
# IE2042 - Database Management Systems for Security

Assignment 01: 2020 Regular Intake

SUPERMARKET MANAGEMENT SYSTEM

Group Members:


| IT Number | Name |
|---|---|
| IT19003610 | SHIRANTHAKA K.G.S. |
| IT19013756 | JAYASINGHE M.H.D.V. |
| IT19007366 | FERNANDO M.S.H. |
| IT19007984 | KARANDENIYA K.I. |

# Contribution for the Project

| IT Number | Contribution |
|---|---|
| IT19003610 | • Create database tables table and partial contribution for inserting data<br>• Creating an admin user of the system<br>• Create view employee_details<br>• Partial contribution prepare countermeasures for web attacks<br>• Create order_item trigger<br>• Create update_item procedure<br>• Partial contribution to preparing additional security implementation |
| IT19013756 | • Draw ER diagram<br>• Creating role manage and adding user.<br>• Create view supplier_details<br>• Partial contribution prepare countermeasures for web attacks<br>• Create login logoff trigger<br>• Create Crt_cshr_amt and item_in_order procedures<br>• Partial contribution to prepare backup and recovery mechanisms |
| IT19007366 | • Mapping the table and draw a relational schema<br>• Creating a role storekeeper and adding the user.<br>• Create view stock_details<br>• Partial contribution for identifying the possible attacks and when database host in the web<br>• Create stock_price function<br>• Create update_stock procedure<br>• Partial contribution to preparing additional security implementation |

| | |
|---|---|
| IT19007984 | • Partial contribution for inserting data<br>• Creating role cashier and adding user<br>• Create view cashier_details<br>• Partial contribution for identifying the possible attacks and when database host in the web<br>• Create total_salary function<br>• Create totSalary procedure<br>• Partial contribution to prepare backup and recovery mechanisms |

# Acknowledgment

The success and outcome of this assignment required a lot of guidance and assistance from many people, and we extremely fortunate to have got this all along with the competence of our assignment work. Whatever we have done is only due to such guidance and assistance, and we would not forget to thank them. I respect and thank Ms. Tharika Munasinghe for providing us all support and advice, which made us complete the assignment on time, Also this assignment cannot be achieved without the effort and co-operation from our group members, and we would like to express our gratitude to who everyone supported us.

# Table of Contents


Contribution for the Project ..... 1
Acknowledgment ..... 3
Abstract ..... 6
1.Introduction ..... 7
2.Hypothetical Scenario ..... 8
3.Requirement Analysis ..... 9
3.1 ER-Diagram ..... 11
3.2 Relational Schema ..... 12
3.3 Screenshots of Table Creation Queries ..... 13
3.4 Screenshots of Tables after Inserting Data ..... 16
3.5 SQL Commands to create the tables in the database ..... 25
3.6 Commands to insert the data into the Database ..... 33
4.Implementing access control privileges ..... 39
4.1 Creating a role for the manager and adding the user ..... 39
4.2 Creating a role for store_keeper and adding the user ..... 41
4.3 Creating a role for cashier and adding the user ..... 42
5.Possible attacks and countermeasures when the database connect to a web application ..... 43
5.1 SQL Injection ..... 43
5.1.1 Countermeasures to prevent SQL injection ..... 43
5.2 Cross-Site Scripting ..... 43
5.2.1 Countermeasures to prevent XSS include: ..... 43
5.3 Denial of Service ..... 44
5.3.1 Countermeasures to prevent denial of service ..... 44
5.4 Password Cracking ..... 44
5.4.1 Countermeasures to help prevent password cracking ..... 44
5.5 Privilege Escalation ..... 45
5.5.1 Countermeasures to prevent privilege escalation ..... 45
6. Implementation of countermeasures methods and additional security implementations ..... 46
6.1 Views ..... 46
6.1.1 Create view employee_details ..... 46
6.1.2 Create view stock_details ..... 47
6.1.3 Create view supplier_details ..... 49
6.1.4 Create view cashier_details ..... 50
6.2 Triggers ..... 53
6.2.1 Create order_item_trigger ..... 53
6.2.2 Create login_trigger and logoff_trigger ..... 61

6.3 Functions ..... 65
6.3.1 Create total_salary function ..... 65
6.3.2 Create stock_price function ..... 67
6.4 Procedures ..... 70
6.4.1 Create Procedure totPrice ..... 70
6.4.2 Create Procedure Crt_cshr_amt ..... 73
6.4.3 Create Procedure item_in_order ..... 77
6.4.4 Create Procedure update_stock ..... 80
6.4.5 Create Procedure update_item ..... 85
6.5 Additional Security Implementations ..... 90
6.5.1 Enabling AUDIT_TRAIL and setting it up with distinct privileges ..... 90
6.5.2 Create a security profile and lock the profile after failing to login with password ..... 91
6.5.3 Create a database audit event to detect all the user's logon and logoff details. ..... 95
7. Identified transaction in the supermarket management system ..... 98
8.Explain about recovery mechanisms of the database ..... 100
9. Conclusion ..... 106

# Abstract

In these days' supermarkets, customer's usage is increasing daily. We have designed this system to reduce the problems and to improve the efficiency of supermarket management. In this system will perform all aspects of the supermarket management. (Employees and customer data managing, stock and order details managing)**.**

This report analyses all the database tables, SQL commands, transactions, vulnerabilities, and countermeasures in the system. Also, how grants privileges and end of the report shows an audit trail in the system.

# 1.Introduction

The supermarket management system is a system that handles all facets of proper supermarket management. These factors include knowledge processing with specific items, personnel, administrators, clients, accounting, etc. The program ensures that supermarket knowledge is handled effectively. The customer will also order and pay for the bought products.
Since society started, people have been engaged in trade. Next, the bartering device was used to exchange goods and services. Coins and paper notes have been developed over time, and the currency system has been set up. Trade is becoming an essential part of and guarantees the survival of modern society. Today, the industry has changed in many ways, with supermarkets playing a significant role in daily life. And a broad self-service shop that sells food and other consumer goods is the supermarket at its core. The rapid population growth and consumerism have resulted in increased demand for a broader range and increased quantity of products. This is why supermarkets have become essential in their performance. Luckily, technical development has improved the situation, in particular information systems, which can process vast quantities of data at speed, higher precision, reducing costs, and better health. This helps supermarkets to conveniently satisfy all consumer requirements and, at the same time achieving business objectives. And this report describes the database management of such a supermarket system.

This project is focused on revenue and paying in a supermarket. The first operation is the introduction of the items to the list along with the pricing on the marketplace and the name of the goods to be offered by the marketplace. Only admin (administrator) is given this power. Any changes to the name and cost of the element can only be made by the administrative owner. He is also entitled to delete any object. All the privileges are given to the admin, and he is the one who creates users and assigns the user to a relevant job role.

# 2.Hypothetical Scenario

The supermarket management system is created to make work more comfortable and more efficient for its management. In this system, there are few leading roles, and they categorized as Administrator, Employees (Manager, Storekeeper, and cashier). In this system CEO of the organization act as administrator. An administrator has high-level privileges in this system. Suppliers and customers are the outsiders in the system. The administrator can manage all the employees through this system, and the Manager can manage Storekeeper and cashier according to his privileges. Storekeeper can manage the store, and Cashier can manage cashier machines and orders. All these users can access the system using login. The only administrator has the privilege to create user accounts. When the administrator required to create the salary report, then he/she can create it through the system. The manager can update all the detailed reports except for the salary report. Storekeeper can delete, insert, update all store details, and he/she connects directly with stock. When supplier supplies items stock details will update through the system.

# 3.Requirement Analysis

For some instances, it is disadvantageous to operate a supermarket system manually. There are some problems like customers will also have to stay for a long time in queues before their goods are paid, it is difficult to manage the staff and stock details, and there is a very high scope for error. This can be removed by an information system like the one suggested in this paper. The consumers will be able to check their orders much more efficiently, and the system will be secure and confidential information storage much safer. And employees can handle their stocks quickly. Via review and reports the system development, informed business decisions such as the promotion of goods and customer loyalty programs can be made. Maintenance overall is also more straightforward in this system.

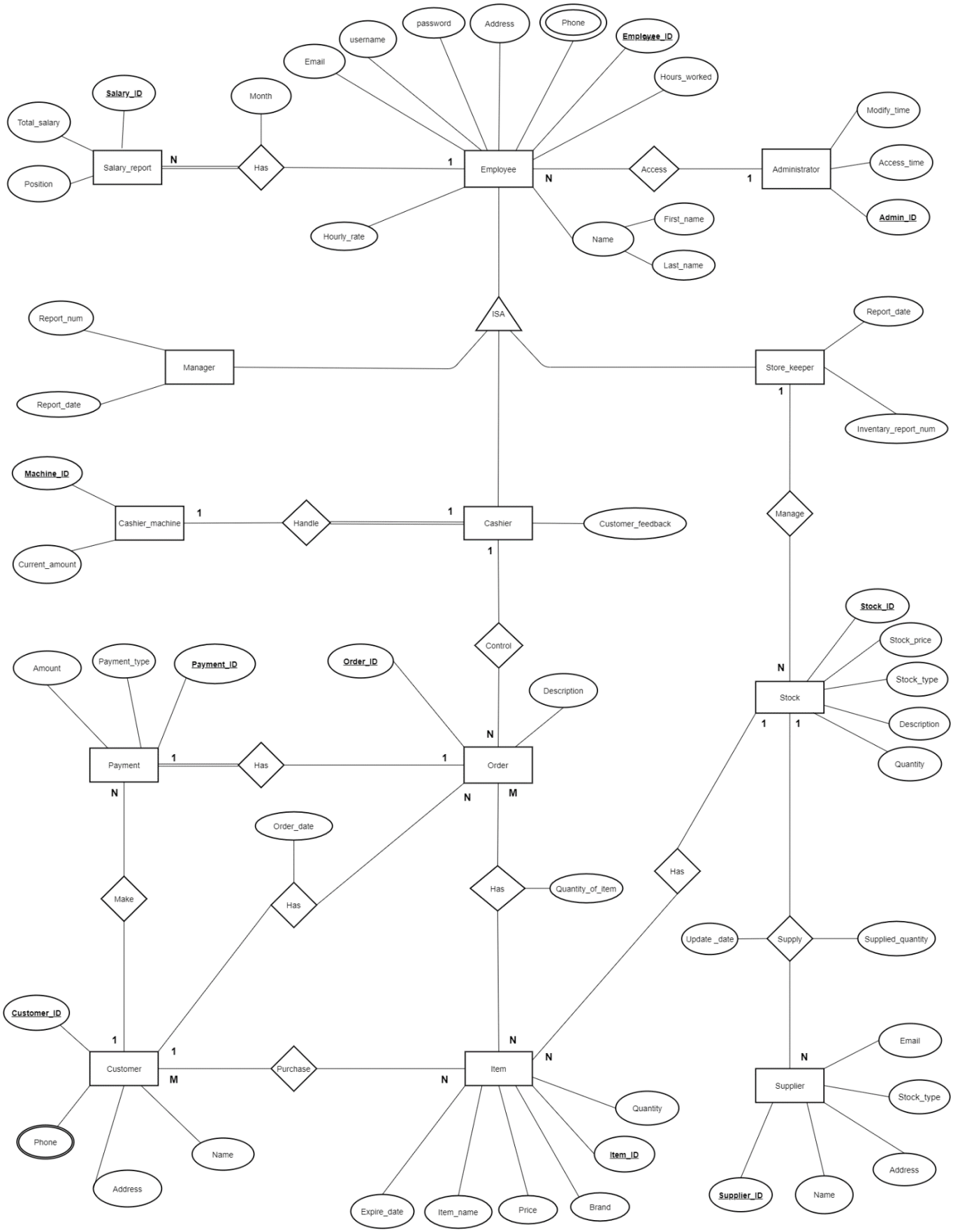
Functional Requirements

1. Able to manage employee details
    - The system should able to do these operations INSERT, UPDATE, DELETE to employee details in the database. And the system stored the salary information of employees in the database.
2. Able to manage stock details
    - The system should able to do these operations INSERT, UPDATE, DELETE to stock details & item details in the database. And the system must provide some procedures to update the stock.
3. Able to manage payment & order details
    - The system should able to do these operations INSERT, UPDATE, DELETE to payment details & order details in the database. And the system must provide some procedures to update the payments.
4. Able to manage aspects of outers of organization
    - The system should able to do these operations INSERT, UPDATE, DELETE to customer details & supplier details in the database.
5. The system must able to log users to the system
    - The system should able to employees for logging to the system.

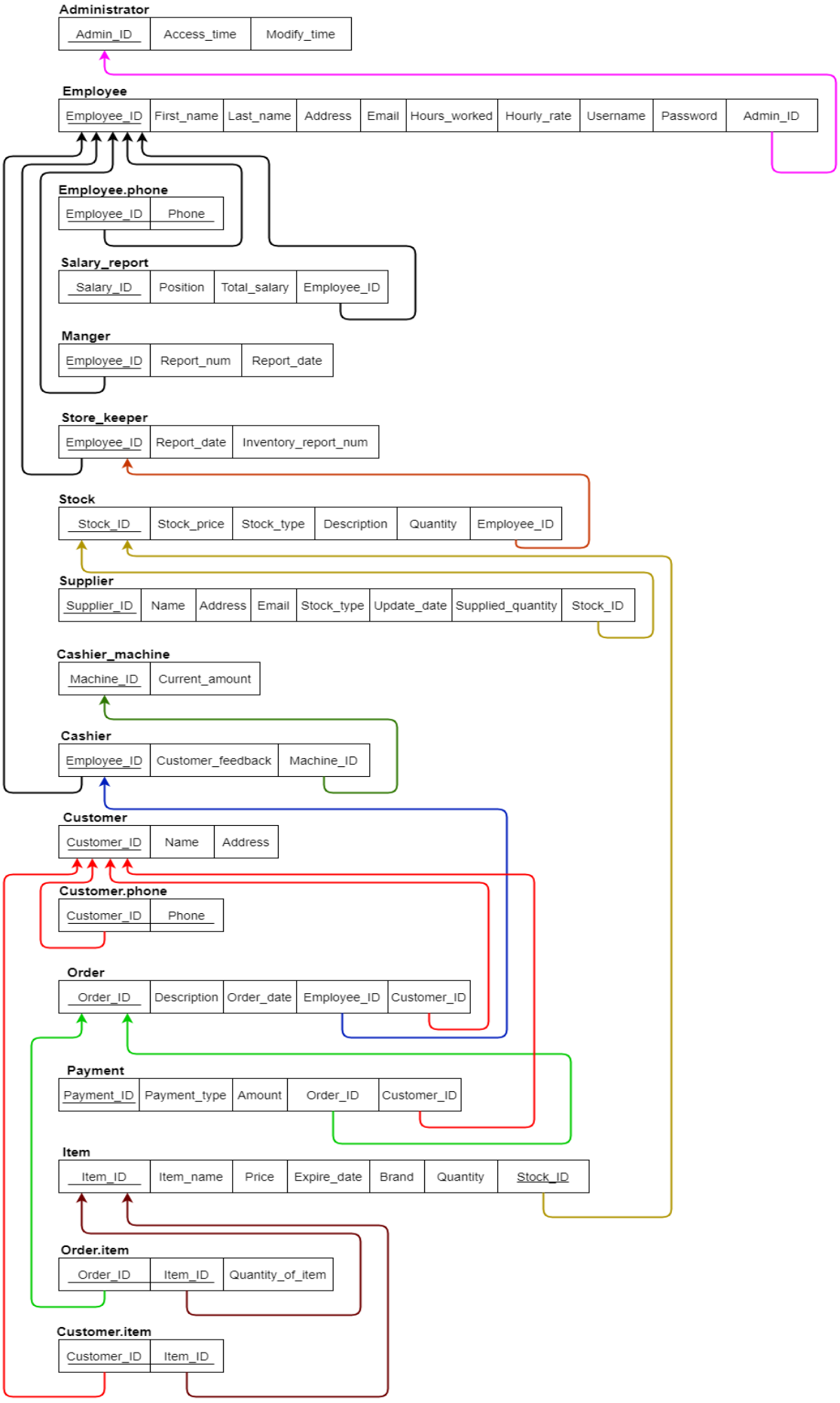
Non-functional Requirements

1. Security
    - The database contains sensitive information such as payment information and should provide safe access to these data. Every of the positions played by the program users will have specific permission rates and will, therefore, often be limited in their behavior within this program. And system able to record user access times.
2. Backup
    - A system able to keep backup of some tables in the database, and this backup modified automatically.
3. Friendly interface for users.
    - The system should have a simple graphical user interface, so that staff can learn how to use the program easily and effectively, regardless of their computer literacy. Depending on the user access levels, the device provides various views.
4. Scalability and reliability.
    - At all times, the system needs to be able to accommodate as many users as possible, and the system needs to respond accurately.

