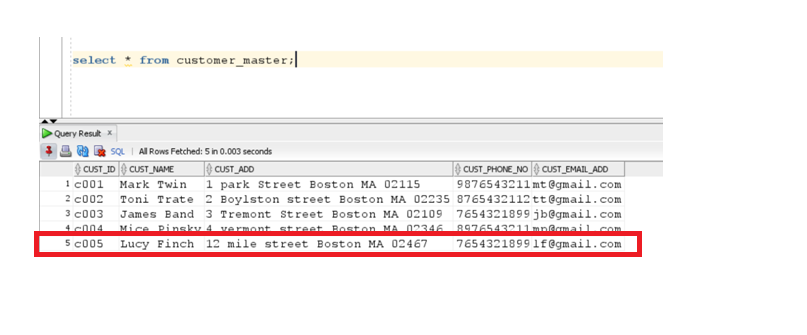
****

**Inside the product\_history table you can view old as well as new price of the product id p003 with effective price\_change\_date.**

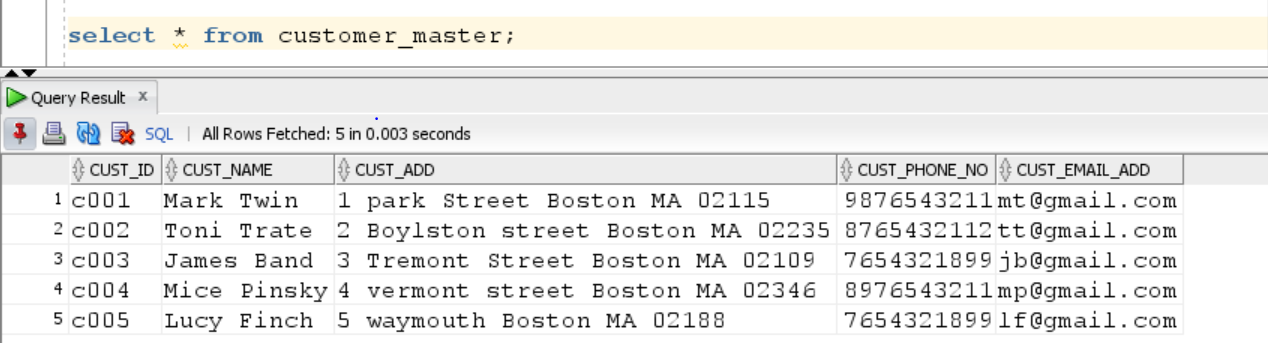


**Query13)**

**Cm\_add\_history\_tirg:- When some customer’s change their respected address this trigger is used to track of the customer’s old and new address. Sometimes admin send some special offers and gift card to respected customer’s address.**

**Before trigger executed record c005 has a 12 mile…address  **

**After trigger is executed record c005 address is changed and that reflection can be visible on customer\_master table.**

****

**Customer\_master\_history table shows the respected customer’s old as well as new address.**

****

**Summary and reflections**

Following are the major usages will be supported by the database:

1. A new user can register.
2. A customer can filter the product based on product details.
3. If admin want to view sum total of all the billed orders for the month.
4. Admin can find out respected product has been ordered by any customer or not.
5. Admin can check the total number of the quantities placed by respected client.
6. Admin can view product allocation based on category and total number of the qty ordered for respected product.
7. Admin can view total number of the products return by the respected sellers.
8. Retrieve the order number, customer name and their order date.
9. Application satisfy business rule whether customer are eligible for free shipping or not. It is achieved with the help of store the procedure.
10. It checks the total quantity of the given product id
11. Application satisfies business rule once order is in shipping stage can not be cancel.

It is also achieve with the help of store procedure.

1. When seller change the respected product's price admin can view old and new price history.
2. If admin want to send gift card to respected customer they can sent it on his address

When customers change their address admin can view their new address in the system.

* The structural rules and the conceptual ERD covers the main entities of my database; customer\_master, salesman\_master, product\_master, sales\_order, sales\_order\_details, payment\_info,product\_price\_history,customer\_master\_history and the relationship between them.
* The design reflects the hierarchy of category of product – man, women, youth,infant.The design also captures a hierarchy of online shopping .
* Entities identified with primary keys, foreign keys and few attributes. Relationships defined between entities follow the business rules.
* The final DBMS Physical ERD for my design is completed with the same entities and relationships, uses the best practice of synthetic keys, and contains the important attributes needed by the database to support the application.
* My tables are not having any partial or transitive dependencies. Verified if my tables are in normalized form and since it was already in 3NF, designing does not have Normalization steps.
* The create table SQL scripts follows the specification from the DBMS physical ERD exactly. Foreign Key indexes and some important query driven indexes have been created to speed up accessing the database.
* Stored procedures have been created and executed transitionally to populate some of my database with data.