## 3.1 ER-Diagram

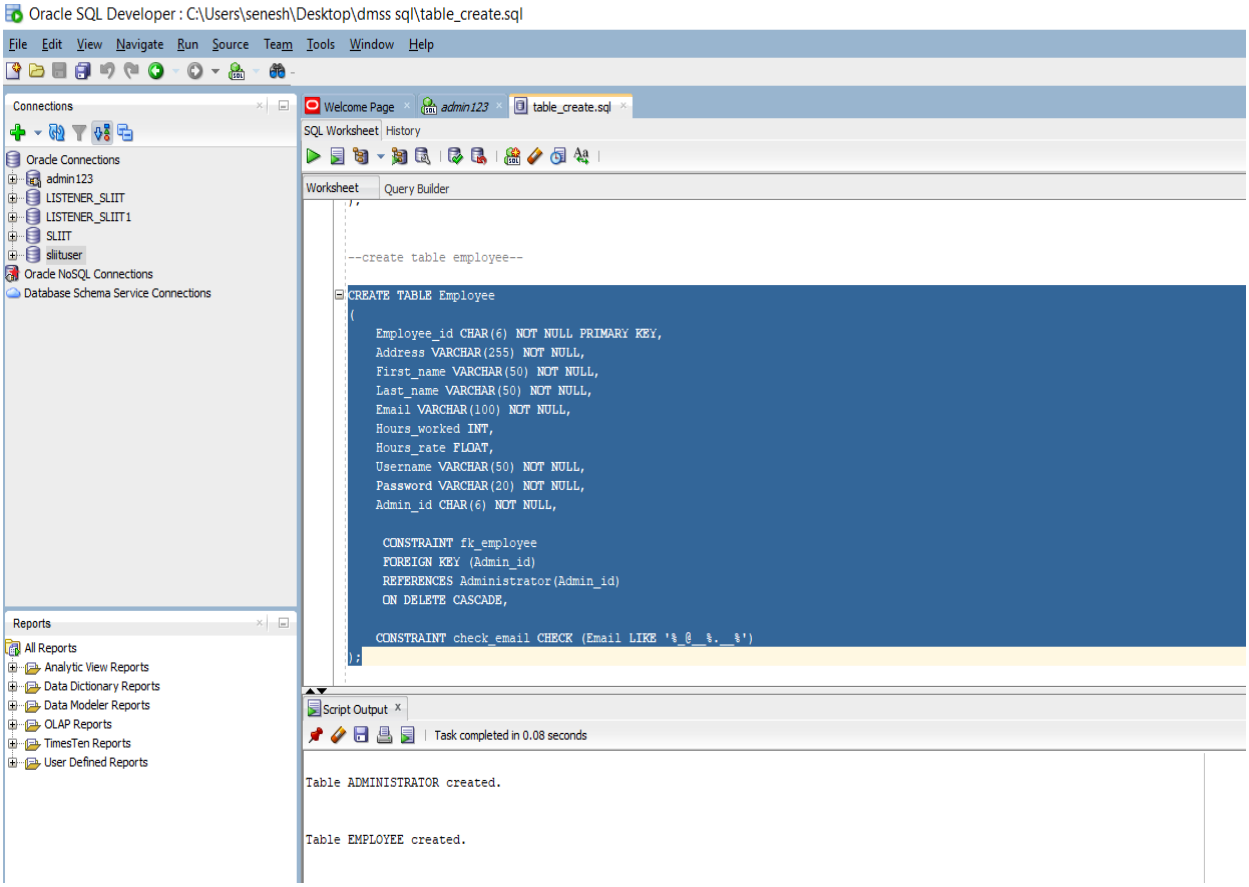


## 3.2 Relational Schema

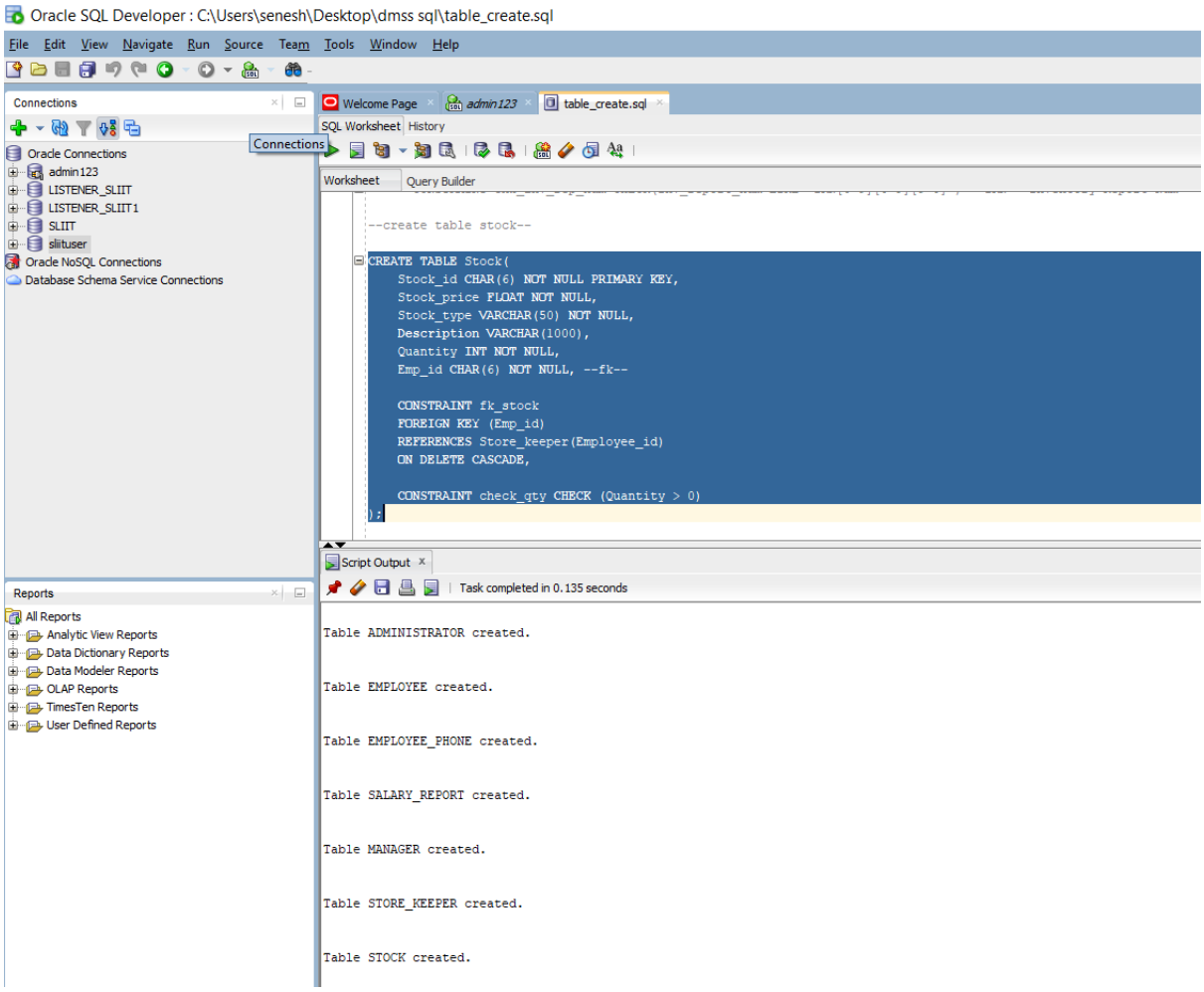


## 3.3 Screenshots of Table Creation Queries

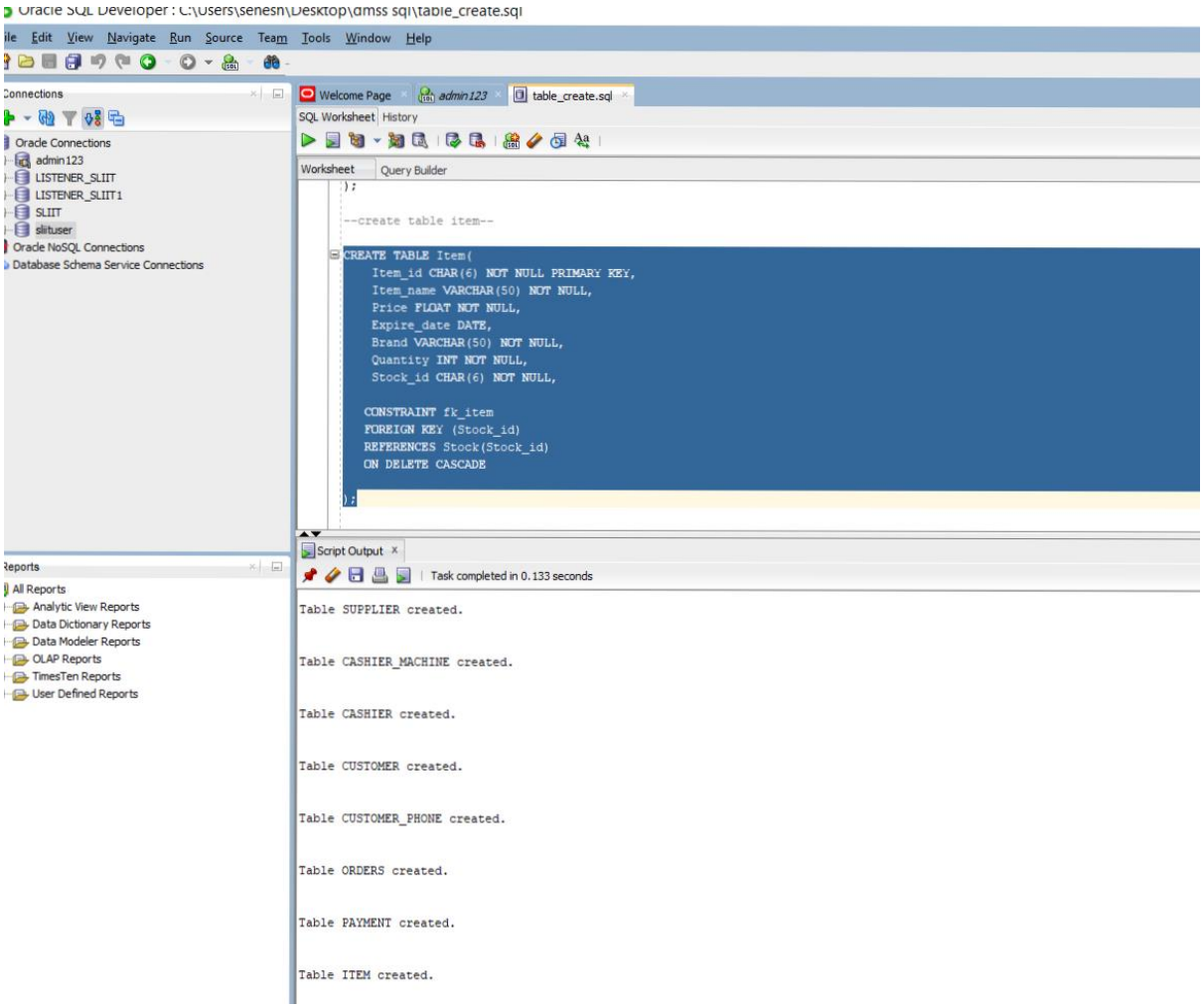
- **Create a table Employee**



- **Create a table Stock**



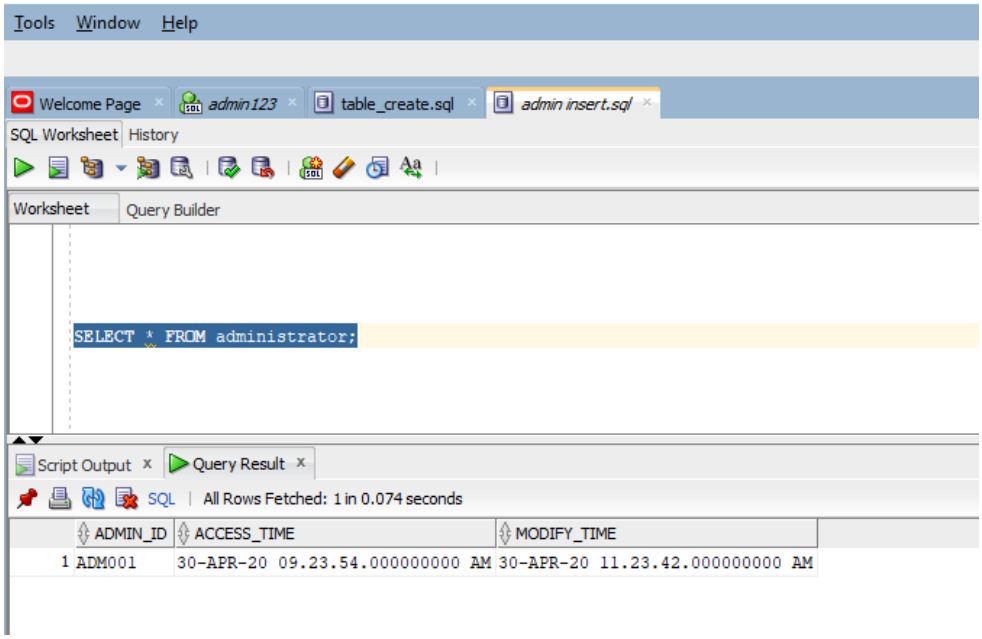
- **Create a table Item**



## 3.4 Screenshots of Tables after Inserting Data

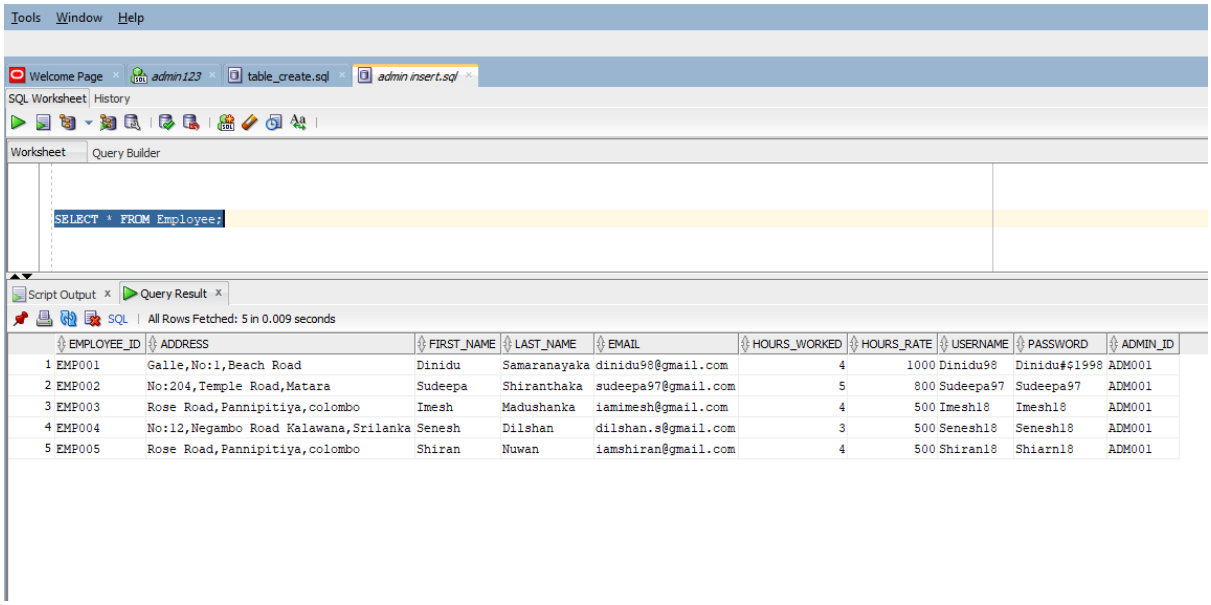
- **Administrator table**

Desktop\insert data screen shoot\admin insert.sql



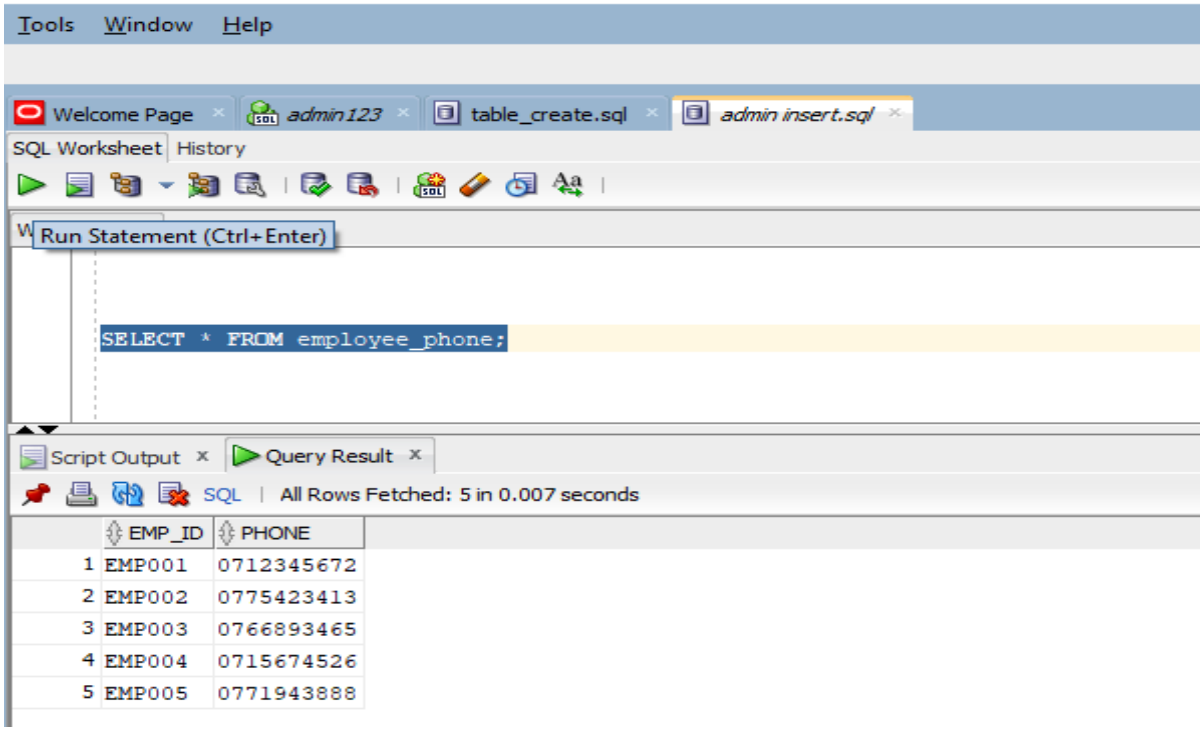

- **Employee table**

Desktop\insert data screen shoot\admin insert.sql



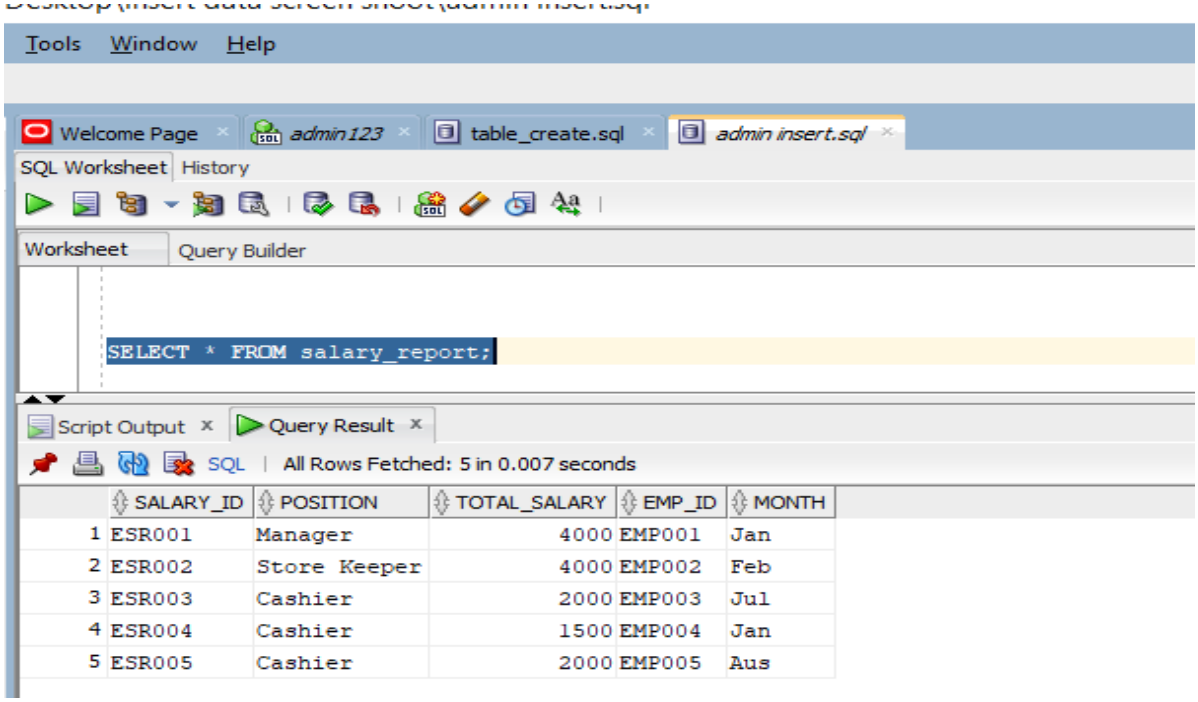
- **Employee_phone table**

Desktop\insert data screen shoot\admin insert.sql




| | EMP_ID | PHONE |
|---|---|---|
| 1 | EMP001 | 0712345672 |
| 2 | EMP002 | 0775423413 |
| 3 | EMP003 | 0766893465 |
| 4 | EMP004 | 0715674526 |
| 5 | EMP005 | 0771943888 |

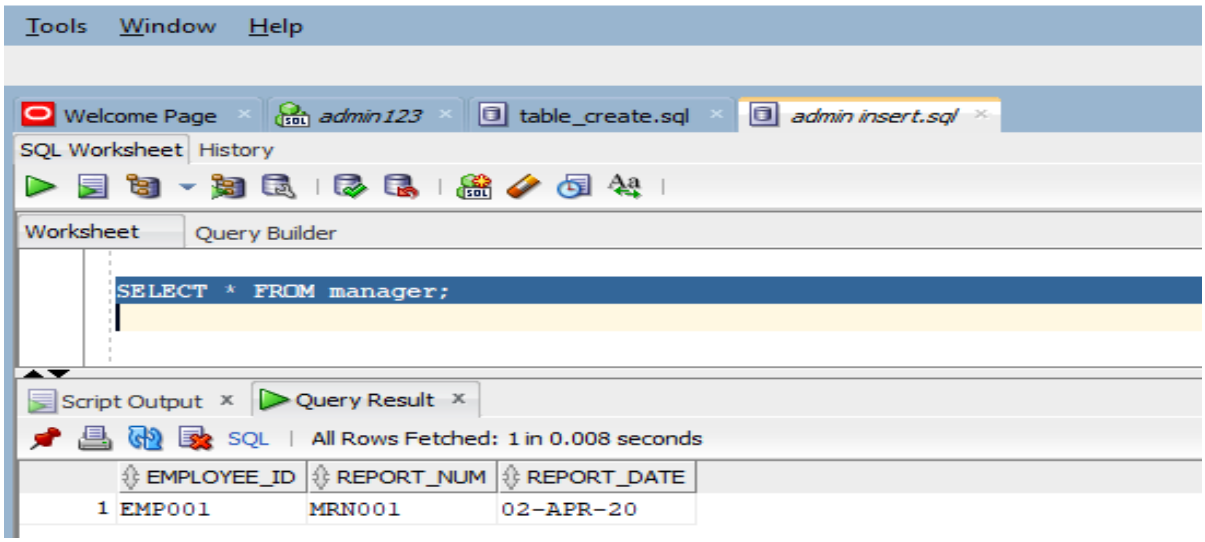
- **Salary_report table**




| | SALARY_ID | POSITION | TOTAL_SALARY | EMP_ID | MONTH |
|---|---|---|---|---|---|
| 1 | ESR001 | Manager | 4000 | EMP001 | Jan |
| 2 | ESR002 | Store Keeper | 4000 | EMP002 | Feb |
| 3 | ESR003 | Cashier | 2000 | EMP003 | Jul |
| 4 | ESR004 | Cashier | 1500 | EMP004 | Jan |
| 5 | ESR005 | Cashier | 2000 | EMP005 | Aus |

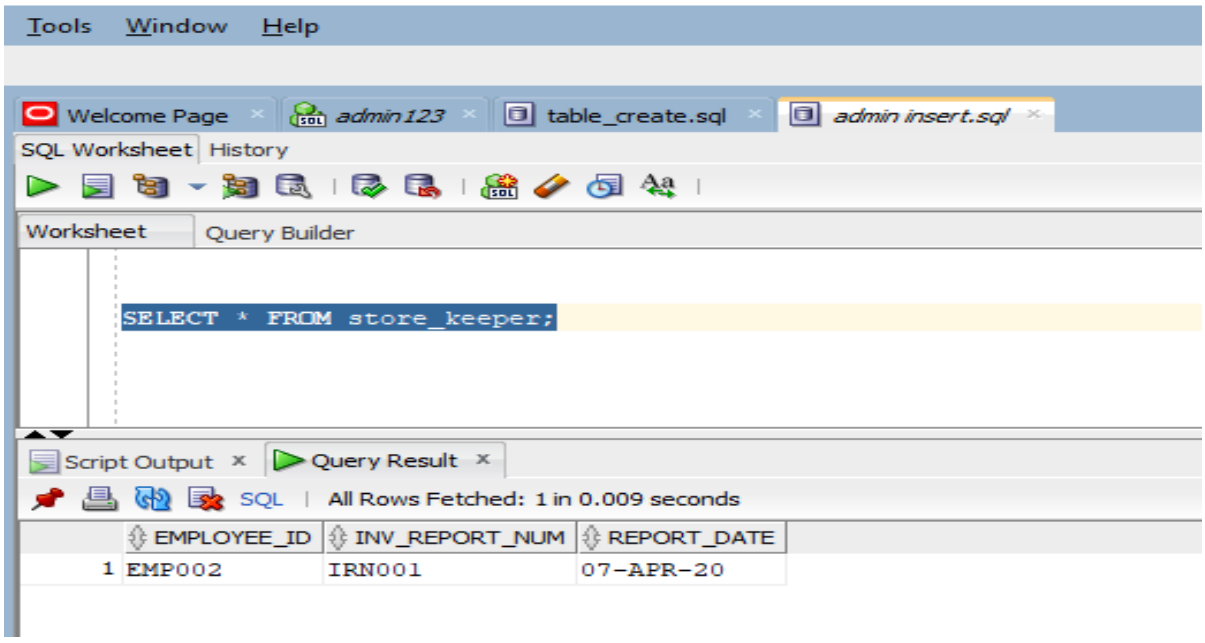
- **Manager table**

Desktop\insert data screen shoot\admin insert.sql



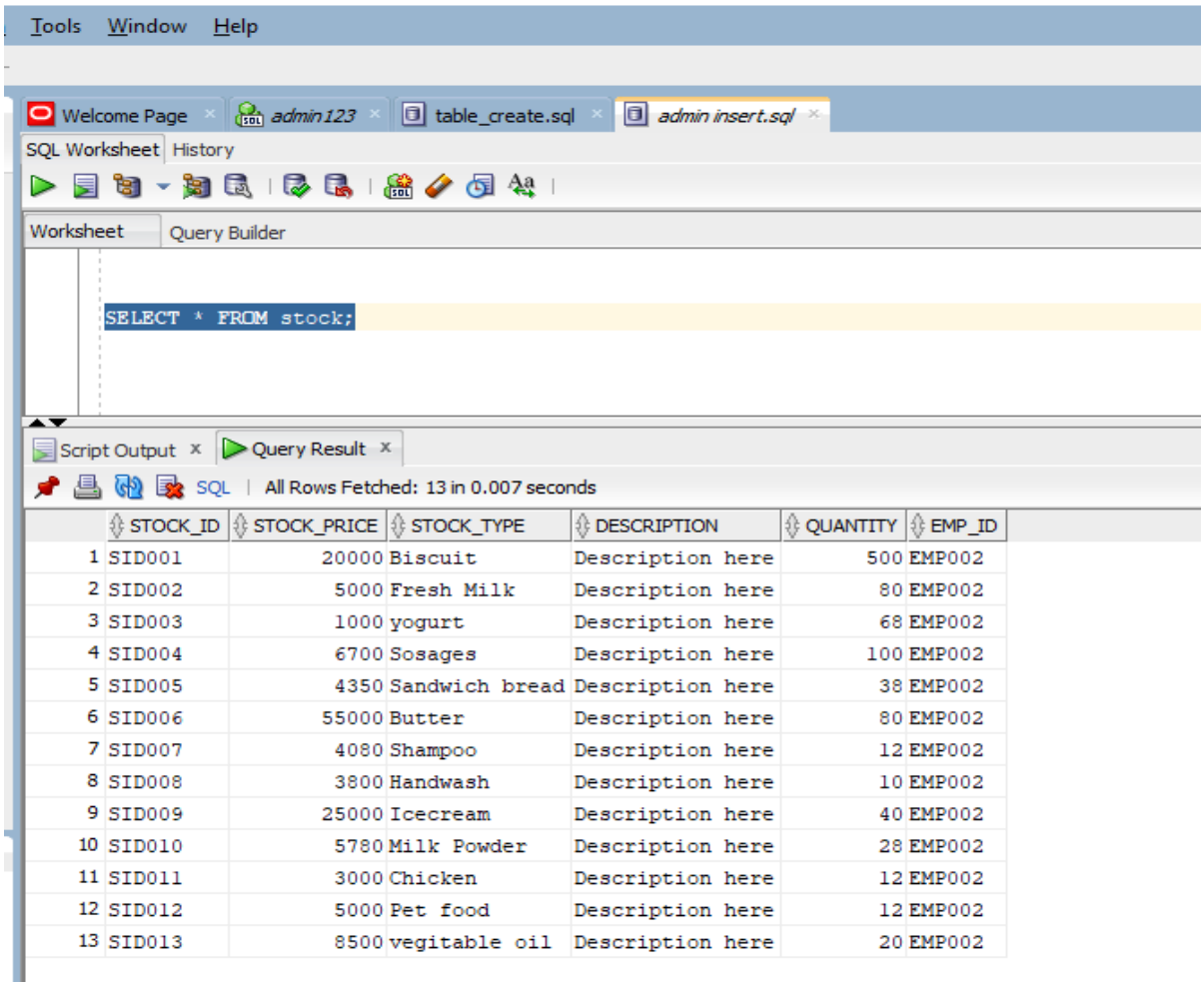

- **Store_keeper table**

Desktop\insert data screen shoot\admin insert.sql



- **Stock table**

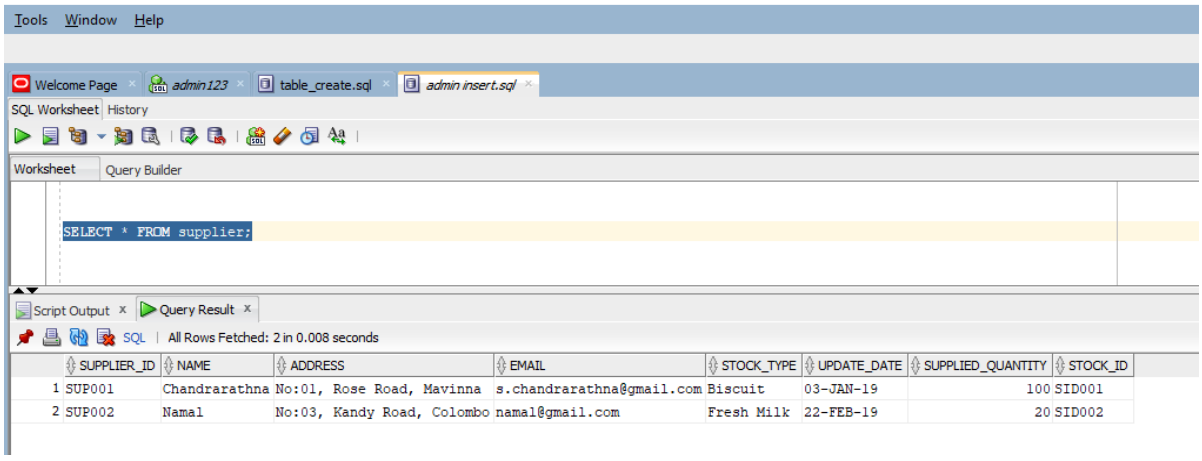
\Desktop\insert data screen shoot\admin insert.sql




| | STOCK_ID | STOCK_PRICE | STOCK_TYPE | DESCRIPTION | QUANTITY | EMP_ID |
|---|---|---|---|---|---|---|
| 1 | SID001 | 20000 | Biscuit | Description here | 500 | EMP002 |
| 2 | SID002 | 5000 | Fresh Milk | Description here | 80 | EMP002 |
| 3 | SID003 | 1000 | yogurt | Description here | 68 | EMP002 |
| 4 | SID004 | 6700 | Sosages | Description here | 100 | EMP002 |
| 5 | SID005 | 4350 | Sandwich bread | Description here | 38 | EMP002 |
| 6 | SID006 | 55000 | Butter | Description here | 80 | EMP002 |
| 7 | SID007 | 4080 | Shampoo | Description here | 12 | EMP002 |
| 8 | SID008 | 3800 | Handwash | Description here | 10 | EMP002 |
| 9 | SID009 | 25000 | Icecream | Description here | 40 | EMP002 |
| 10 | SID010 | 5780 | Milk Powder | Description here | 28 | EMP002 |
| 11 | SID011 | 3000 | Chicken | Description here | 12 | EMP002 |
| 12 | SID012 | 5000 | Pet food | Description here | 12 | EMP002 |
| 13 | SID013 | 8500 | vegitable oil | Description here | 20 | EMP002 |

- **Supplier table**

Desktop\insert data screen shoot\admin insert.sql




| | SUPPLIER_ID | NAME | ADDRESS | EMAIL | STOCK_TYPE | UPDATE_DATE | SUPPLIED_QUANTITY | STOCK_ID |
|---|---|---|---|---|---|---|---|---|
| 1 | SUP001 | Chandrarathna | No:01, Rose Road, Mavinna | s.chandrarathna@gmail.com | Biscuit | 03-JAN-19 | 100 | SID001 |
| 2 | SUP002 | Namal | No:03, Kandy Road, Colombo | namal@gmail.com | Fresh Milk | 22-FEB-19 | 20 | SID002 |

- **Cashier_machine table**

Desktop\insert data screen shoot\admin insert.sql

Tools Window Help

Welcome Page | admin123 | table_create.sql | admin insert.sql

SQL Worksheet History

Worksheet Query Builder

```
SELECT * FROM cashier_machine;
```

Script Output | Query Result

SQL | All Rows Fetched: 3 in 0.007 seconds

| | MACHINE_ID | CURRENT_AMOUNT |
|---|---|---|
| 1 | CMS001 | 10000 |
| 2 | CMS002 | 20000 |
| 3 | CMS003 | 30000 |

- **Cashier table**

Desktop\insert data screen shoot\admin insert.sql

Tools Window Help

Welcome Page | admin123 | table_create.sql | admin insert.sql

SQL Worksheet History

Worksheet Query Builder

```
SELECT * FROM cashier;
```

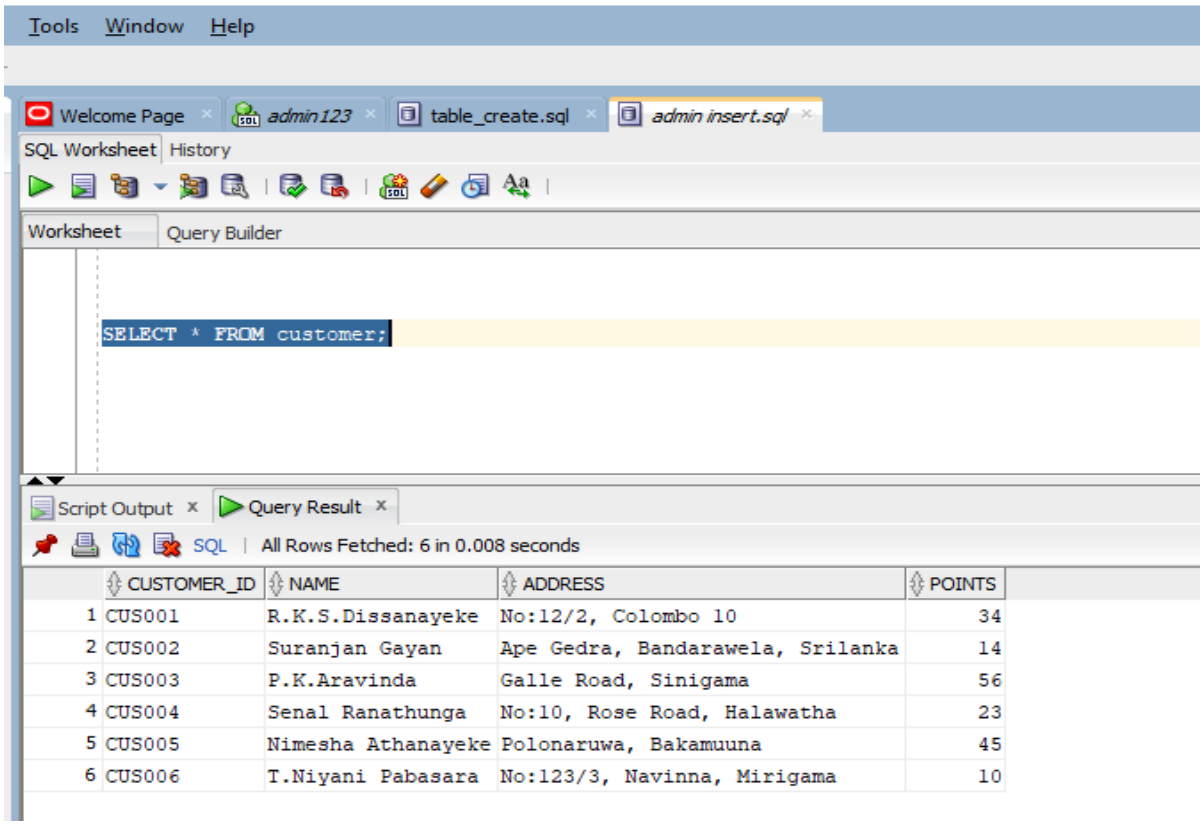
Script Output | Query Result

SQL | All Rows Fetched: 3 in 0.008 seconds

| | EMPLOYEE_ID | CUSTOMER_FEEDBACK | MACHINE_ID |
|---|---|---|---|
| 1 | EMP003 | enter the feed back here | CMS001 |
| 2 | EMP004 | enter the feed back here | CMS002 |
| 3 | EMP005 | enter the feed back here | CMS003 |

- **Customer table**

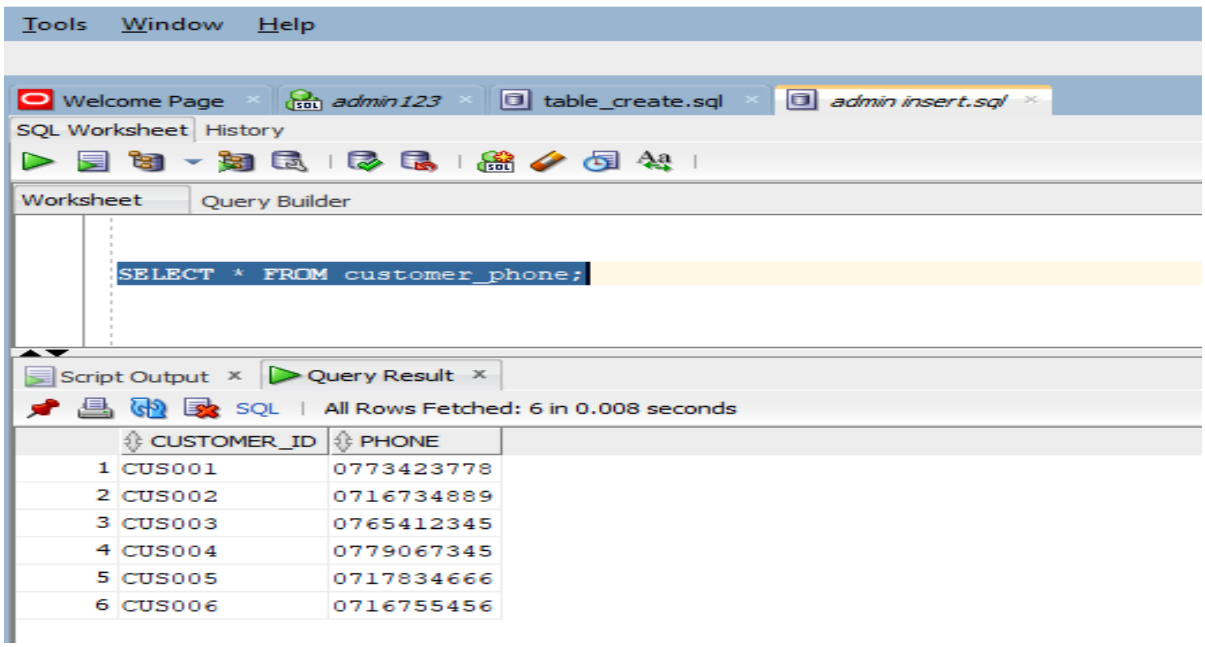
\Desktop\insert data screen shoot\admin insert.sql




| | CUSTOMER_ID | NAME | ADDRESS | POINTS |
|---|---|---|---|---|
| 1 | CUS001 | R.K.S.Dissanayeke | No:12/2, Colombo 10 | 34 |
| 2 | CUS002 | Suranjan Gayan | Ape Gedra, Bandarawela, Srilanka | 14 |
| 3 | CUS003 | P.K.Aravinda | Galle Road, Sinigama | 56 |
| 4 | CUS004 | Senal Ranathunga | No:10, Rose Road, Halawatha | 23 |
| 5 | CUS005 | Nimesha Athanayeke | Polonaruwa, Bakamuuna | 45 |
| 6 | CUS006 | T.Niyani Pabasara | No:123/3, Navinna, Mirigama | 10 |

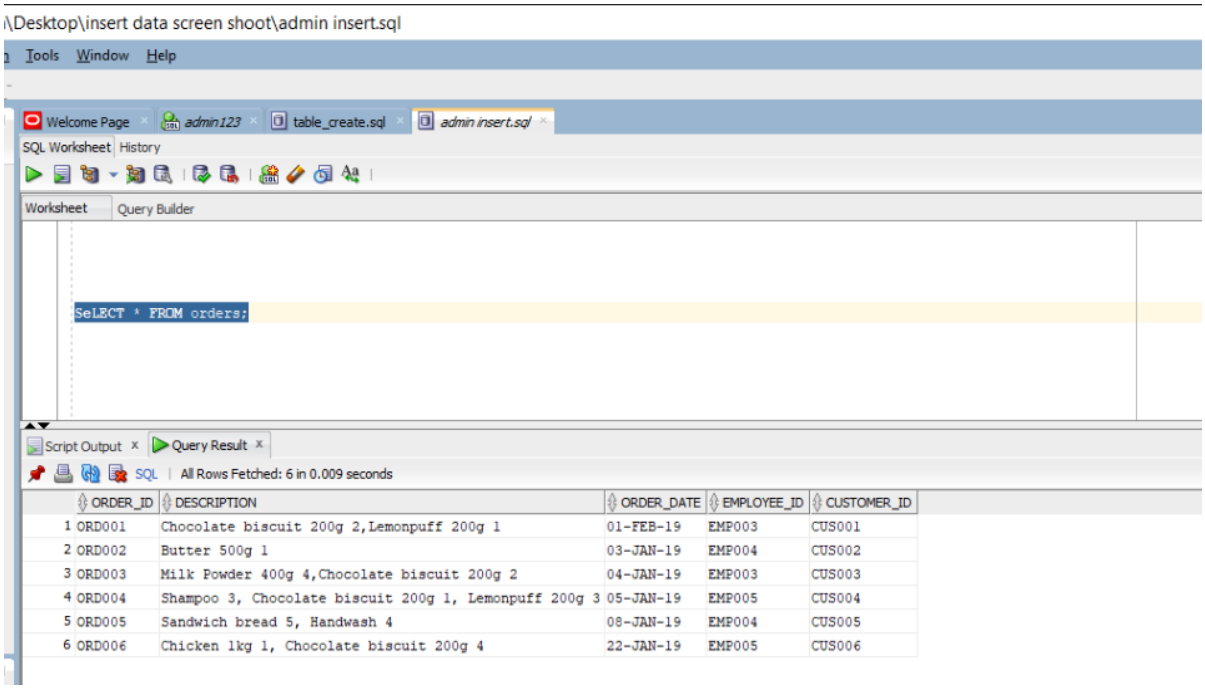
- **Customer_phone table**

Desktop\insert data screen shoot\admin insert.sql

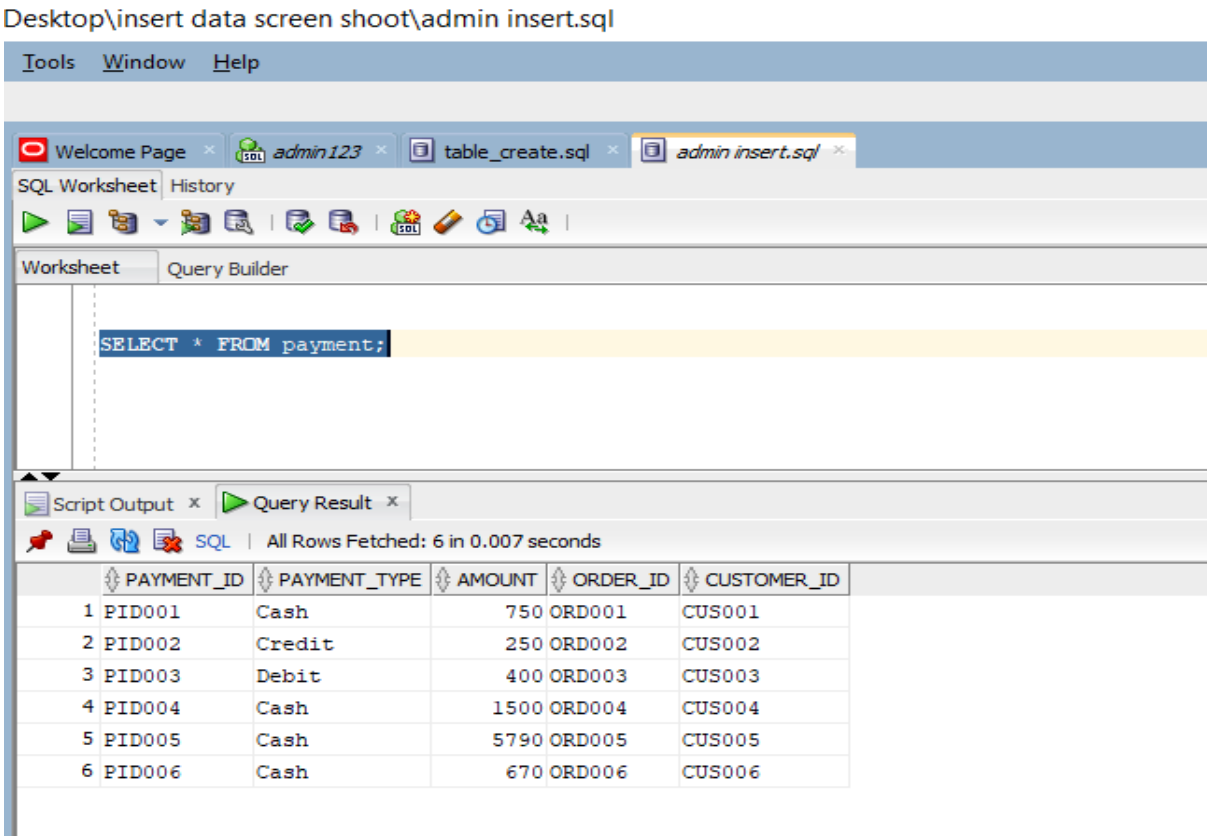



| | CUSTOMER_ID | PHONE |
|---|---|---|
| 1 | CUS001 | 0773423778 |
| 2 | CUS002 | 0716734889 |
| 3 | CUS003 | 0765412345 |
| 4 | CUS004 | 0779067345 |
| 5 | CUS005 | 0717834666 |
| 6 | CUS006 | 0716755456 |

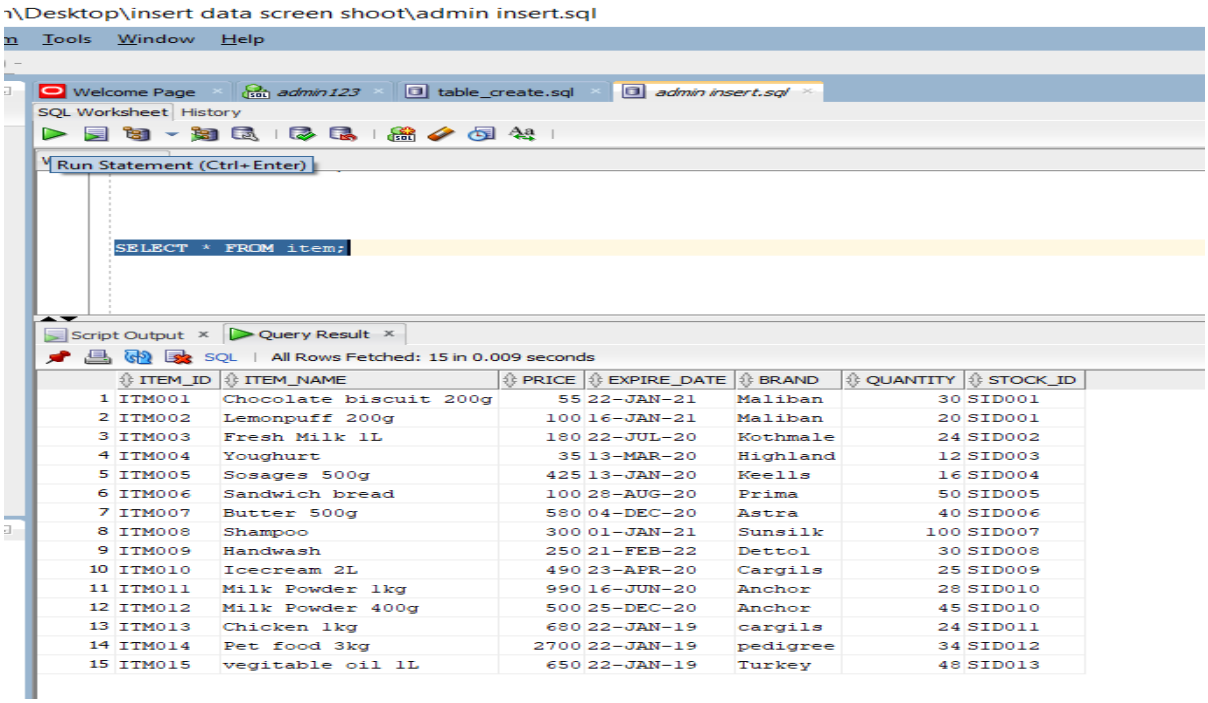
- **Orders table**




- **Payments table**

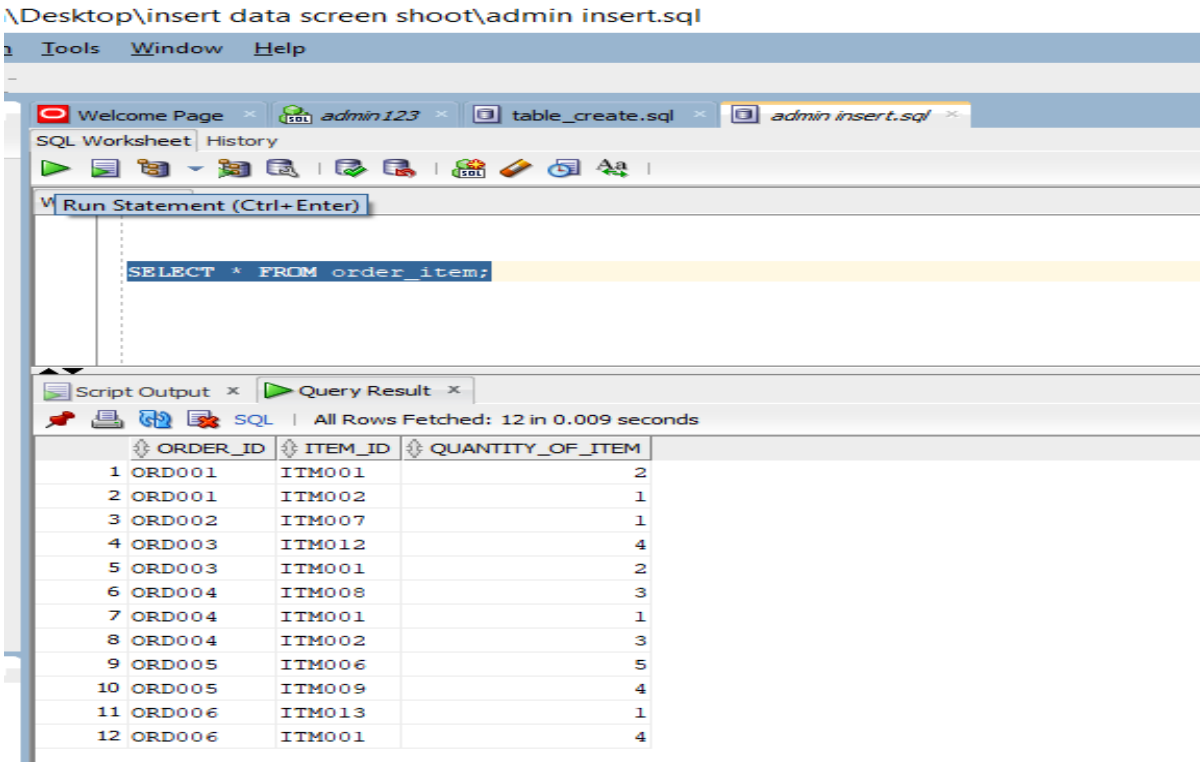


- **Items table**



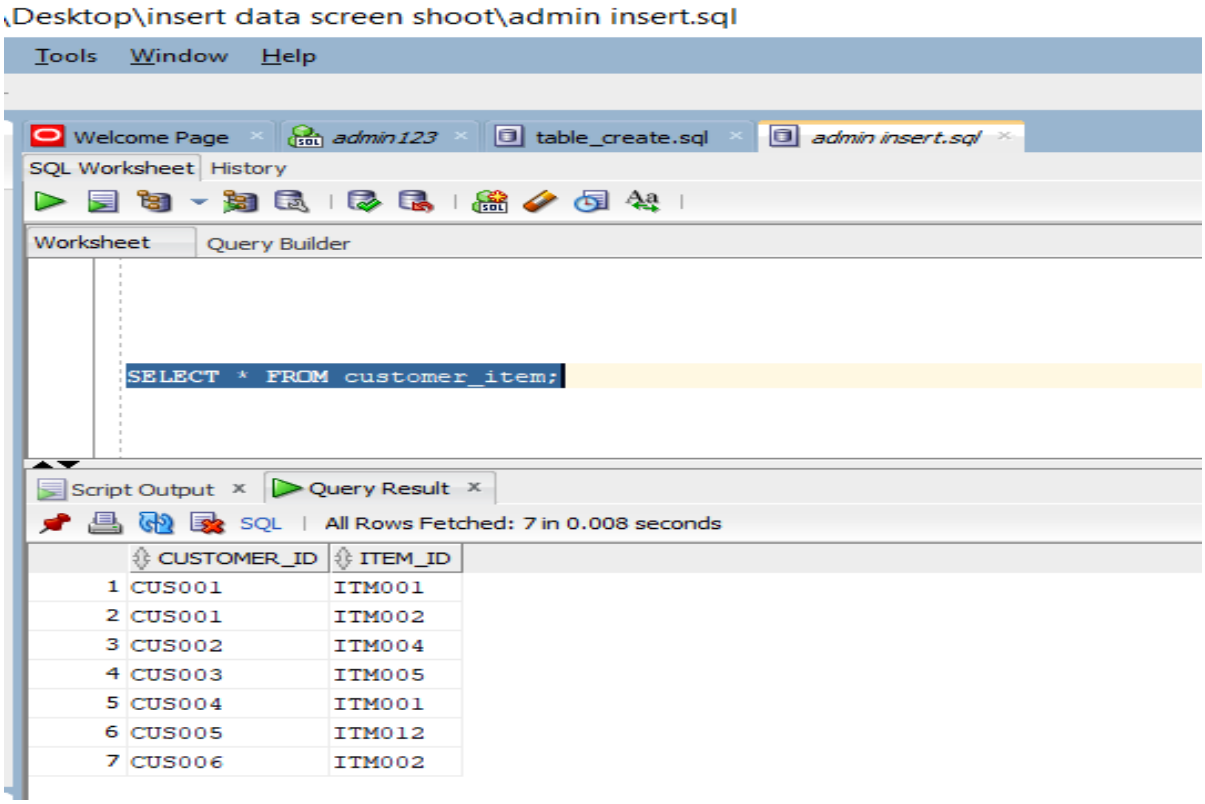

| | ITEM_ID | ITEM_NAME | PRICE | EXPIRE_DATE | BRAND | QUANTITY | STOCK_ID |
|---|---|---|---|---|---|---|---|
| 1 | ITM001 | Chocolate biscuit 200g | 55 | 22-JAN-21 | Maliban | 30 | SID001 |
| 2 | ITM002 | Lemonpuff 200g | 100 | 16-JAN-21 | Maliban | 20 | SID001 |
| 3 | ITM003 | Fresh Milk 1L | 180 | 22-JUL-20 | Kothmale | 24 | SID002 |
| 4 | ITM004 | Youghurt | 35 | 13-MAR-20 | Highland | 12 | SID003 |
| 5 | ITM005 | Sosages 500g | 425 | 13-JAN-20 | Keells | 16 | SID004 |
| 6 | ITM006 | Sandwich bread | 100 | 28-AUG-20 | Prima | 50 | SID005 |
| 7 | ITM007 | Butter 500g | 580 | 04-DEC-20 | Astra | 40 | SID006 |
| 8 | ITM008 | Shampoo | 300 | 01-JAN-21 | Sunsilk | 100 | SID007 |
| 9 | ITM009 | Handwash | 250 | 21-FEB-22 | Dettol | 30 | SID008 |
| 10 | ITM010 | Icecream 2L | 490 | 23-APR-20 | Cargils | 25 | SID009 |
| 11 | ITM011 | Milk Powder 1kg | 990 | 16-JUN-20 | Anchor | 28 | SID010 |
| 12 | ITM012 | Milk Powder 400g | 500 | 25-DEC-20 | Anchor | 45 | SID010 |
| 13 | ITM013 | Chicken 1kg | 680 | 22-JAN-19 | cargils | 24 | SID011 |
| 14 | ITM014 | Pet food 3kg | 2700 | 22-JAN-19 | pedigree | 34 | SID012 |
| 15 | ITM015 | vegitable oil 1L | 650 | 22-JAN-19 | Turkey | 48 | SID013 |

- **Order_items table**




| | ORDER_ID | ITEM_ID | QUANTITY_OF_ITEM |
|---|---|---|---|
| 1 | ORD001 | ITM001 | 2 |
| 2 | ORD001 | ITM002 | 1 |
| 3 | ORD002 | ITM007 | 1 |
| 4 | ORD003 | ITM012 | 4 |
| 5 | ORD003 | ITM001 | 2 |
| 6 | ORD004 | ITM008 | 3 |
| 7 | ORD004 | ITM001 | 1 |
| 8 | ORD004 | ITM002 | 3 |
| 9 | ORD005 | ITM006 | 5 |
| 10 | ORD005 | ITM009 | 4 |
| 11 | ORD006 | ITM013 | 1 |
| 12 | ORD006 | ITM001 | 4 |

- **Customer_items table**



## 3.5 SQL Commands to create the tables in the database

```
--create table admin--
CREATE TABLE Administrator
(
   Admin_id CHAR(6) NOT NULL PRIMARY KEY,
   Access_time TIMESTAMP,
   Modify_time TIMESTAMP
);

--create table employee--
CREATE TABLE Employee
(
   Employee_id CHAR(6) NOT NULL PRIMARY KEY,
   Address VARCHAR(255) NOT NULL,
   First_name VARCHAR(50) NOT NULL,
   Last_name VARCHAR(50) NOT NULL,
   Email VARCHAR(100) NOT NULL,
   Hours_worked INT,
   Hours_rate FLOAT,
   Username VARCHAR(50) NOT NULL,
   Password VARCHAR(20) NOT NULL,
   Admin_id CHAR(6) NOT NULL,

    CONSTRAINT fk_employee
    FOREIGN KEY (Admin_id)
    REFERENCES Administrator(Admin_id)
  ON DELETE CASCADE,
     CONSTRAINT check_email CHECK (Email LIKE '%_@__%.__%')
);
```

```
--create table employee_phone--
CREATE TABLE Employee_phone
(
   Emp_id CHAR(6) NOT NULL,
   Phone CHAR(10),
    CONSTRAINT pk_employee_phone PRIMARY KEY(Emp_id,Phone),
    CONSTRAINT fk_employee_phone
    FOREIGN KEY (Emp_id)
    REFERENCES Employee(Employee_id)
    ON DELETE CASCADE
);

--create table salary report--
CREATE TABLE Salary_report
(
   Salary_id CHAR(6) NOT NULL PRIMARY KEY,
   Position VARCHAR(50) NOT NULL,
   Total_salary FLOAT NOT NULL,
   Emp_id CHAR(6) NOT NULL,
   Month VARCHAR(20),
   CONSTRAINT fk_salary_employee
   FOREIGN KEY (Emp_id)
 REFERENCES Employee(Employee_id)
   ON DELETE CASCADE
  CONSTRAINT check_position CHECK (Position = 'Manager' OR Position = 'Store Keeper'
OR Position = 'Cashier')
);

--create table manager--
CREATE TABLE Manager
(
   Employee_id CHAR(6) NOT NULL PRIMARY KEY,
   Report_num CHAR(6),
   Report_date DATE,
```

```
    CONSTRAINT fk_manager
    FOREIGN KEY (Employee_id)
    REFERENCES Employee(Employee_id)
    ON DELETE CASCADE
);

--create table store_keeper--
CREATE TABLE Store_keeper(
    Employee_id CHAR(6) NOT NULL PRIMARY KEY,
    Inv_report_num CHAR(6),
    Report_date DATE,

    CONSTRAINT fk_store_keeper
    FOREIGN KEY (Employee_id)
    REFERENCES Employee(Employee_id)
    ON DELETE CASCADE
     );

--create table stock--
CREATE TABLE Stock(
    Stock_id CHAR(6) NOT NULL PRIMARY KEY,
    Stock_price FLOAT NOT NULL,
    Stock_type VARCHAR(50) NOT NULL,
    Description VARCHAR(1000),
    Quantity INT NOT NULL,
    Emp_id CHAR(6) NOT NULL, --fk--

    CONSTRAINT fk_stock
    FOREIGN KEY (Emp_id)
    REFERENCES Store_keeper(Employee_id)
    ON DELETE CASCADE,

    CONSTRAINT check_qty CHECK (Quantity > 0)
);
```

```
--create table supplier--
CREATE TABLE Supplier(
    Supplier_id CHAR(6) NOT NULL PRIMARY KEY,
    Name VARCHAR(50) NOT NULL,
    Address VARCHAR(255) NOT NULL,
    Email VARCHAR(100) NOT NULL,
    Stock_type VARCHAR(50) NOT NULL,
    Update_date DATE NOT NULL,
    Supplied_quantity INT NOT NULL,
    Stock_id CHAR(6) NOT NULL,

    CONSTRAINT fk_supplier
    FOREIGN KEY (Stock_id)
    REFERENCES Stock(Stock_id)
    ON DELETE CASCADE

);

--create table cashier machine--
CREATE TABLE Cashier_machine(
    Machine_id CHAR(6) NOT NULL PRIMARY KEY,
    Current_amount FLOAT NOT NULL
);

--create table cashier--
CREATE TABLE Cashier(
    Employee_id CHAR(6) NOT NULL PRIMARY KEY,
    Customer_feedback VARCHAR(500),
    Machine_id CHAR(6) NOT NULL,
   CONSTRAINT fk_cashier_1
   FOREIGN KEY (Employee_id)
   REFERENCES Employee(Employee_id)
   ON DELETE CASCADE,
   CONSTRAINT fk_cashier_2
```

```
  FOREIGN KEY (Machine_id)
  REFERENCES Cashier_machine(Machine_id)
  ON DELETE CASCADE

);

--create table customer--
CREATE TABLE Customer(
   Customer_id CHAR(6) NOT NULL PRIMARY KEY,
   Name VARCHAR(50),
   Address VARCHAR(255),
   Points INT
);

--create table customer phone--
CREATE TABLE Customer_phone(
   Customer_id CHAR(6) NOT NULL PRIMARY KEY,
   Phone CHAR(10),
CONSTRAINT fk_customer_phone
  FOREIGN KEY (Customer_id)
  REFERENCES Customer(Customer_id)
  ON DELETE CASCADE

);

--create table orders--
CREATE TABLE Orders(
   Order_id CHAR(6) NOT NULL PRIMARY KEY,
   Description VARCHAR(255),
   Order_date DATE NOT NULL,
   Employee_id CHAR(6) NOT NULL,
   Customer_id CHAR(6) NOT NULL,
```

```
  CONSTRAINT fk_order_1
  FOREIGN KEY (Employee_id)
  REFERENCES Cashier(Employee_id)
  ON DELETE CASCADE,

   CONSTRAINT fk_order_2
  FOREIGN KEY (Customer_id)
  REFERENCES Customer(Customer_id)
  ON DELETE CASCADE
);

--create table payment--
CREATE TABLE Payment(
   Payment_id CHAR(6) NOT NULL PRIMARY KEY,
   Payment_type VARCHAR(10) NOT NULL,
   Amount FLOAT NOT NULL,
   Order_id CHAR(6) NOT NULL,
   Customer_id CHAR(6) NOT NULL,

  CONSTRAINT fk_payment_1
  FOREIGN KEY (Order_id)
  REFERENCES Orders(Order_id)
  ON DELETE CASCADE,

  CONSTRAINT fk_payment_2
  FOREIGN KEY (Customer_id)
  REFERENCES Customer(Customer_id)
  ON DELETE CASCADE,

  CONSTRAINT check_payment_type CHECK (Payment_type = 'Cash' OR Payment_type
= 'Credit' OR Payment_type = 'Debit')
);
```

```
--create table item--
CREATE TABLE Item(
    Item_id CHAR(6) NOT NULL PRIMARY KEY,
    Item_name VARCHAR(50) NOT NULL,
    Price FLOAT NOT NULL,
    Expire_date DATE,
    Brand VARCHAR(50) NOT NULL,
    Quantity INT NOT NULL,
    Stock_id CHAR(6) NOT NULL,

   CONSTRAINT fk_item
   FOREIGN KEY (Stock_id)
   REFERENCES Stock(Stock_id)
   ON DELETE CASCADE

);

--create table order item--
    CREATE TABLE Order_item(
    Order_id CHAR(6) NOT NULL,
    Item_id CHAR(6) NOT NULL,
    Quantity_of_item INT NOT NULL,

    CONSTRAINT pk_order_item PRIMARY KEY(Order_id,Item_id),
    CONSTRAINT fk_order_item_1
    FOREIGN KEY (Order_id)
    REFERENCES Orders(Order_id)
    ON DELETE CASCADE,
    CONSTRAINT fk_order_item_2
    FOREIGN KEY (Item_id)
    REFERENCES Item(Item_id)
    ON DELETE CASCADE

);
```

```
--create table customer item--
   CREATE TABLE Customer_item(
   Customer_id CHAR(6) NOT NULL,
   Item_id CHAR(6) NOT NULL,

   CONSTRAINT pk_customer_item PRIMARY KEY(Customer_id,Item_id),
   CONSTRAINT fk_customer_item_1
   FOREIGN KEY (Customer_id)
   REFERENCES Customer(Customer_id)
   ON DELETE CASCADE,
   CONSTRAINT fk_customer_item_2
   FOREIGN KEY (Item_id)
   REFERENCES Item(Item_id)
   ON DELETE CASCADE

);
```

## 3.6 Commands to insert the data into the Database

```
--insert values to Administrator table--
INSERT ALL
INTO Administrator VALUES ('ADM001', '30-APR-2020 09:23:54','30-APR-2020
11:23:42')
SELECT * FROM DUAL;

--insert values to Employee table--
INSERT ALL
INTO Employee VALUES ('EMP001', 'Galle,No:1,Beach Road', 'Dinidu','Samaranayaka',
'dinidu98@gmail.com', 4, 1000.0, 'Dinidu98', 'Dinidu#$1998','ADM001')      --manager--
INTO Employee VALUES ('EMP002','No:204,Temple
Road,Matara','Sudeepa','Shiranthaka','sudeepa97@gmail.com', 5, 800.0, 'Sudeepa97',
'Sudeepa97','ADM001')   --store keeper--
INTO Employee VALUES ('EMP003','Rose
Road,Pannipitiya,colombo','Imesh','Madushanka','iamimesh@gmail.com', 4, 500.0,
'Imesh18','Imesh18','ADM001')      --cashier1--
INTO Employee VALUES ('EMP004','No:12,Negambo Road
Kalawana,Srilanka','Senesh','Dilshan','dilshan.s@gmail.com', 3, 500.0,
'Senesh18','Senesh18','ADM001')  --cashier2--
INTO Employee VALUES ('EMP005','Rose
Road,Pannipitiya,colombo','Shiran','Nuwan','iamshiran@gmail.com', 4, 500.0,
'Shiran18','Shiarn18','ADM001')  --cashier3--
SELECT * FROM DUAL;

--insert values to emp_phone table--
INSERT ALL
INTO Employee_phone VALUES ('EMP001','0712345672')
INTO Employee_phone VALUES ('EMP002','0775423413')
INTO Employee_phone VALUES ('EMP003','0766893465')
INTO Employee_phone VALUES ('EMP004','0715674526')
INTO Employee_phone VALUES ('EMP005','0771943888')
```

```
SELECT * FROM DUAL;

--insert values to Salary_report--
INSERT ALL
INTO Salary_report VALUES ('ESR001','Manager',4000.00,'EMP001','Jan')     --employee
salary report = ESR--
INTO Salary_report VALUES ('ESR002','Store Keeper',4000.00,'EMP002','Feb')
INTO Salary_report VALUES ('ESR003','Cashier',2000.00,'EMP003','Jul')
INTO Salary_report VALUES ('ESR004','Cashier',1500.00,'EMP004','Jan')
INTO Salary_report VALUES ('ESR005','Cashier',2000.00,'EMP005','Aus')
SELECT * FROM DUAL;

--insert values to manager table--
INSERT ALL
INTO Manager VALUES ('EMP001','MRN001','02-APR-2020')
SELECT * FROM DUAL;

--insert values to store_keeper table--
INSERT ALL
INTO Store_keeper VALUES ('EMP002','IRN001','07-APR-2020')
SELECT * FROM DUAL;

--insert values to stock--
INSERT ALL
INTO Stock VALUES ('SID001','20000.00','Biscuit','Description here',500,'EMP002')
INTO Stock VALUES ('SID002','5000.00','Fresh Milk','Description here',80,'EMP002')
INTO Stock VALUES ('SID003','1000.00','yogurt','Description here',68,'EMP002')
INTO Stock VALUES ('SID004','6700.00','Sosages','Description here',100,'EMP002')
INTO Stock VALUES ('SID005','4350.00','Sandwich bread','Description here',38,'EMP002')
INTO Stock VALUES ('SID006','55000.00','Butter','Description here',80,'EMP002')
INTO Stock VALUES ('SID007','4080.00','Shampoo','Description here',12,'EMP002')
INTO Stock VALUES ('SID008','3800.00','Handwash','Description here',10,'EMP002')
INTO Stock VALUES ('SID009','25000.00','Icecream','Description here',40,'EMP002')
INTO Stock VALUES ('SID010','5780.00','Milk Powder','Description here',28,'EMP002')
```

```
INTO Stock VALUES ('SID011','3000.00','Chicken','Description here',12,'EMP002')
INTO Stock VALUES ('SID012','5000.00','Pet food','Description here',12,'EMP002')
INTO Stock VALUES ('SID013','8500.00','vegitable oil','Description here',20,'EMP002')
SELECt * FROM DUAL;

--insert values to supplier--
INSERT ALL
INTO Supplier VALUES ('SUP001','Chandrarathna','No:01, Rose Road, Mavinna',
's.chandrarathna@gmail.com','Biscuit','03-JAN-2019',100,'SID001')
INTO Supplier VALUES ('SUP002','Namal','No:03, Kandy Road, Colombo',
'namal@gmail.com','Fresh Milk','22-FEB-2019',20,'SID002')
SELECT * FROM DUAL;

--insert values to cashier machine--
INSERT ALL
INTO Cashier_machine VALUES ('CMS001','10000.00') --CMS = cashier machine in store--
INTO Cashier_machine VALUES ('CMS002','20000.00') --CMS = cashier machine in store--
INTO Cashier_machine VALUES ('CMS003','30000.00') --CMS = cashier machine in store--
SELECT * FROM DUAL;

--insert values to cashier--
INSERT ALL
INTO Cashier VALUES ('EMP003','enter the feed back here','CMS001')
INTO Cashier VALUES ('EMP004','enter the feed back here','CMS002')
INTO Cashier VALUES ('EMP005','enter the feed back here','CMS003')
SELECT * FROM DUAL;

--insert values to customer--
INSERT ALL
INTO Customer VALUES ('CUS001','R.K.S.Dissanayeke','No:12/2, Colombo 10',34)
INTO Customer VALUES ('CUS002','Suranjan Gayan','Ape Gedra, Bandarawela,
Srilanka',14)
INTO Customer VALUES ('CUS003','P.K.Aravinda','Galle Road, Sinigama',56)
INTO Customer VALUES ('CUS004','Senal Ranathunga','No:10, Rose Road, Halawatha',23)
```

```
INTO Customer VALUES ('CUS005','Nimesha Athanayeke','Polonaruwa, Bakamuuna',45)
INTO Customer VALUES ('CUS006','T.Niyani Pabasara','No:123/3, Navinna, Mirigama',10)
SELECT * FROM DUAL;

--insert values to customer phone--
INSERT ALL
INTO Customer_phone VALUES ('CUS001','0773423778')
INTO Customer_phone VALUES ('CUS002','0716734889')
INTO Customer_phone VALUES ('CUS003','0765412345')
INTO Customer_phone VALUES ('CUS004','0779067345')
INTO Customer_phone VALUES ('CUS005','0717834666')
INTO Customer_phone VALUES ('CUS006','0716755456')
SELECT * FROM DUAL;

--insert values to orders--
INSERT ALL
INTO Orders VALUES ('ORD001','Chocolate biscuit 200g 2,Lemonpuff 200g 1','01-FEB-2019','EMP003','CUS001')
INTO Orders VALUES ('ORD002','Butter 500g 1','03-JAN-2019','EMP004','CUS002')
INTO Orders VALUES ('ORD003','Milk Powder 400g 4,Chocolate biscuit 200g 2','04-JAN-2019','EMP003','CUS003')
INTO Orders VALUES ('ORD004','Shampoo 3, Chocolate biscuit 200g 1, Lemonpuff 200g 3','05-JAN-2019','EMP005','CUS004')
INTO Orders VALUES ('ORD005','Sandwich bread 5, Handwash 4','08-JAN-2019','EMP004','CUS005')
INTO Orders VALUES ('ORD006','Chicken 1kg 1, Chocolate biscuit 200g 4','22-JAN-2019','EMP005','CUS006')
SELECT * FROM DUAL;

--insert values to payments--
INSERT ALL
INTO Payment VALUES ('PID001','Cash','750.00','ORD001','CUS001')
--PID = paymentID--
INTO Payment VALUES ('PID002','Credit','250.00','ORD002','CUS002')
```

```
INTO Payment VALUES ('PID003','Debit','400.00','ORD003','CUS003')
INTO Payment VALUES ('PID004','Cash','1500.00','ORD004','CUS004')
INTO Payment VALUES ('PID005','Cash','5790.00','ORD005','CUS005')
INTO Payment VALUES ('PID006','Cash','670.00','ORD006','CUS006')
SELECT * FROM DUAL;

--insert values to items--
INSERT ALL
INTO Item VALUES ('ITM001','Chocolate biscuit 200g','55.00','22-JAN-
2021','Maliban',30,'SID001')
INTO Item VALUES ('ITM002','Lemonpuff 200g','100.00','16-JAN-
2021','Maliban',20,'SID001')
INTO Item VALUES ('ITM003','Fresh Milk 1L','180.00','22-JUL-
2020','Kothmale',24,'SID002')
INTO Item VALUES ('ITM004','Youghurt','35.00','13-MAR-2020','Highland',12,'SID003')
INTO Item VALUES ('ITM005','Sosages 500g','425.00','13-JAN-2020','Keells',16,'SID004')
INTO Item VALUES ('ITM006','Sandwich bread','100.00','28-AUG-
2020','Prima',50,'SID005')
INTO Item VALUES ('ITM007','Butter 500g','580.00','04-DEC-2020','Astra',40,'SID006')
INTO Item VALUES ('ITM008','Shampoo','300.00','01-JAN-2021','Sunsilk',100,'SID007')
INTO Item VALUES ('ITM009','Handwash','250.00','21-FEB-2022','Dettol',30,'SID008')
INTO Item VALUES ('ITM010','Icecream 2L','490.00','23-APR-2020','Cargils',25,'SID009')
INTO Item VALUES ('ITM011','Milk Powder 1kg','990.00','16-JUN-
2020','Anchor',28,'SID010')
INTO Item VALUES ('ITM012','Milk Powder 400g','500.00','25-DEC-
2020','Anchor',45,'SID010')
INTO Item VALUES ('ITM013','Chicken 1kg','680.00','22-JAN-2019','cargils',24,'SID011')
INTO Item VALUES ('ITM014','Pet food 3kg','2700.00','22-JAN-
2019','pedigree',34,'SID012')
INTO Item VALUES ('ITM015','vegitable oil 1L','650.00','22-JAN-
2019','Turkey',48,'SID013')
SELECT * FROM DUAL;
```

```
--insert valus to order items--
INSERT ALL
INTO Order_item VALUES ('ORD001','ITM001',2)
INTO Order_item VALUES ('ORD001','ITM002',1)
INTO Order_item VALUES ('ORD002','ITM007',1)
INTO Order_item VALUES ('ORD003','ITM012',4)
INTO Order_item VALUES ('ORD003','ITM001',2)

INTO Order_item VALUES ('ORD004','ITM008',3)
INTO Order_item VALUES ('ORD004','ITM001',1)
INTO Order_item VALUES ('ORD004','ITM002',3)

INTO Order_item VALUES ('ORD005','ITM006',5)
INTO Order_item VALUES ('ORD005','ITM009',4)

INTO Order_item VALUES ('ORD006','ITM013',1)
INTO Order_item VALUES ('ORD006','ITM001',4)
SELECT * FROM DUAL;

--insert values to customer items table--
INSERT ALL
INTO Customer_item VALUES ('CUS001','ITM001')
INTO Customer_item VALUES ('CUS001','ITM002')
INTO Customer_item VALUES ('CUS002','ITM004')
INTO Customer_item VALUES ('CUS003','ITM005')
INTO Customer_item VALUES ('CUS004','ITM001')
INTO Customer_item VALUES ('CUS005','ITM012')
INTO Customer_item VALUES ('CUS006','ITM002')
SELECT * FROM DUAL;
```

# 4.Implementing access control privileges

When giving the access control privileges and adding a user for relevant roles had done by the admin of the supermarket management system. Admin is the person who responsible for all superuser or administrative privileges after system user.

## 4.1 Creating a role for the manager and adding the user

SQL>alter session set "_oracle_script" = true;
Session altered.
SQL>create role Manager;
Role created.
SQL>create user Dinidu98 identified by Dinidu#$1998;
User created.
SQL>grant create session to Manager;
Grant succeeded.
SQL>grant connect to Manager;
Grant succeeded.
SQL>grant select, update, delete on Stock to Manager;
Grant succeeded.
SQL>grant select, update, delete on Employee to Manager;
Grant succeeded.
SQL>grant create any view to Manager;
Grant succeeded.
SQL>grant select, insert, update, delete on Salary_report to Manager;
Grant succeeded.
SQL>grant select, update on Customer to Manager;
Grant succeeded.
SQL>grant select, insert, update, delete on Cashier_machine to Manager;
Grant succeeded.
SQL>grant select, insert, update, delete on Payment to Manager;
Grant succeeded.
SQL>grant select, insert, update, delete on Orders to Manager;
Grant succeeded.

SQL>grant select, update on Customer to Manager;

Grant succeeded.

SQL> grant select, update on Item to Manager;

Grant succeeded.

SQL>grant select, update on Supplier to Manager;

Grant succeeded.

SQL>grant select on employee_details to Manager;

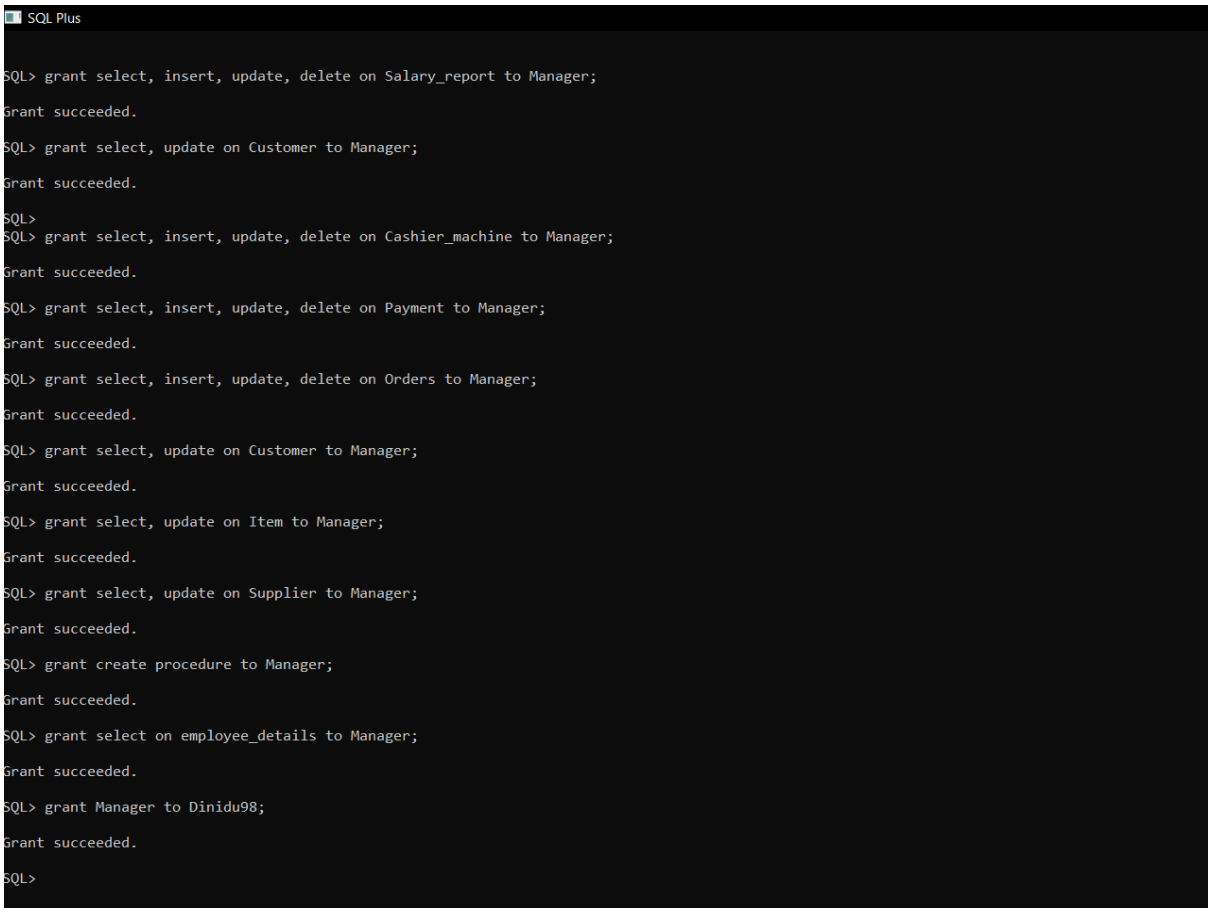
Grant succeeded.

SQL>grant create procedure to Manager;

Grant succeeded.

SQL> grant Manager to Dinidu98;

Grant succeeded.




## 4.2 Creating a role for store_keeper and adding the user

SQL>create role Store_keeper;

Role created.

SQL>create user Sudeepa97 identified by Sudeepa97;

User created.

SQL>grant create session to Store_keeper;

Grant succeeded.

SQL>grant connect to Store_keeper;

Grant succeeded.

SQL>grant select, insert, delete, update on Stock to Store_keeper;

Grant succeeded.

SQL>grant select, insert, delete, update on Item to Store_keeper;

Grant succeeded.

SQL>grant select on Supplier to Store_keeper;

Grant succeeded.

SQL>grant select Supplier_Details to Store_keeper;

Grant succeeded.

SQL> grant create procedure to Store_keeper;

Grant succeeded.

SQL>grant Store_keeper to Sudeepa97**;**

Grant succeeded.

## 4.3 Creating a role for cashier and adding the user

SQL>create role Cashier;

Role created.

SQL>create user Imesh18 identified by Imesh18#;

User created.

SQL>grant create session to Cashier;

Grant succeeded.

SQL>grant connect to Cashier;

Grant succeeded.

SQL>grant select, insert, update on Payment to Cashier;

Grant succeeded.

SQL>grant select on Stock to Cashier;

Grant succeeded.

SQL>grant select on Item to Cashier;

Grant succeeded.

SQL>grant create procedure to Cashier;

Grant succeeded.

SQL>grant select, update on Cashier_machine to Cashier;

Grant succeeded.

SQL>grant insert, update, select on Orders to Cashier;

Grant succeeded.

SQL>grant select on stock_details to Cashier;

Grant succeeded.

SQL>grant Cashier to Imesh18;

Grant succeeded.

# 5.Possible attacks and countermeasures when the database connect to a web application

## 5.1 SQL Injection

SQL Injection (SQLi) is a kind of injection assault that makes it conceivable to execute malevolent SQL explanations. These announcements control a database server behind a web application. Aggressors can utilize SQL Injection vulnerabilities to sidestep application safety efforts. They can circumvent confirmation and approval of a website page or web application and recover the substance of the whole SQL database. They can likewise utilize SQL Injection to include, alter, and erase records in the database.

### 5.1.1 Countermeasures to prevent SQL injection

- Train and maintain awareness
- Don’t trust any user inputs
- Use whitelist, not backlist
- Employee verification mechanism

## 5.2 Cross-Site Scripting

Cross-site Scripting (XSS) is a customer side code injection assault. The aggressor plans to execute malignant contents in an internet browser of the casualty by remembering malevolent code for an authentic page or web application. The real assault happens when the casualty visits the page or web application that executes the pernicious code. The site page or web application turns into a vehicle to convey the malignant content to the client's program. Helpless vehicles that are ordinarily utilized for Cross-webpage Scripting assaults are gatherings, message sheets, and site pages that permit remarks.

### 5.2.1 Countermeasures to prevent XSS include:

- Never insert untrusted data except in allowed locations
- Sanitize HTML files and pages.
- Attribute escape before inserting untrusted data into HTML common attributes
- JavaScript escape before inserting untrusted data into JavaScript data values
- URL escape before inserting untrusted data into HTML URL parameter values

## 5.3 Denial of Service

A denial-of-service (DoS) assault is a kind of digital assault wherein a malevolent on-screen character expects to render a PC or other gadget inaccessible to its planned clients by intruding on the gadget's typical working. DoS assaults commonly work by overpowering or flooding a focused on a machine with demands until ordinary traffic can't be handled, bringing about disavowal of-administration to expansion clients. A DoS assault is described by utilizing a solitary PC to dispatch the assault.

### 5.3.1 Countermeasures to prevent denial of service

- Test and apply security patches (including service packs and firmware updates) as soon as possible
- Use an IPS to monitor regularly for DoS attacks.
- Configure firewalls and routers to block malformed traffic.
- Block all ICMP traffic inbound to your network unless you specifically need it

## 5.4 Password Cracking

On the off chance that the aggressor can't build up a mysterious association with the server, the individual in question will attempt to set up a verified association. For this, the assailant must know a legitimate username and secret essential blend. If you use default account names, you are giving the assailant a head start. At that point, the assailant just needs to break the record's secret word. The utilization of clear or feeble passwords makes the assailant's activity even more accessible. Since the clients need to make a record before utilizing the application, and because the record contains all the individual and installment data of the client, it will be the primary objective of the attacker. The client probably won't consider this while making their record, so we have to ensure that the client utilizes a solid secret word.

### 5.4.1 Countermeasures to help prevent password cracking

- Use passwords policies.
- Create strong passwords.
- Enable security auditing to help monitor and track password attacks
- Test your applications to make sure they aren’t storing passwords indefinitely in memory or writing them to disk

## 5.5 Privilege Escalation

Benefit heightening happens when a malignant client abuses a bug, plan defect, or setup blunder in an application or working framework to increase raised access to assets that ought to be inaccessible to that client typically. The assailant would then be able to utilize the recently picked up benefits to take secret information, run managerial orders or send malware – and possibly harm your working framework, server applications, association, and notoriety. In this blog entry, we will take a gander at the run of the mill benefit heightening situations and figure out how you can ensure client accounts in your frameworks and applications to keep up a decent security pose.

### 5.5.1 Countermeasures to prevent privilege escalation

- Preventing Privilege Account Escalations
- Monitor for Creeps and Exploits

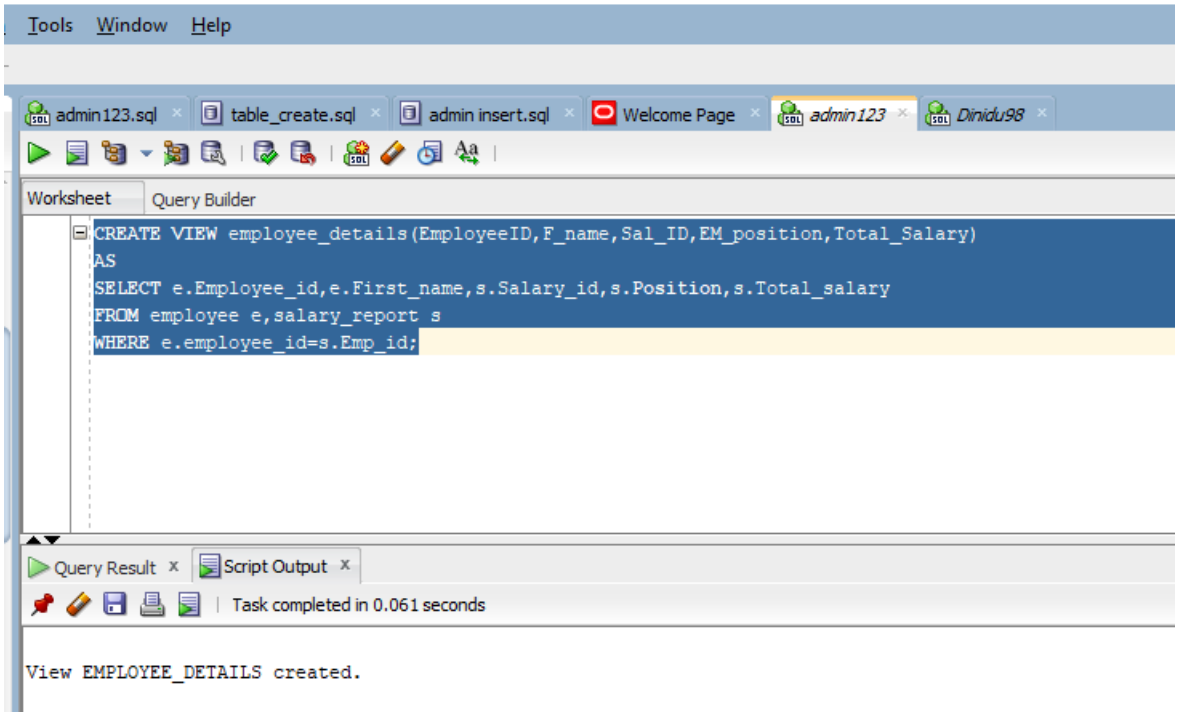
# 6. Implementation of countermeasures methods and additional security implementations

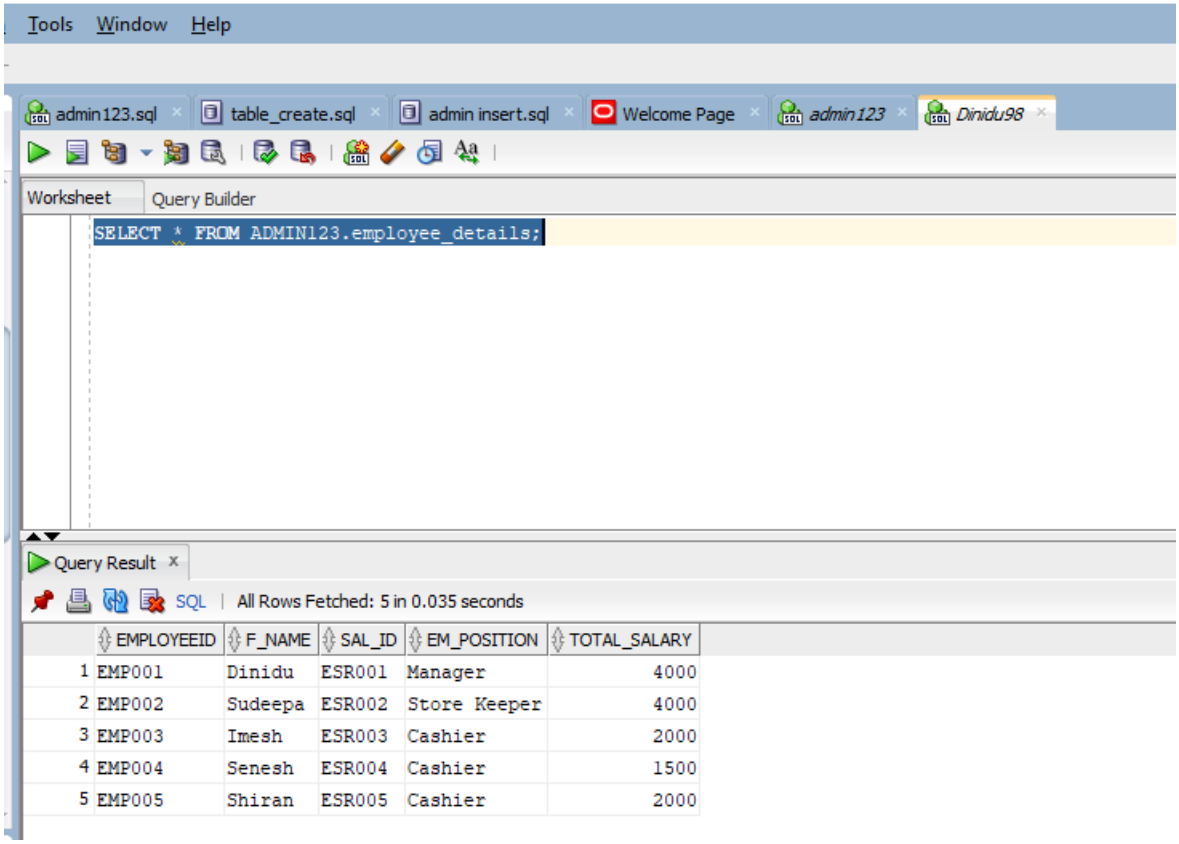
## 6.1 Views

### 6.1.1 Create view employee_details

Creating a View called “employee_details” by Admin and granting only “Select” permission to Manager only to see the relevant fields.

**CREATE VIEW**
**employee_details(EmployeeID,F_name,Sal_ID,EM_position,Total_Salary)**
**AS**
**SELECT e.Employee_id,e.First_name,s.Salary_id,s.Position,s.Total_salary**
**FROM employee e,salary_report s**
**WHERE e.employee_id=s.Emp_id;**



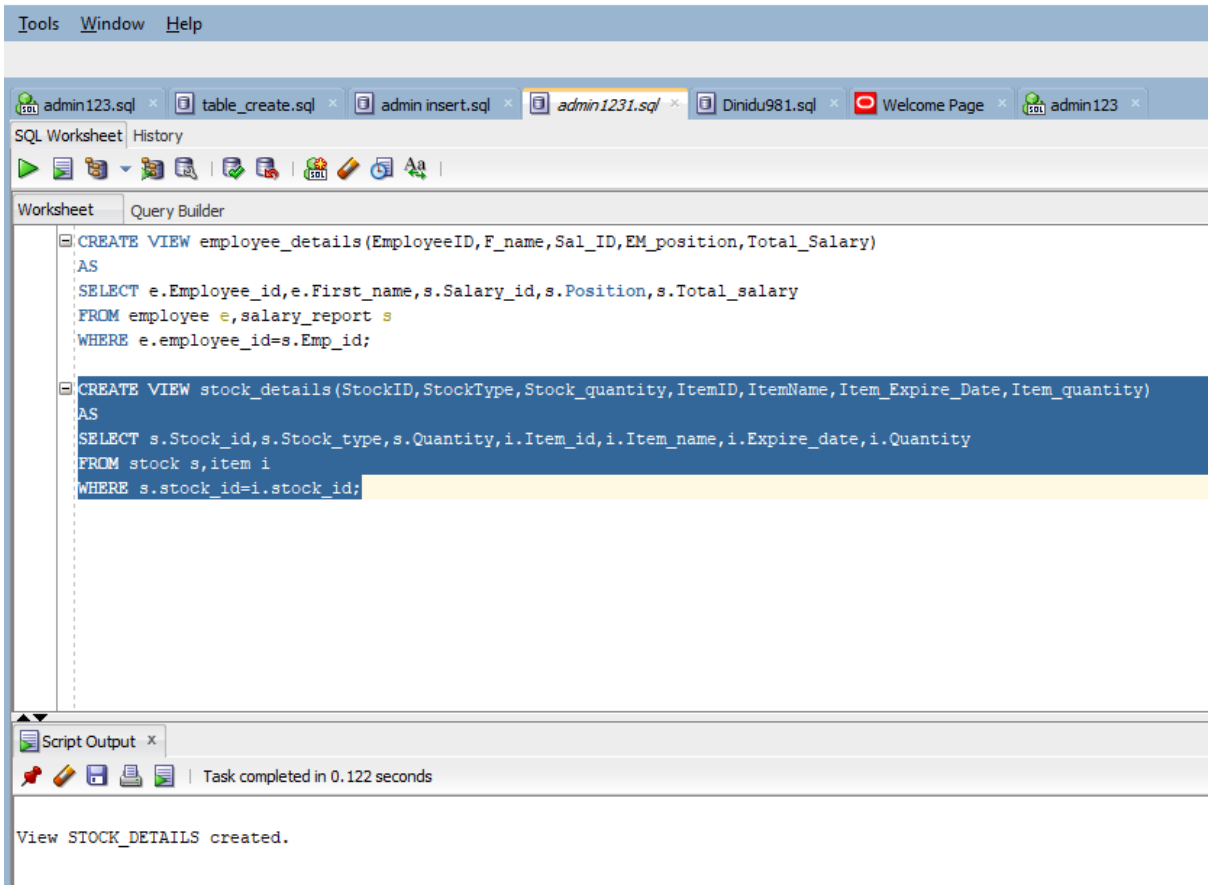


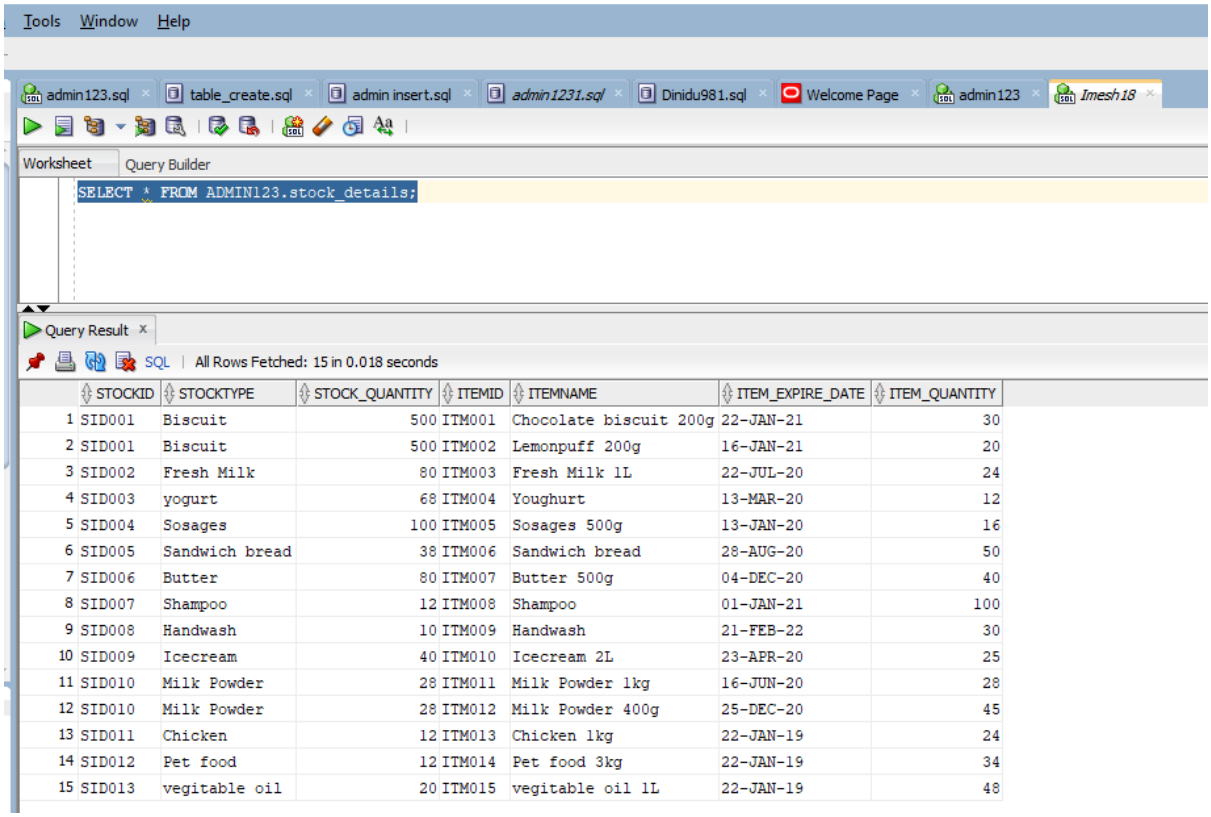

### 6.1.2 Create view stock_details

Creating a View called “stock_details” by Admin and granting only “Select” permission to Cashier only to see the relevant fields.

**CREAT VIEW**
**stock_details(StockID,StockType,Stock_quantity,ItemID,ItemName,Item_Expire_Date, Item_quantity)**
**AS**
**SELECT**
**s.Stock_id,s.Stock_type,s.Quantity,i.Item_id,i.Item_name,i.Expire_date,i.Quantity**
**FROM stock s,item i**
**WHERE s.stock_id=i.stock_id;**

admin1231.sql



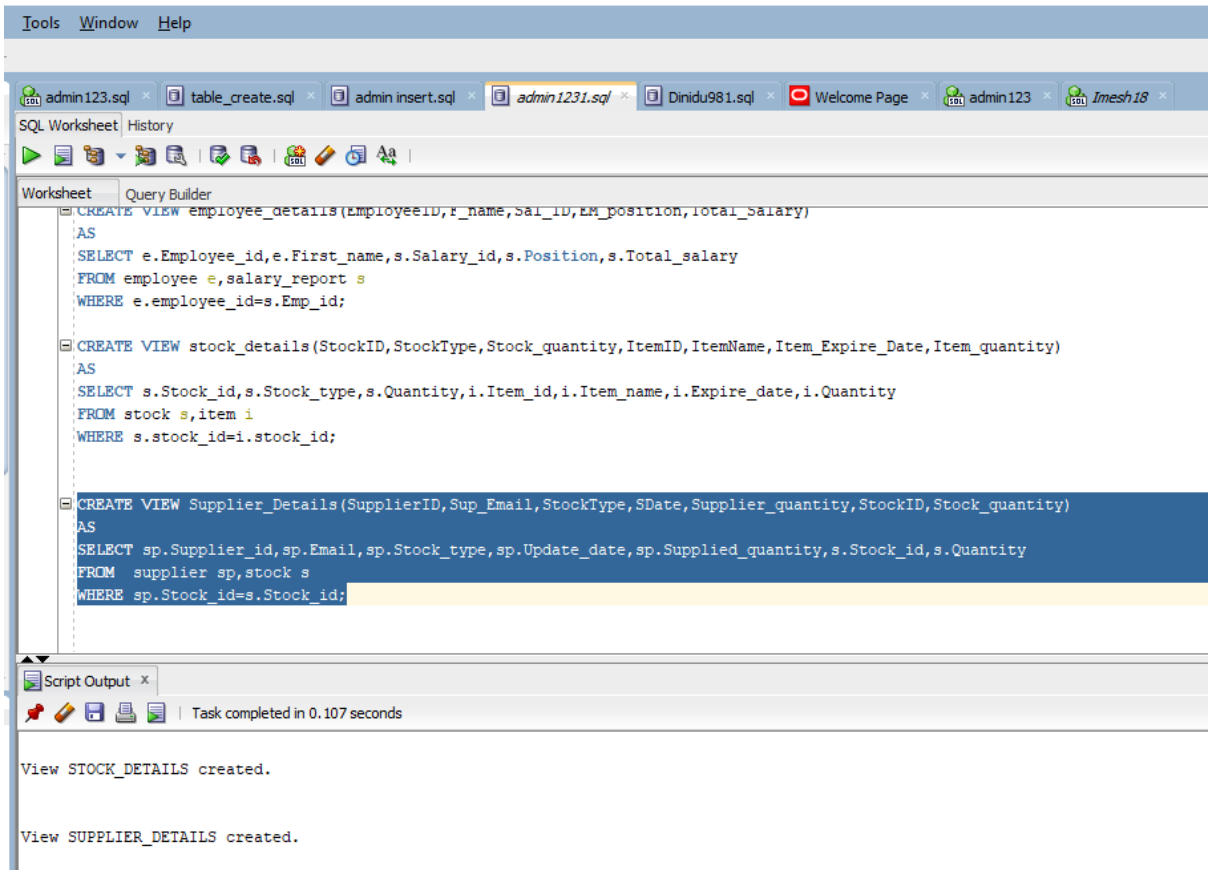

| | STOCKID | STOCKTYPE | STOCK_QUANTITY | ITEMID | ITEMNAME | ITEM_EXPIRE_DATE | ITEM_QUANTITY |
|---|---|---|---|---|---|---|---|
| 1 | SID001 | Biscuit | 500 | ITM001 | Chocolate biscuit 200g | 22-JAN-21 | 30 |
| 2 | SID001 | Biscuit | 500 | ITM002 | Lemonpuff 200g | 16-JAN-21 | 20 |
| 3 | SID002 | Fresh Milk | 80 | ITM003 | Fresh Milk 1L | 22-JUL-20 | 24 |
| 4 | SID003 | yogurt | 68 | ITM004 | Youghurt | 13-MAR-20 | 12 |
| 5 | SID004 | Sosages | 100 | ITM005 | Sosages 500g | 13-JAN-20 | 16 |
| 6 | SID005 | Sandwich bread | 38 | ITM006 | Sandwich bread | 28-AUG-20 | 50 |
| 7 | SID006 | Butter | 80 | ITM007 | Butter 500g | 04-DEC-20 | 40 |
| 8 | SID007 | Shampoo | 12 | ITM008 | Shampoo | 01-JAN-21 | 100 |
| 9 | SID008 | Handwash | 10 | ITM009 | Handwash | 21-FEB-22 | 30 |
| 10 | SID009 | Icecream | 40 | ITM010 | Icecream 2L | 23-APR-20 | 25 |
| 11 | SID010 | Milk Powder | 28 | ITM011 | Milk Powder 1kg | 16-JUN-20 | 28 |
| 12 | SID010 | Milk Powder | 28 | ITM012 | Milk Powder 400g | 25-DEC-20 | 45 |
| 13 | SID011 | Chicken | 12 | ITM013 | Chicken 1kg | 22-JAN-19 | 24 |
| 14 | SID012 | Pet food | 12 | ITM014 | Pet food 3kg | 22-JAN-19 | 34 |
| 15 | SID013 | vegitable oil | 20 | ITM015 | vegitable oil 1L | 22-JAN-19 | 48 |

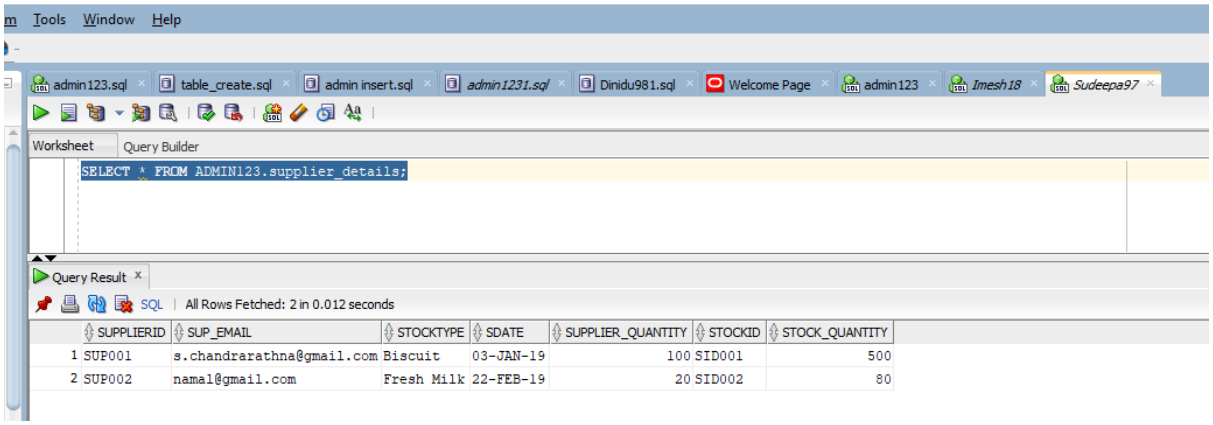
### 6.1.3 Create view supplier_details

Creating a View called “Supplier_Details by Admin and granting only “Select” permission to Store_keeper only to see the relevant fields.

**CREATE VIEW**
**Supplier_Details(SupplierID,Sup_Email,StockType,SDate,Supplier_quantity,StockID,Stock_quantity)**
**AS**
**SELECT**
**sp.Supplier_id,sp.Email,sp.Stock_type,sp.Update_date,sp.Supplied_quantity,s.Stock_id,s.Quantity**
**FROM  supplier sp,stock s**
**WHERE sp.Stock_id=s.Stock_id;**

admin1231.sql

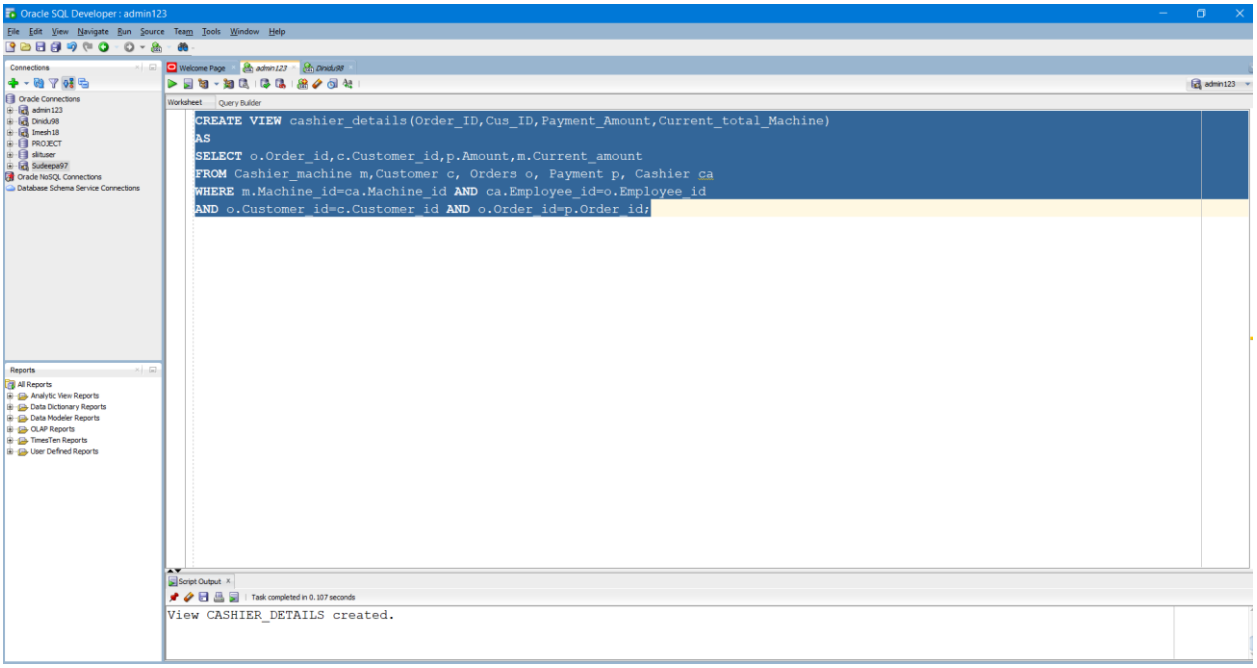





## 6.1.4 Create view cashier_details

Admin creates a view cashier_details, and it can view by the manager and cashier.

**CREATE VIEW**
**cashier_details(Order_ID,Cus_ID,Payment_Amount,Current_total_Machine)**
**AS**
**SELECT o.Order_id,c.Customer_id,p.Amount,m.Current_amount**
**FROM Cashier_machine m,Customer c, Orders o, Payment p, Cashier ca**
**WHERE m.Machine_id=ca.Machine_id AND ca.Employee_id=o.Employee_id**
**AND o.Customer_id=c.Customer_id AND o.Order_id=p.Order_id;**



Admin gives permission to it to roles.

**SQL> grant create session to manager;**

**Grant succeeded.**

**SQL> grant connect to manager;**

**Grant succeeded.**

**SQL> grant select on cashier_details to manager;**

**Grant succeeded.**

**SQL> grant manager to Dinidu98;**

**Grant succeeded.**

**SQL> grant create session to cashier;**

**Grant succeeded.**

**SQL> grant connect to cashier;**

**Grant succeeded.**

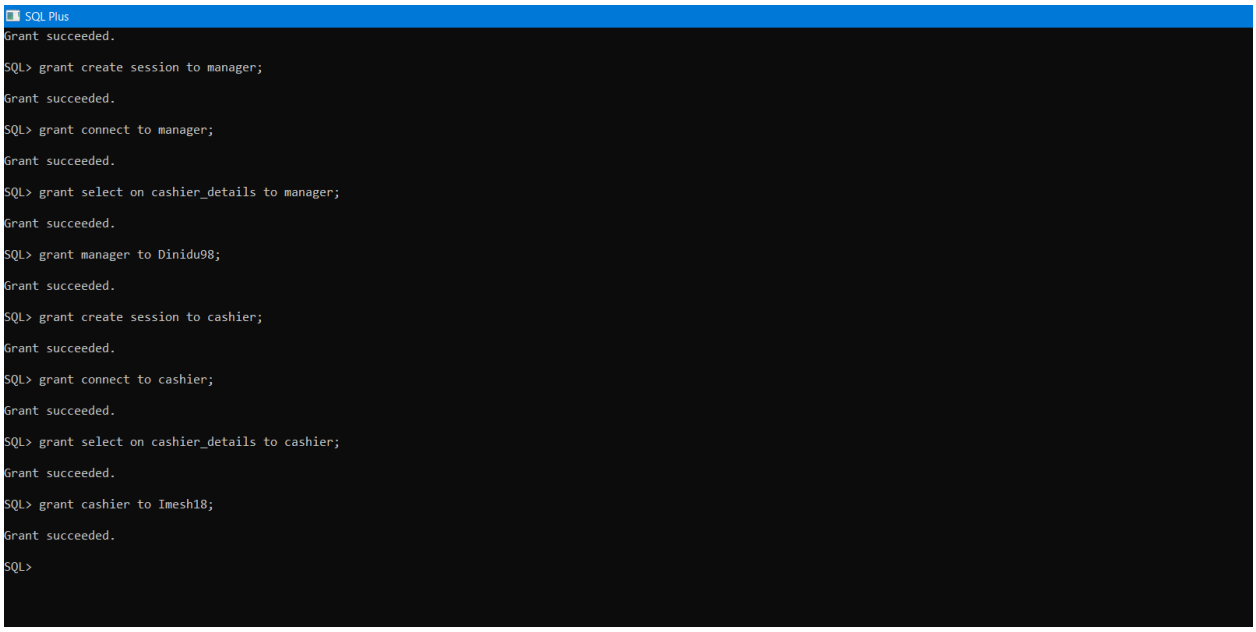
**SQL> grant select on cashier_details to cashier;**

**Grant succeeded.**

**SQL> grant cashier to Imesh18;**

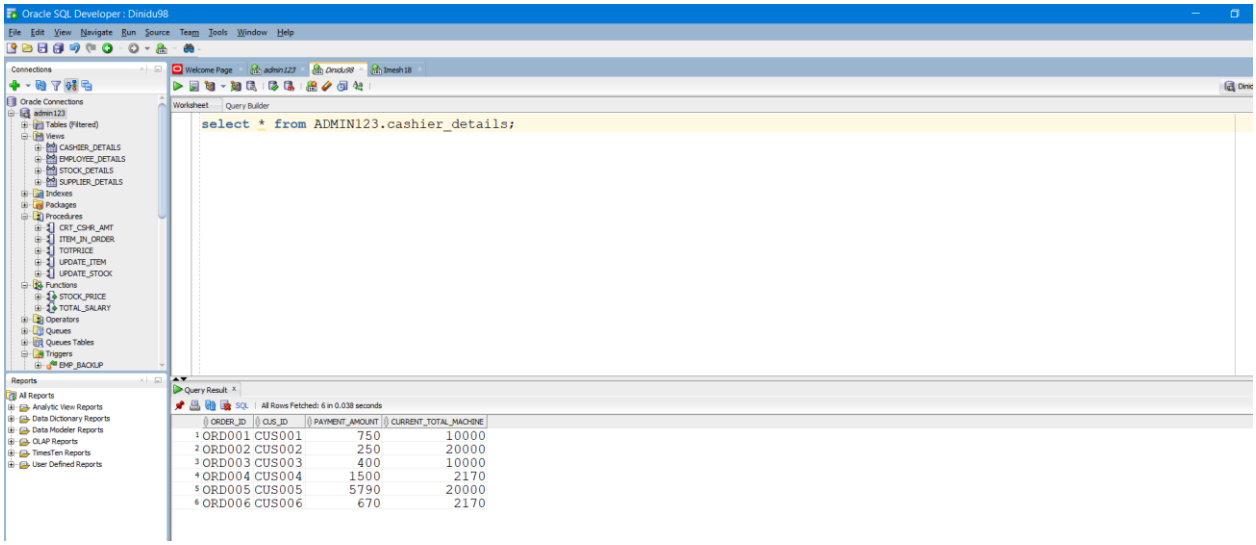
**Grant succeeded.**

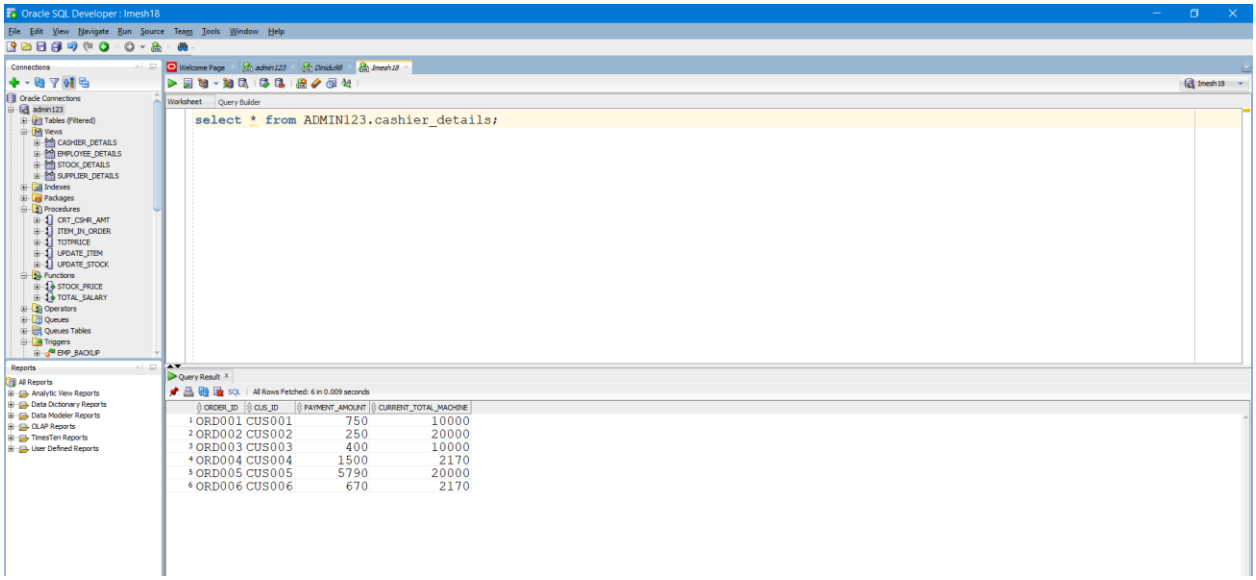
**SQL>**



The manager or cashier can select this view by the following command.

**select * from ADMIN123.cashier_details;**



## 6.2 Triggers

### 6.2.1 Create order_item_trigger

Assume customer order two orders and add and remove order items from the orders. We create an additional table column for the Orders table, which counts the line count of order items.

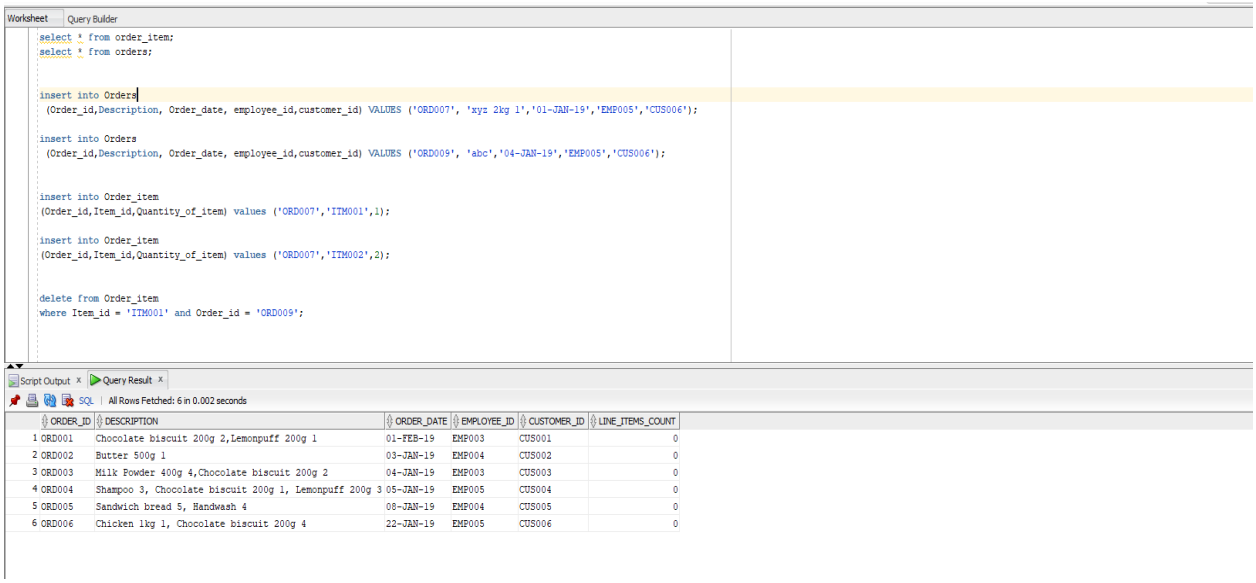
Login as admin123:

```
ALTER TABLE Orders
ADD Line_items_count NUMBER DEFAULT 0;
```

Create a trigger as order_item_trigger.

```
--create trigger order_item_trigger--
CREATE OR REPLACE TRIGGER order_items_trigger
  AFTER INSERT OR UPDATE OR DELETE ON Order_item
  FOR EACH ROW
BEGIN
  IF (INSERTING OR UPDATING)
  THEN
    UPDATE Orders SET line_items_count = NVL(line_items_count,0)+1
    WHERE Order_id = :new.Order_id;
  END IF;
  IF (DELETING OR UPDATING)
  THEN
    UPDATE Orders SET line_items_count = NVL(line_items_count,0)-1
    WHERE Order_id = :old.Order_id;
  END IF;
END;
```

Order table preview

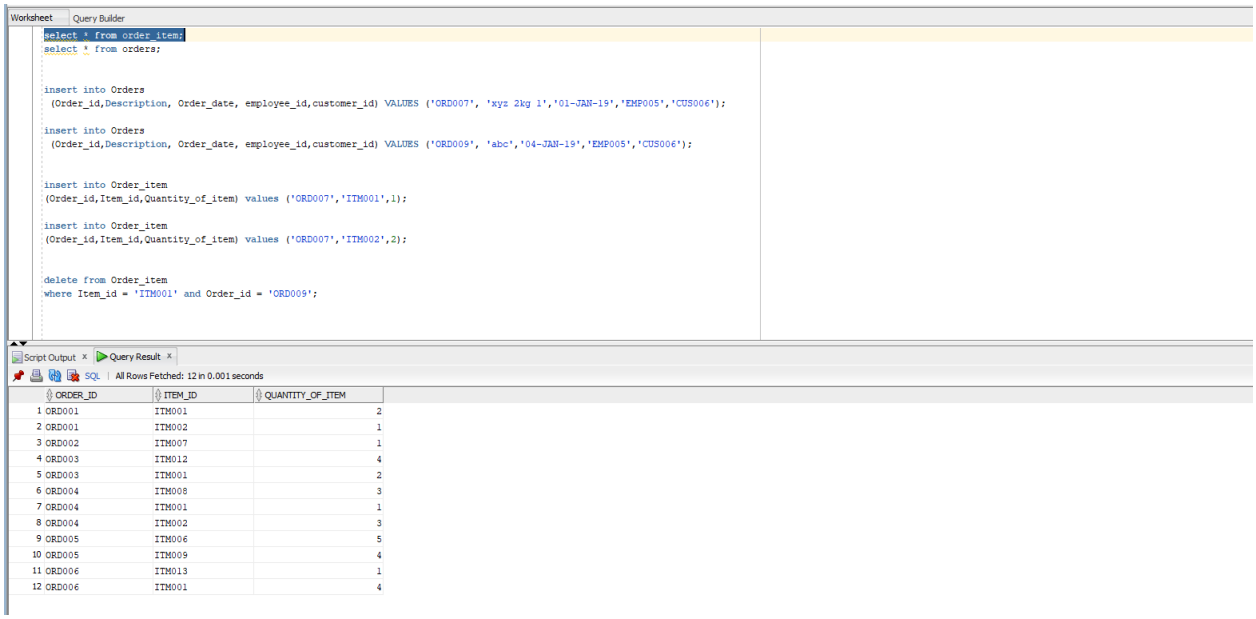


```
select * from order_item;
select * from orders;


insert into Orders
 (Order_id,Description, Order_date, employee_id,customer_id) VALUES ('ORD007', 'xyz 2kg 1','01-JAN-19','EMP005','CUS006');

insert into Orders
 (Order_id,Description, Order_date, employee_id,customer_id) VALUES ('ORD009', 'abc','04-JAN-19','EMP005','CUS006');


insert into Order_item
(Order_id,Item_id,Quantity_of_item) values ('ORD007','ITM001',1);

insert into Order_item
(Order_id,Item_id,Quantity_of_item) values ('ORD007','ITM002',2);


delete from Order_item
where Item_id = 'ITM001' and Order_id = 'ORD009';
```

Script Output | Query Result | All Rows Fetched: 6 in 0.002 seconds

| | ORDER_ID | DESCRIPTION | ORDER_DATE | EMPLOYEE_ID | CUSTOMER_ID | LINE_ITEMS_COUNT |
|---|---|---|---|---|---|---|
| 1 | ORD001 | Chocolate biscuit 200g 2,Lemonpuff 200g 1 | 01-FEB-19 | EMP003 | CUS001 | 0 |
| 2 | ORD002 | Butter 500g 1 | 03-JAN-19 | EMP004 | CUS002 | 0 |
| 3 | ORD003 | Milk Powder 400g 4,Chocolate biscuit 200g 2 | 04-JAN-19 | EMP003 | CUS003 | 0 |
| 4 | ORD004 | Shampoo 3, Chocolate biscuit 200g 1, Lemonpuff 200g 3 | 05-JAN-19 | EMP005 | CUS004 | 0 |
| 5 | ORD005 | Sandwich bread 5, Handwash 4 | 08-JAN-19 | EMP004 | CUS005 | 0 |
| 6 | ORD006 | Chicken 1kg 1, Chocolate biscuit 200g 4 | 22-JAN-19 | EMP005 | CUS006 | 0 |

Order_items table preview



```
select * from order_item;
select * from orders;


insert into Orders
 (Order_id,Description, Order_date, employee_id,customer_id) VALUES ('ORD007', 'xyz 2kg 1','01-JAN-19','EMP005','CUS006');

insert into Orders
 (Order_id,Description, Order_date, employee_id,customer_id) VALUES ('ORD009', 'abc','04-JAN-19','EMP005','CUS006');


insert into Order_item
(Order_id,Item_id,Quantity_of_item) values ('ORD007','ITM001',1);

insert into Order_item
(Order_id,Item_id,Quantity_of_item) values ('ORD007','ITM002',2);


delete from Order_item
where Item_id = 'ITM001' and Order_id = 'ORD009';
```

Script Output | Query Result | All Rows Fetched: 12 in 0.001 seconds

| | ORDER_ID | ITEM_ID | QUANTITY_OF_ITEM |
|---|---|---|---|
| 1 | ORD001 | ITM001 | 2 |
| 2 | ORD001 | ITM002 | 1 |
| 3 | ORD002 | ITM007 | 1 |
| 4 | ORD003 | ITM012 | 4 |
| 5 | ORD003 | ITM001 | 2 |
| 6 | ORD004 | ITM008 | 3 |
| 7 | ORD004 | ITM001 | 1 |
| 8 | ORD004 | ITM002 | 3 |
| 9 | ORD005 | ITM006 | 5 |
| 10 | ORD005 | ITM009 | 4 |
| 11 | ORD006 | ITM013 | 1 |
| 12 | ORD006 | ITM001 | 4 |

Give them permission to Cashier.

**SQL> conn Imesh18**

**Enter password:**

**Connected.**

**SQL>grant create session cashier;**

**Grant succeeded.**

**SQL>grant connect to cashier;**

**Grant succeeded.**

**SQL>grant insert, update, delete, select on Orders to cashier;**

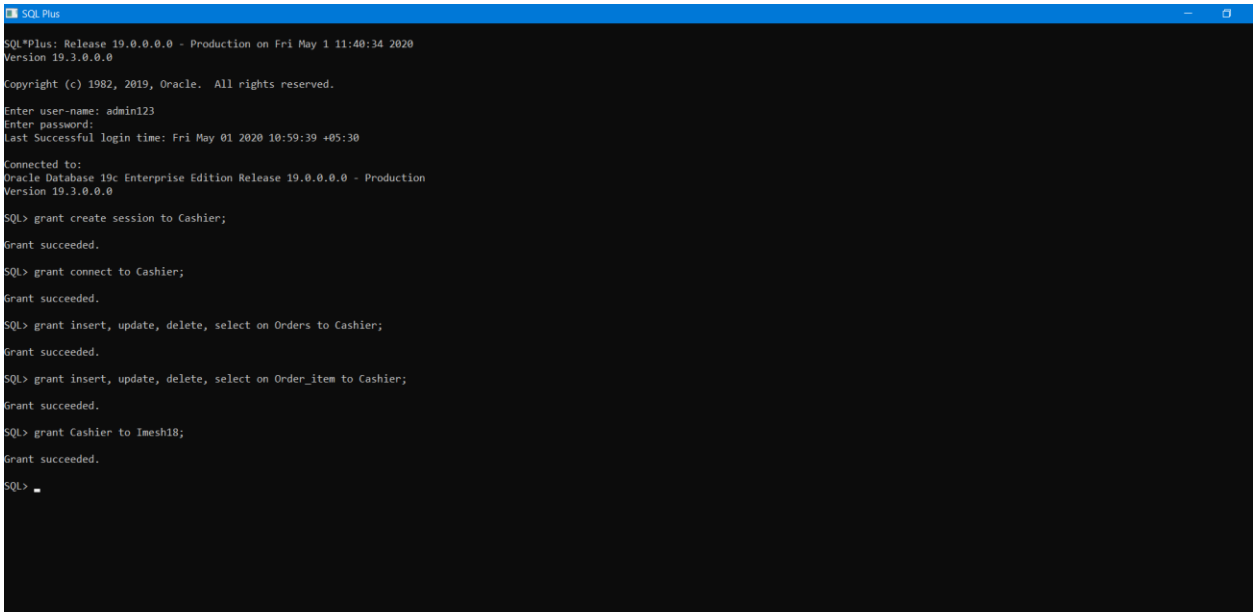
**Grant succeeded.**

**SQL>grant insert, update, delete, select on Order_item to cashier;**

**Grant succeeded.**

**SQL>grant cashier to Imesh18;**

**Grant succeeded.**




Insert new records to Orders table as ORD007 and ORD009.

**INSERT INTO ADMIN123.orders**

**(Order_id,Description, Order_date, employee_id,customer_id) VALUES ('ORD007', 'xyz 2kg 1','01-JAN-19','EMP005','CUS006');**

Using ID ORD007, the customer makes an order. The customer does not have products in the order at this stage. The mechanism is not triggered because no action is taken on the Order item list.

**INSERT INTO ADMIN123.orders**
**(Order_id,Description, Order_date, employee_id,customer_id) VALUES ('ORD009', 'abc 500g 1','04-JAN-19','EMP005','CUS006');**

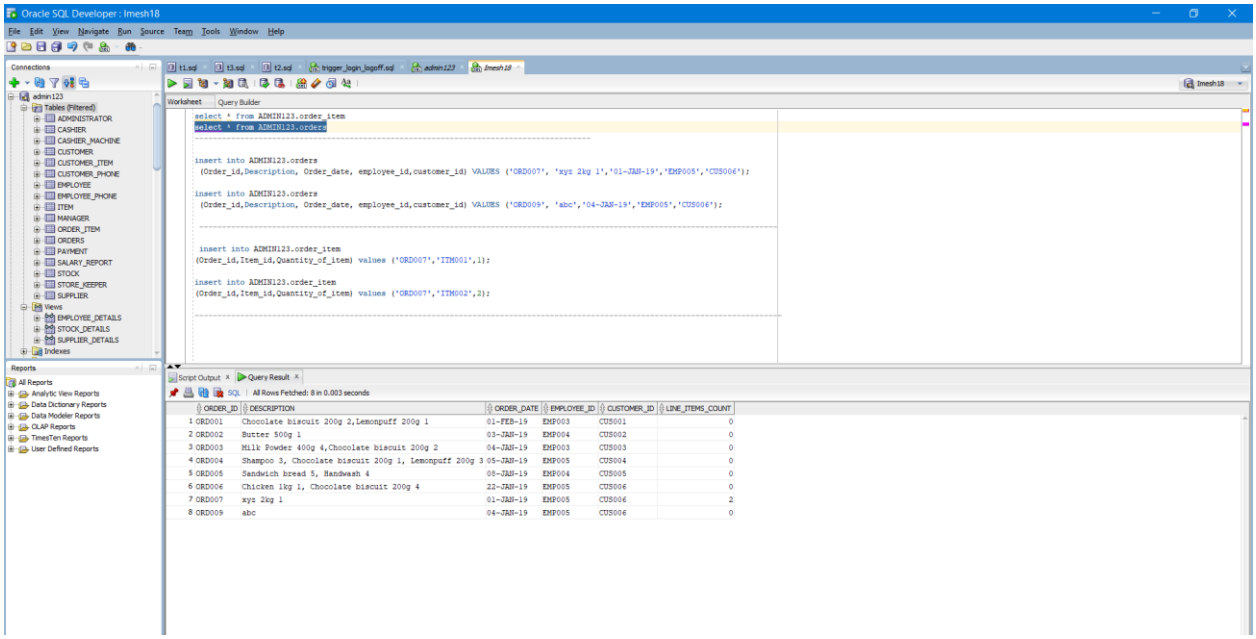
The customer creates another separate order with ID ORD009. The mechanism is not triggered because no action is taken on the Order item list.
After inserting the data to Order_item table.

**INSERT INTO ADMIN123.order_item**
**(Order_id,Item_id,Quantity_of_item) values ('ORD007','ITM001',1);**

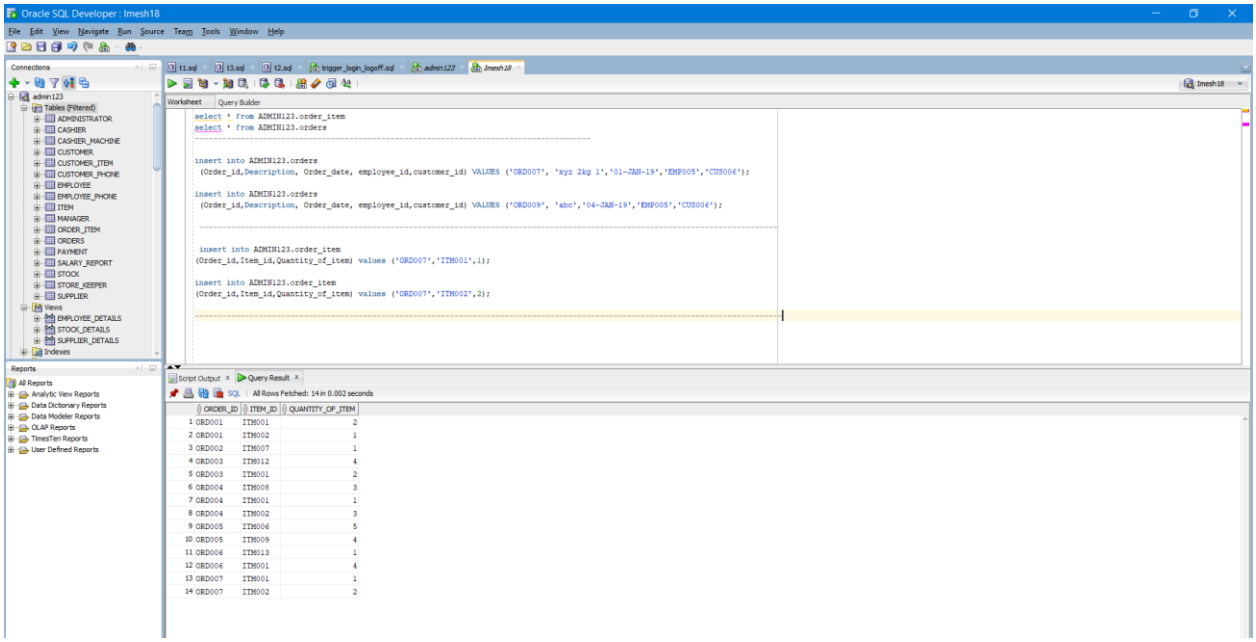
In order ORD007, the customer adds an object. The INSERT calls on the computer. The declaration caused raises the object count from 0 to 1.

**INSERT INTO ADMIN123.order_item**
**(Order_id,Item_id,Quantity_of_item) values ('ORD007','ITM002',2);**

The customer adds another additional item to ORD007, and the count will increase by one, and it raises from 1 to 2.




The customer asks about both orders' status. Two products are included in order ORD007. No products are included in order ORD009.

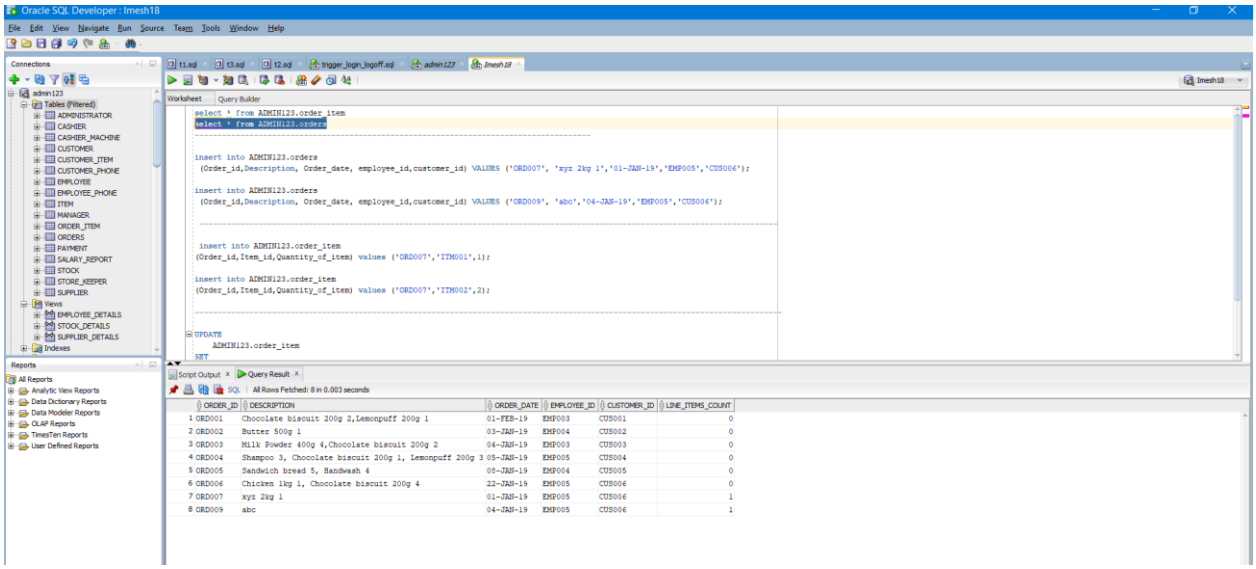


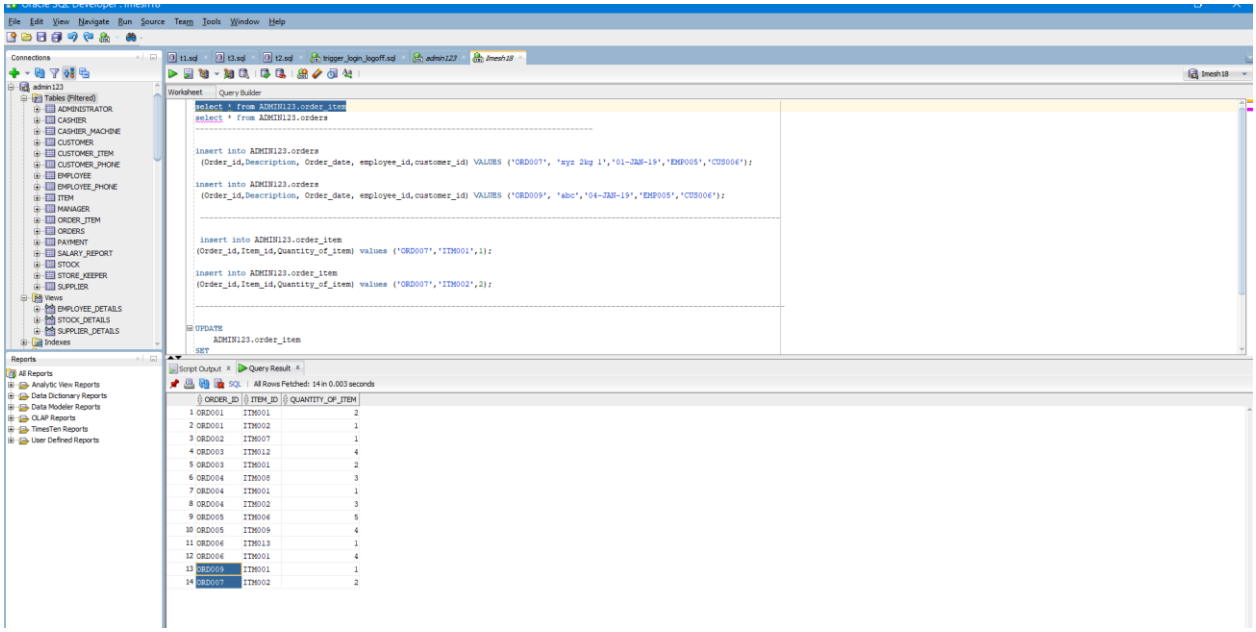
The customer asks about the order items status. The order ID and the quantity of the item are individually specified for every element.

The customer moves the products on the ORD007 to ORD009.

In the order item table, the UPDATE declaration changes two rows, invoking the trigger once for each line.

```
UPDATE
   ADMIN123.order_item
SET
   Order_id = 'ORD009',
   Item_id = 'ITM001'
WHERE
   Order_id = 'ORD007' AND
   Item_id = 'ITM001';
```





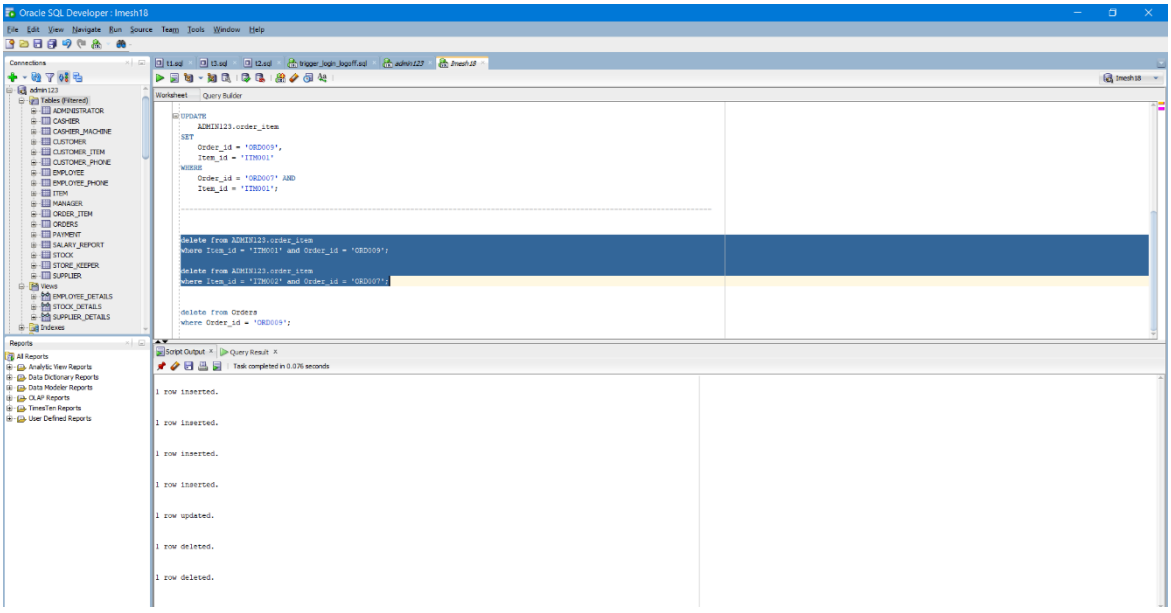
Now the customer removes all the order_items from all orders.

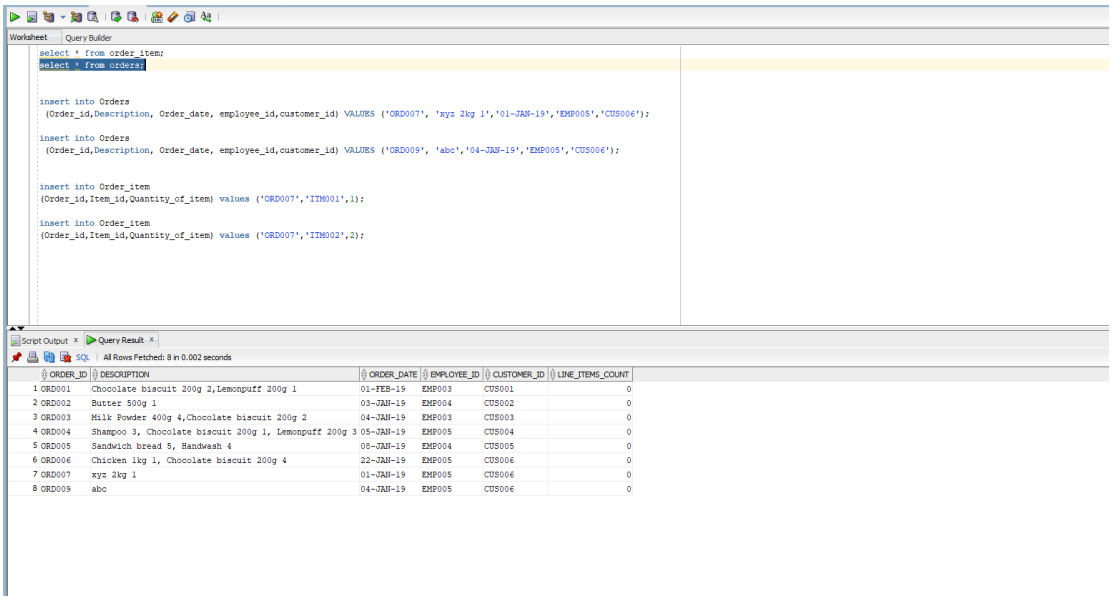
**DELETE from ADMIN123.order_item**

**WHERE Item_id = 'ITM001' and Order_id = 'ORD009';**

**DELETE from ADMIN123.order_item**

**WHERE Item_id = 'ITM002' and Order_id = 'ORD007';**



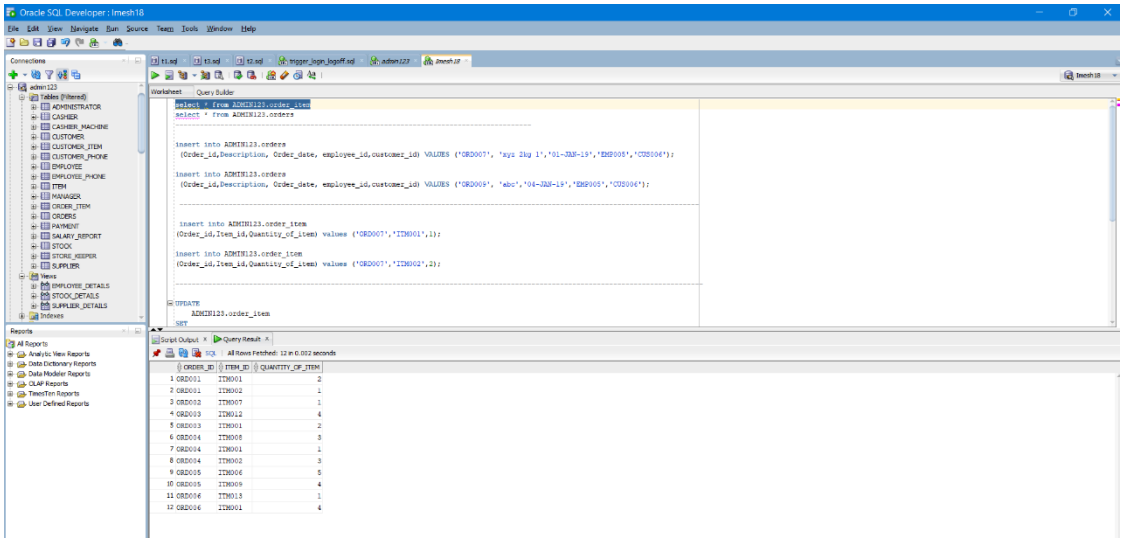


```
select * from order_item;
select * from orders;


insert into Orders
 (Order_id,Description, Order_date, employee_id,customer_id) VALUES ('ORD007', 'xyz 2kg 1','01-JAN-19','EMP005','CUS006');

insert into Orders
 (Order_id,Description, Order_date, employee_id,customer_id) VALUES ('ORD009', 'abc','04-JAN-19','EMP005','CUS006');


insert into Order_item
(Order_id,Item_id,Quantity_of_item) values ('ORD007','ITM001',1);

insert into Order_item
(Order_id,Item_id,Quantity_of_item) values ('ORD007','ITM002',2);
```

Script Output | Query Result

All Rows Fetched: 8 in 0.002 seconds

| | ORDER_ID | DESCRIPTION | ORDER_DATE | EMPLOYEE_ID | CUSTOMER_ID | LINE_ITEMS_COUNT |
|---|---|---|---|---|---|---|
| 1 | ORD001 | Chocolate biscuit 200g 2,Lemonpuff 200g 1 | 01-FEB-19 | EMP003 | CUS001 | 0 |
| 2 | ORD002 | Butter 500g 1 | 03-JAN-19 | EMP004 | CUS002 | 0 |
| 3 | ORD003 | Milk Powder 400g 4,Chocolate biscuit 200g 2 | 04-JAN-19 | EMP003 | CUS003 | 0 |
| 4 | ORD004 | Shampoo 3, Chocolate biscuit 200g 1, Lemonpuff 200g 3 | 05-JAN-19 | EMP005 | CUS004 | 0 |
| 5 | ORD005 | Sandwich bread 5, Handwash 4 | 08-JAN-19 | EMP004 | CUS005 | 0 |
| 6 | ORD006 | Chicken 1kg 1, Chocolate biscuit 200g 4 | 22-JAN-19 | EMP005 | CUS006 | 0 |
| 7 | ORD007 | xyz 2kg 1 | 01-JAN-19 | EMP005 | CUS006 | 0 |
| 8 | ORD009 | abc | 04-JAN-19 | EMP005 | CUS006 | 0 |



Oracle SQL Developer : lmesh18

```
select * from ADMIN123.order_item
select * from ADMIN123.orders


insert into ADMIN123.orders
 (Order_id,Description, Order_date, employee_id,customer_id) VALUES ('ORD007', 'xyz 2kg 1','01-JAN-19','EMP005','CUS006');

insert into ADMIN123.orders
 (Order_id,Description, Order_date, employee_id,customer_id) VALUES ('ORD009', 'abc','04-JAN-19','EMP005','CUS006');


 insert into ADMIN123.order_item
(Order_id,Item_id,Quantity_of_item) values ('ORD007','ITM001',1);

insert into ADMIN123.order_item
(Order_id,Item_id,Quantity_of_item) values ('ORD007','ITM002',2);


UPDATE
    ADMIN123.order_item
  SET
```

| | ORDER_ID | ITEM_ID | QUANTITY_OF_ITEM |
|---|---|---|---|
| 1 | ORD001 | ITM001 | 2 |
| 2 | ORD001 | ITM002 | 1 |
| 3 | ORD002 | ITM007 | 1 |
| 4 | ORD003 | ITM012 | 4 |
| 5 | ORD003 | ITM001 | 2 |
| 6 | ORD004 | ITM008 | 3 |
| 7 | ORD004 | ITM001 | 1 |
| 8 | ORD004 | ITM002 | 3 |
| 9 | ORD005 | ITM006 | 5 |
| 10 | ORD005 | ITM009 | 4 |
| 11 | ORD006 | ITM013 | 1 |
| 12 | ORD006 | ITM001 | 4 |

### 6.2.2 Create login_trigger and logoff_trigger

Create a trigger to give access time and action admin logged in to the admin account. First, we alter the Administrator table and create a separate table to provide access details. The administrator table will give the details of the admin user, such as Admin id, Name of admin, etc.

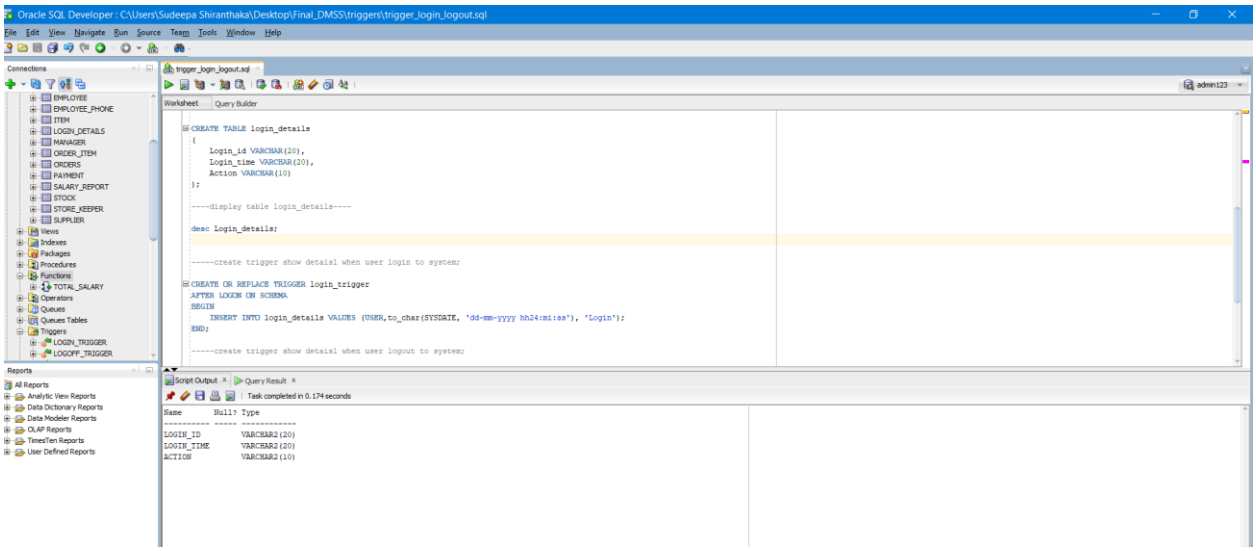
```
SELECT * FROM Administrator;
ALTER TABLE Administrator
DROP COLUMN Access_time;

ALTER TABLE Administrator
DROP COLUMN Modify_time;

ALTER TABLE Administrator
ADD Admin_name VARCHAR(20);

UPDATE Administrator
SET Admin_name = 'admin123'
WHERE Admin_id = 'ADM001';
```

Now we create a new table to give login details and after create the triggers.

```
CREATE TABLE login_details
(
   Login_id VARCHAR(20),
   Login_time VARCHAR(20),
   Action VARCHAR(10)
);
```

----display table login_details----

desc Login_details;




--create trigger show details when user login to system--

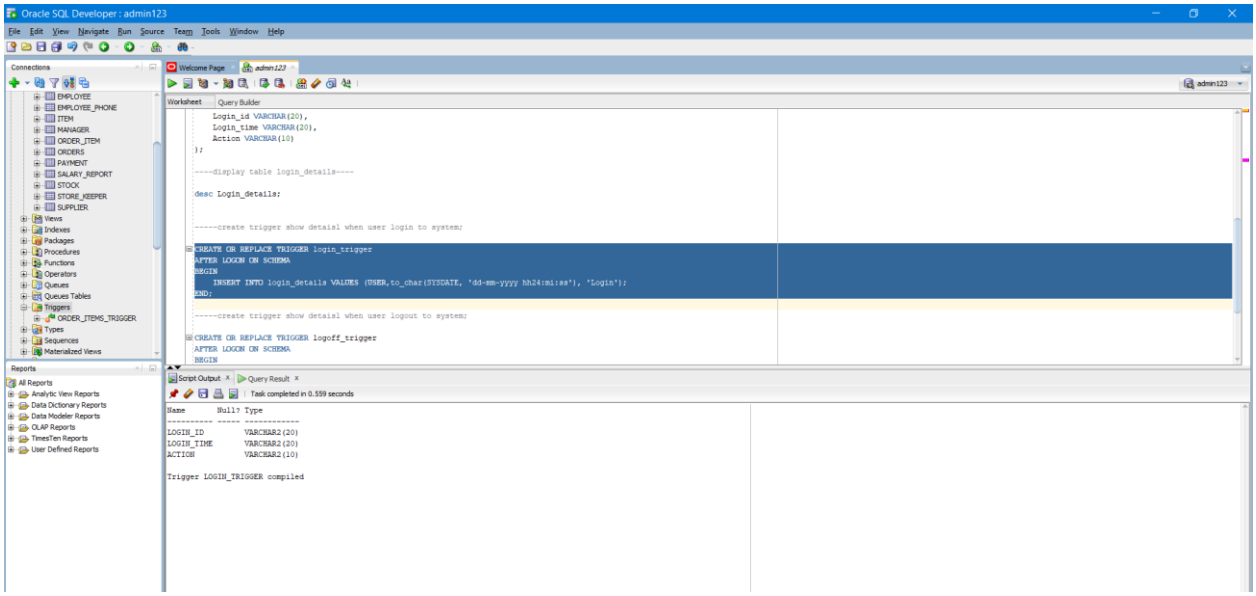
**CREATE OR REPLACE TRIGGER login_trigger**

**AFTER LOGON ON SCHEMA**

**BEGIN**

**INSERT INTO login_details VALUES (USER,to_char(SYSDATE, 'dd-mm-yyyy hh24:mi:ss'), 'Login');**

**END;**



--create trigger show details when user logout to system--

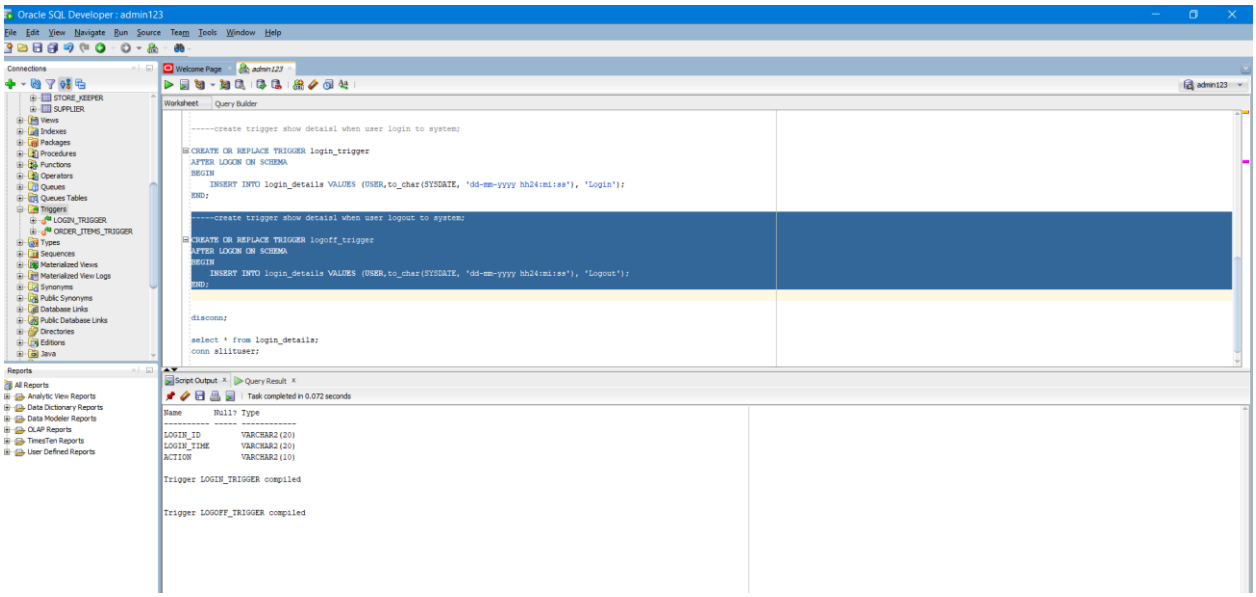
**CREATE OR REPLACE TRIGGER logoff_trigger**

**AFTER LOGON ON SCHEMA**

**BEGIN**

**INSERT INTO login_details VALUES (USER,to_char(SYSDATE, 'dd-mm-yyyy hh24:mi:ss'), 'Logout');**

**END;**

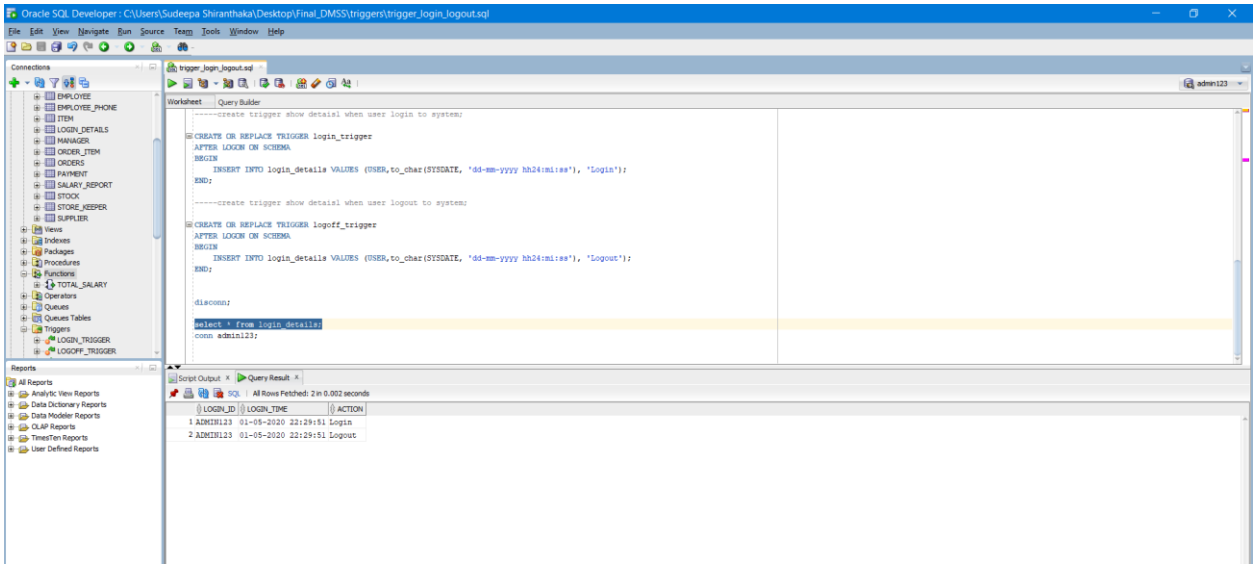


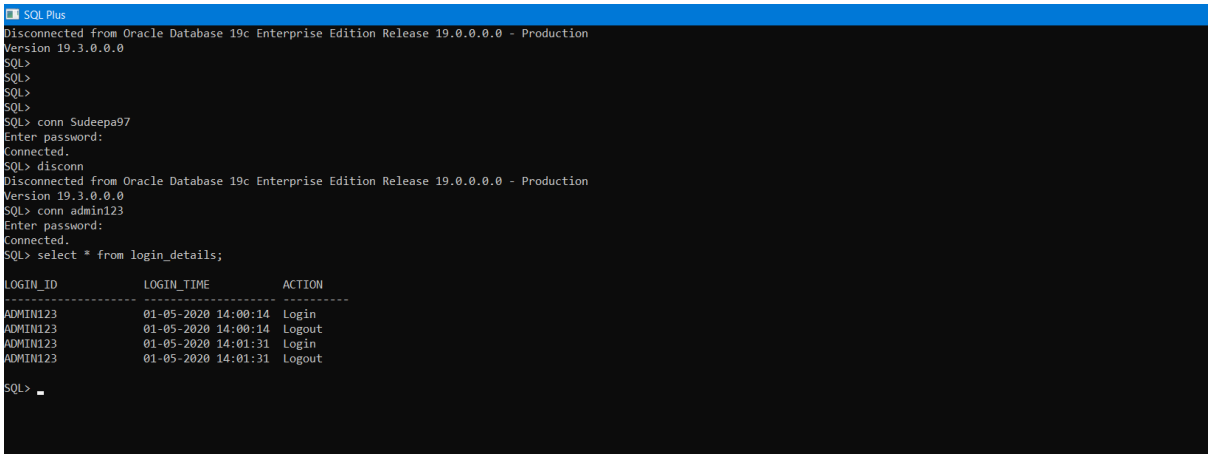

After creating the trigger for login action and logout action, then disconnect the connection and log in again.

**disconn;**

**conn admin123;**

**select * from login_details;**


Oracle SQL Developer : C:\Users\Sudeepa Shiranthaka\Desktop\Final_DMSS\triggers\trigger_login_logout.sql
File Edit View Navigate Run Source Team Tools Window Help
Connections
EMPLOYEE
EMPLOYEE_PHONE
ITEM
LOGIN_DETAILS
MANAGER
ORDER_ITEM
ORDERS
PAYMENT
SALARY_REPORT
STOCK
STORE_KEEPER
SUPPLIER
Views
Indexes
Packages
Procedures
Functions
TOTAL_SALARY
Operators
Queues
Queues Tables
Triggers
LOGIN_TRIGGER
LOGOFF_TRIGGER
Reports
All Reports
Analytic View Reports
Data Dictionary Reports
Data Modeler Reports
OLAP Reports
TimesTen Reports
User Defined Reports
trigger_login_logout.sql
admin123
Worksheet
Query Builder
-----create trigger show details1 when user login to system;
CREATE OR REPLACE TRIGGER login_trigger
AFTER LOGON ON SCHEMA
BEGIN
INSERT INTO login_details VALUES (USER,to_char(SYSDATE, 'dd-mm-yyyy hh24:mi:ss'), 'Login');
END;
-----create trigger show details1 when user logout to system;
CREATE OR REPLACE TRIGGER logoff_trigger
AFTER LOGON ON SCHEMA
BEGIN
INSERT INTO login_details VALUES (USER,to_char(SYSDATE, 'dd-mm-yyyy hh24:mi:ss'), 'Logout');
END;
disconn;
select * from login_details;
conn admin123;
Script Output
Query Result
All Rows Fetched: 2 in 0.002 seconds
LOGIN_ID
LOGIN_TIME
ACTION
1 ADMIN123 01-05-2020 22:29:51 Login
2 ADMIN123 01-05-2020 22:29:51 Logout


SQL Plus
Disconnected from Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production
Version 19.3.0.0.0
SQL>
SQL>
SQL>
SQL>
SQL> conn Sudeepa97
Enter password:
Connected.
SQL> disconn
Disconnected from Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production
Version 19.3.0.0.0
SQL> conn admin123
Enter password:
Connected.
SQL> select * from login_details;
LOGIN_ID LOGIN_TIME ACTION
ADMIN123 01-05-2020 14:00:14 Login
ADMIN123 01-05-2020 14:00:14 Logout
ADMIN123 01-05-2020 14:01:31 Login
ADMIN123 01-05-2020 14:01:31 Logout
SQL>

## 6.3 Functions

### 6.3.1 Create total_salary function

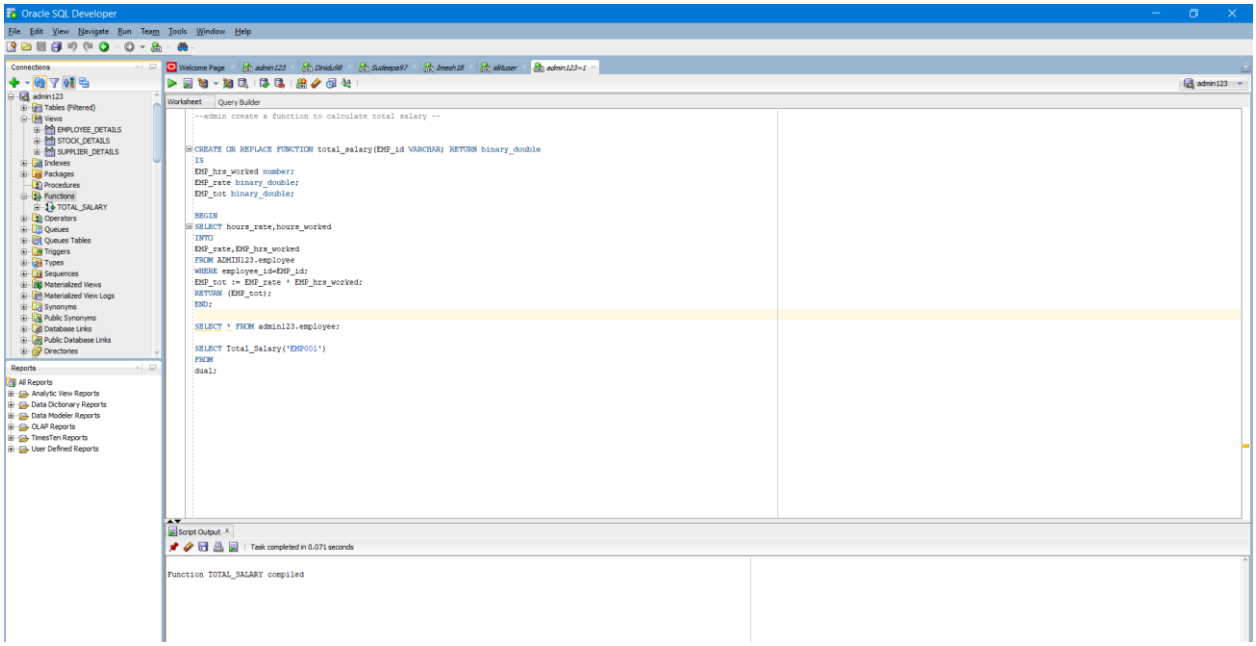
Admin create a function to calculate the total salary of any employee and give permission to execute the total_salary function to the manager (Dinidu98)

Login as admin123:

**SQL> conn admin123**
**Enter password:**
**Connected.**
**SQL>**

The following command is to create a function Total_salary.

```
CREATE OR REPLACE FUNCTION total_salary(EMP_id VARCHAR) RETURN
binary_double
IS
EMP_hrs_worked number;
EMP_rate binary_double;
EMP_tot binary_double;

BEGIN
SELECT hours_rate,hours_worked
INTO
EMP_rate,EMP_hrs_worked
FROM ADMIN123.employee
WHERE employee_id=EMP_id;
EMP_tot := EMP_rate * EMP_hrs_worked;
RETURN (EMP_tot);
END ;
```




Login as admin123:

**SQL> conn admin123.**

**Enter password:**

**Connected.**

**SQL>grant create session to manager;**

**Grant succeeded.**

**SQL>grant connect to manager;**

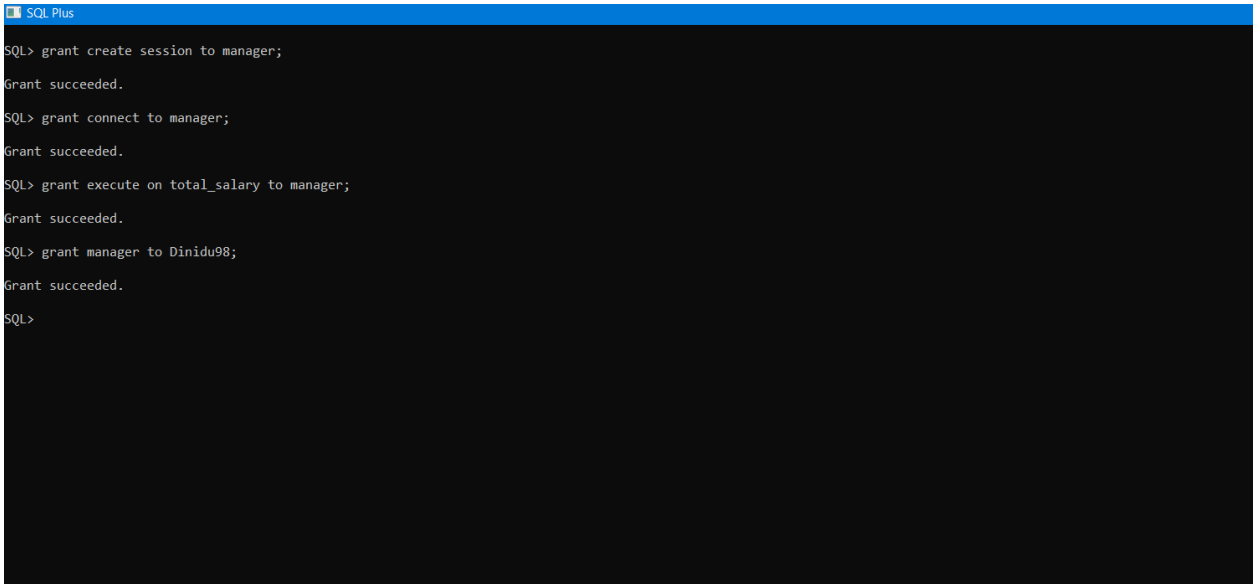
**Grant succeeded.**

**SQL>grant execute on total_salary to manager;**

**Grant succeeded.**

**SQL>grant manager to Dinidu98;**

**Grant succeeded.**




Now the manager can execute total_salary function by,

**SELECT admin123.total_salary('EMP001')**
**FROM DUAL;**

### 6.3.2 Create stock_price function

Admin creates a function and gives grant the privilege to the manger. The manager enters item_id and views the total price of all quantity in each item.

**CREATE OR REPLACE FUNCTION stock_price(itemID CHAR) RETURN BINARY_FLOAT**
**AS**
**item_price BINARY_FLOAT;**
**item_quantity NUMBER;**
**tot_price BINARY_FLOAT;**

**BEGIN**

**SELECT Price,Quantity**
**INTO**

**item_price,item_quantity**

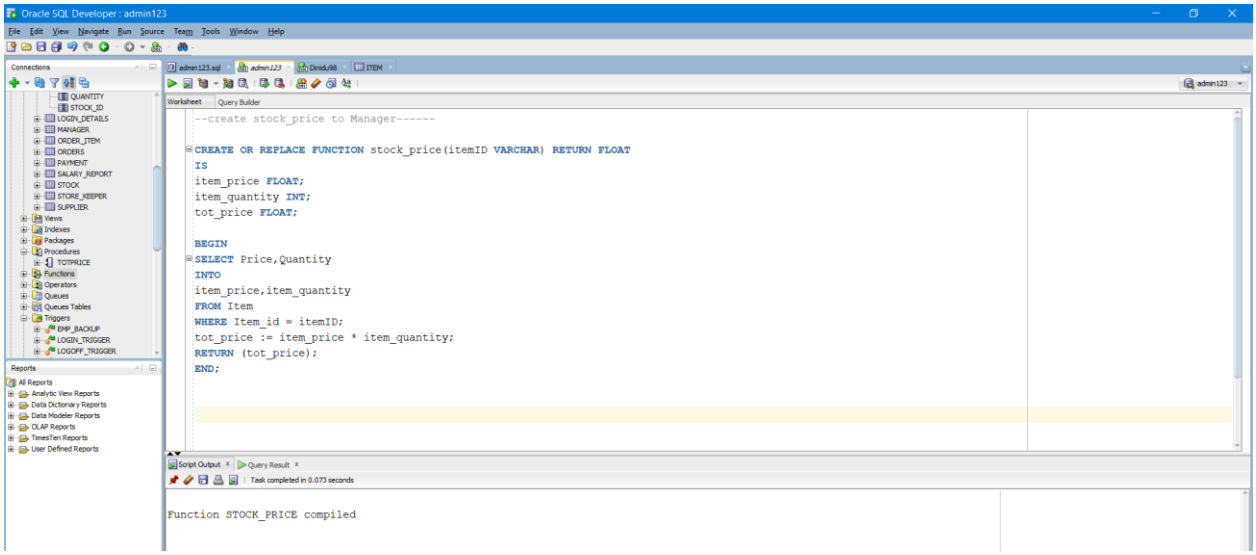
**FROM ADMIN123.item**

**WHERE Item_id = itemID;**

**tot_price := item_price * item_quantity;**

**RETURN (tot_price);**

**END;**




Grant the permission to execute stock_price function.

**SQL> grant create session to Manager;**

**Grant succeeded.**

**SQL> grant connect to Manager;**

**Grant succeeded.**

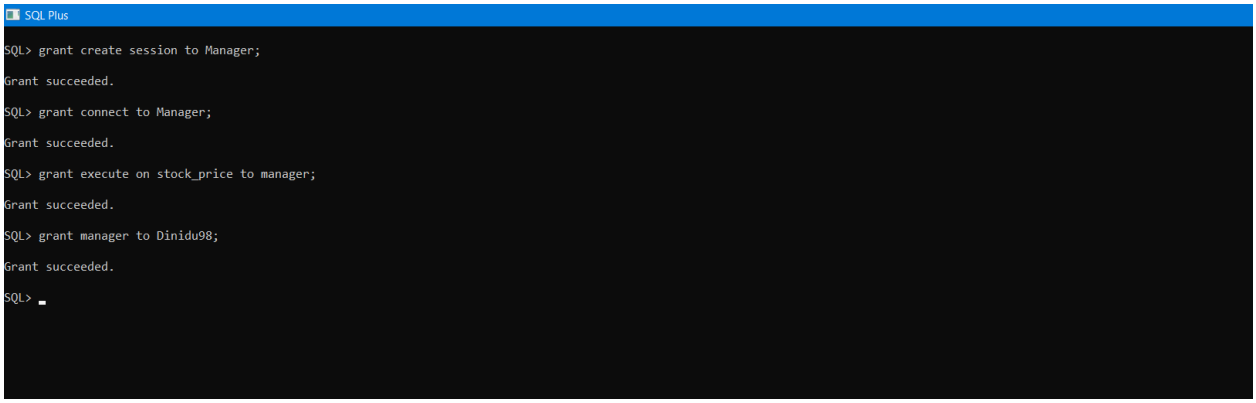
**SQL> grant execute on stock_price to manager;**

**Grant succeeded.**

**SQL> grant manager to Dinidu98;**

**Grant succeeded.**

**SQL>**

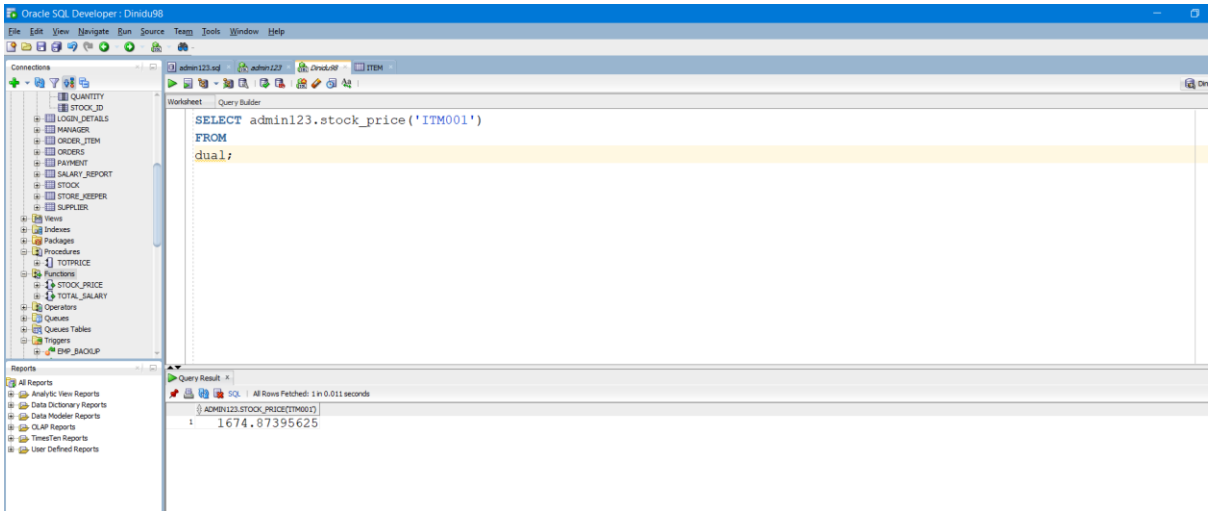



Manager Dinidu98 can execute the function.

**SELECT admin123.stock_price('ITM008')**

**FROM**

**dual;**

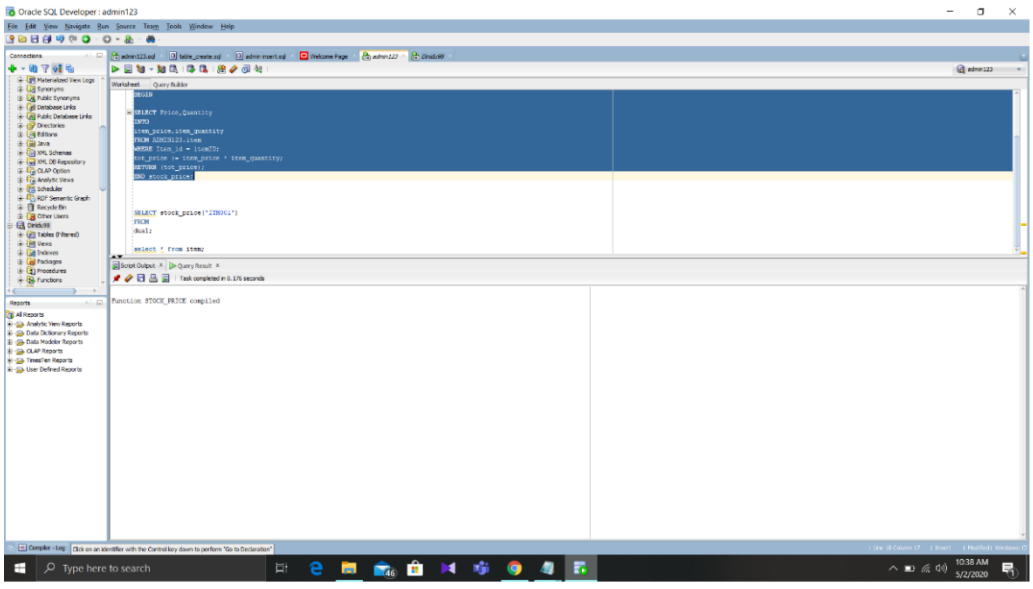


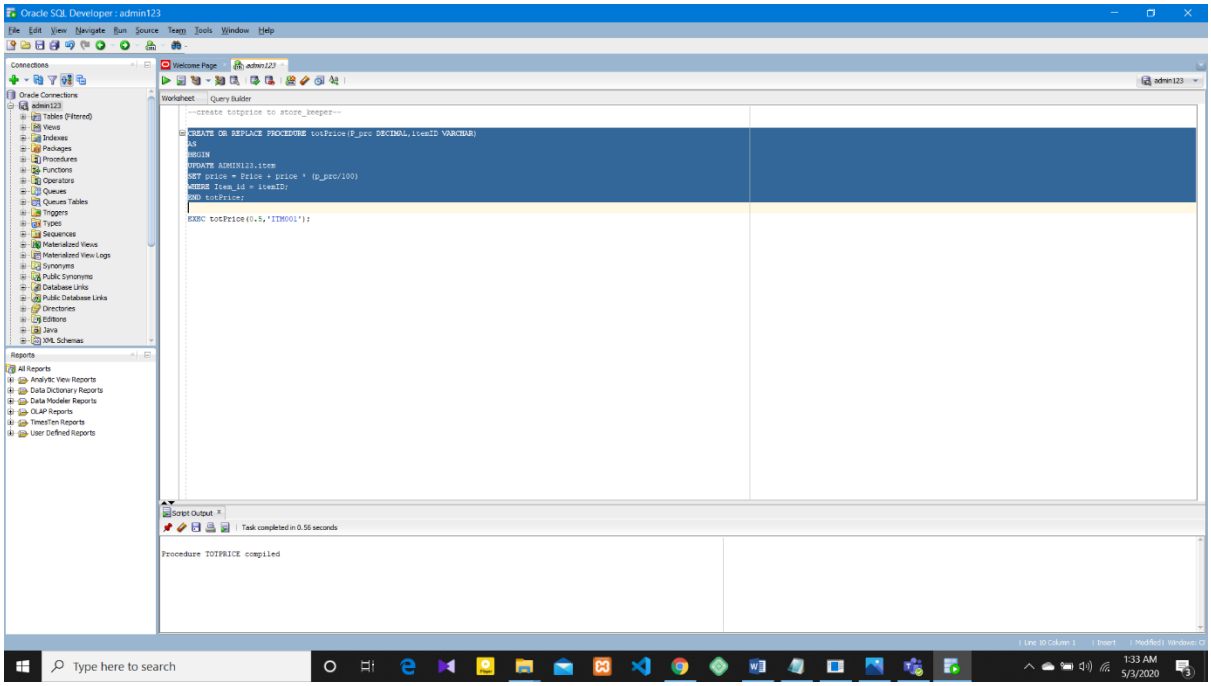
## 6.4 Procedures

### 6.4.1 Create Procedure totPrice

Admin creates a procedure and grants the execute privilege to Store_keeper. Store_keeper enter each item_id and change item price.

```
--create totprice to store_keeper--
CREATE OR REPLACE PROCEDURE totPrice(P_prc DECIMAL,itemID
VARCHAR)
AS
BEGIN
UPDATE ADMIN123.item
SET price = Price + price * (p_prc/100)
WHERE Item_id = itemID;
END totPrice;
```






Grant the permission to execute Procedure totPrice

**SQL> conn admin123**

**Enter password:**

**Connected.**

**SQL> grant create session to Store_keeper;**

**Grant succeeded.**

**SQL> grant connect to Store_keeper;**

**Grant succeeded.**

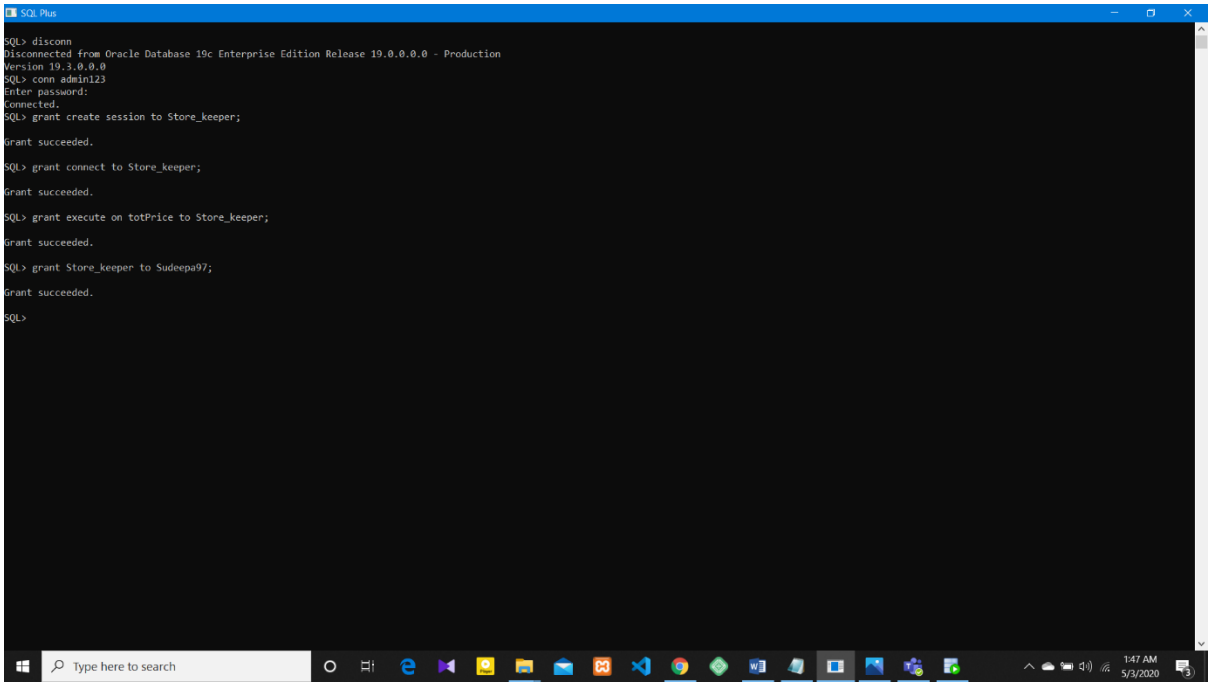
**SQL> grant execute on totPrice to Store_keeper;**

**Grant succeeded.**

**SQL> grant Store_keeper to Sudeepa97;**

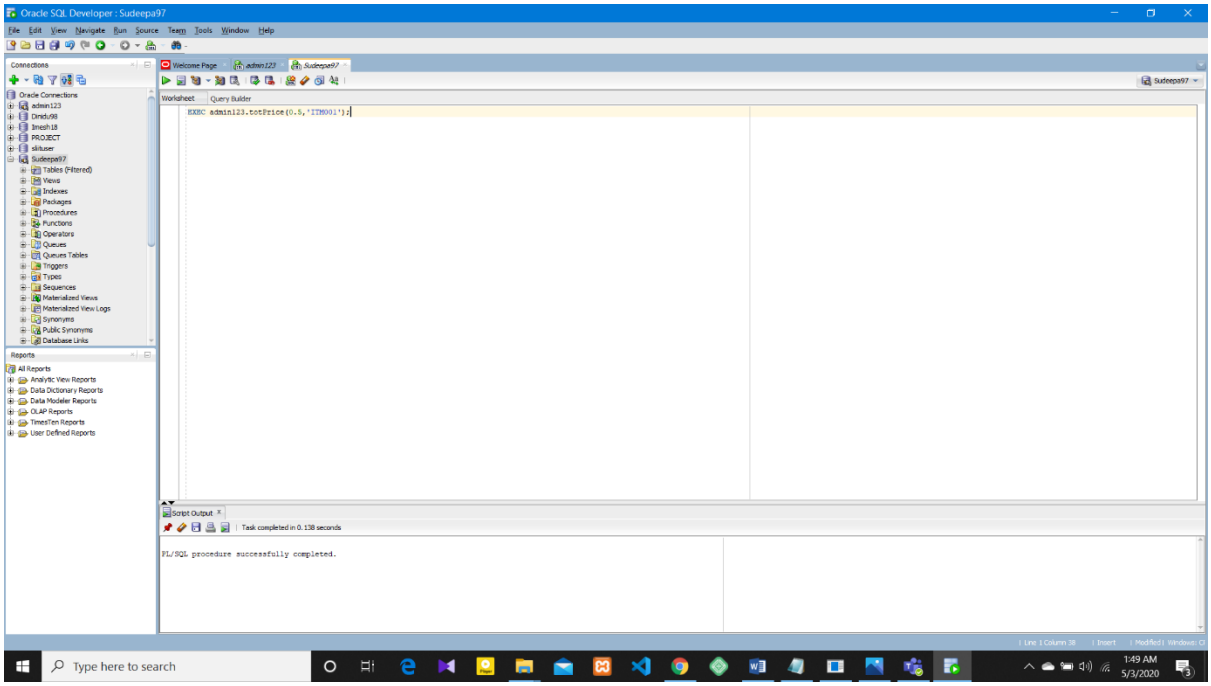
**Grant succeeded.**

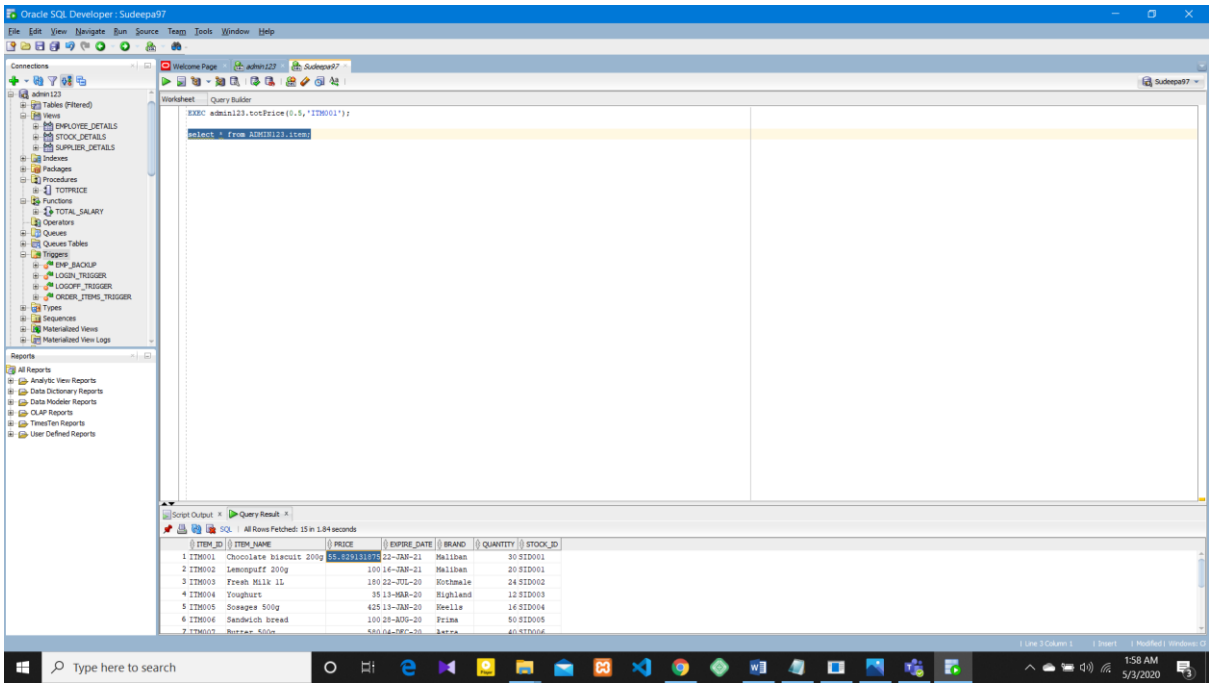
**SQL>**




Storekeeper execute the procedure

**EXEC admin123.totPrice(0.5,'ITM001');**

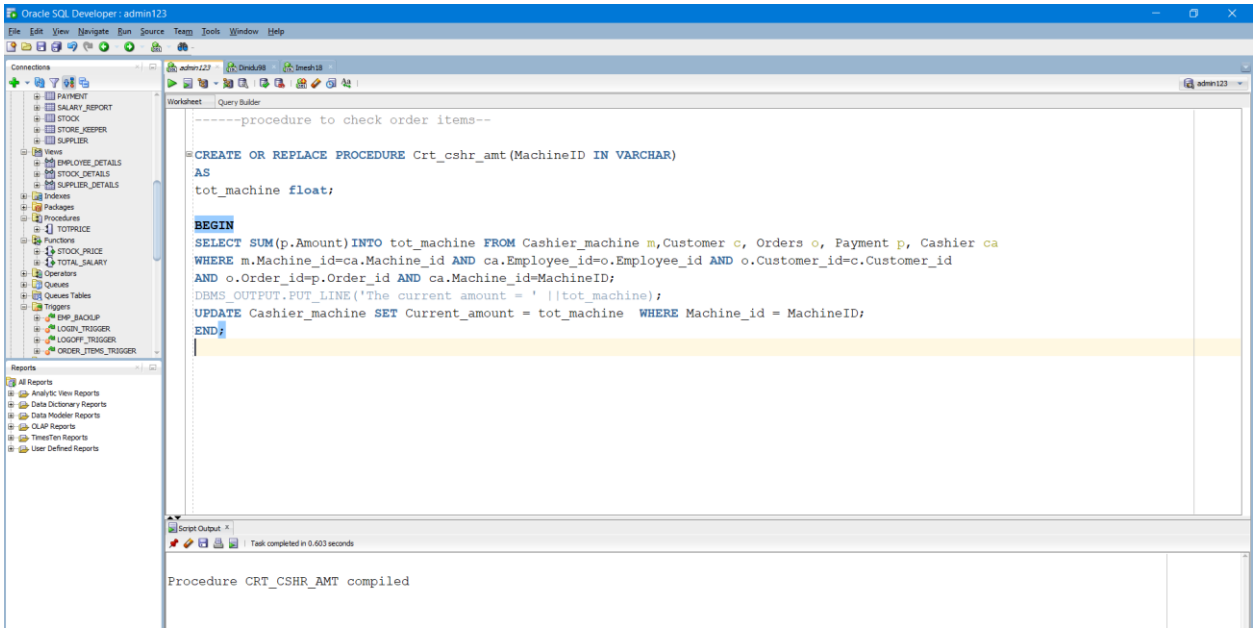




## 6.4.2 Create Procedure Crt_cshr_amt

Admin creates a procedure to procedure update the current amount of cashier machines.

```
--procedure to check order items--
CREATE OR REPLACE PROCEDURE Crt_cshr_amt(MachineID IN VARCHAR)
AS
tot_machine float;
BEGIN
SELECT SUM(p.Amount)INTO tot_machine FROM Cashier_machine m,Customer c,
Orders o, Payment p, Cashier ca
WHERE m.Machine_id=ca.Machine_id AND ca.Employee_id=o.Employee_id AND
o.Customer_id=c.Customer_id
AND o.Order_id=p.Order_id AND ca.Machine_id=MachineID;
DBMS_OUTPUT.PUT_LINE('The current amount = ' ||tot_machine);
UPDATE Cashier_machine SET Current_amount = tot_machine  WHERE Machine_id
= MachineID;
END;
```




Grant privileges to manager to execute Crt_cshr_amt

**Enter user-name: admin123**

**Enter password:**

**Last Successful login time: Sun May 03 2020 12:18:51 +05:30**

**Connected to:**

**Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production**

**Version 19.3.0.0.0**

**SQL> show user;**

**USER is "ADMIN123"**

**SQL> grant create session to manager;**

**Grant succeeded.**

**SQL> grant connect to manager;**

**Grant succeeded.**

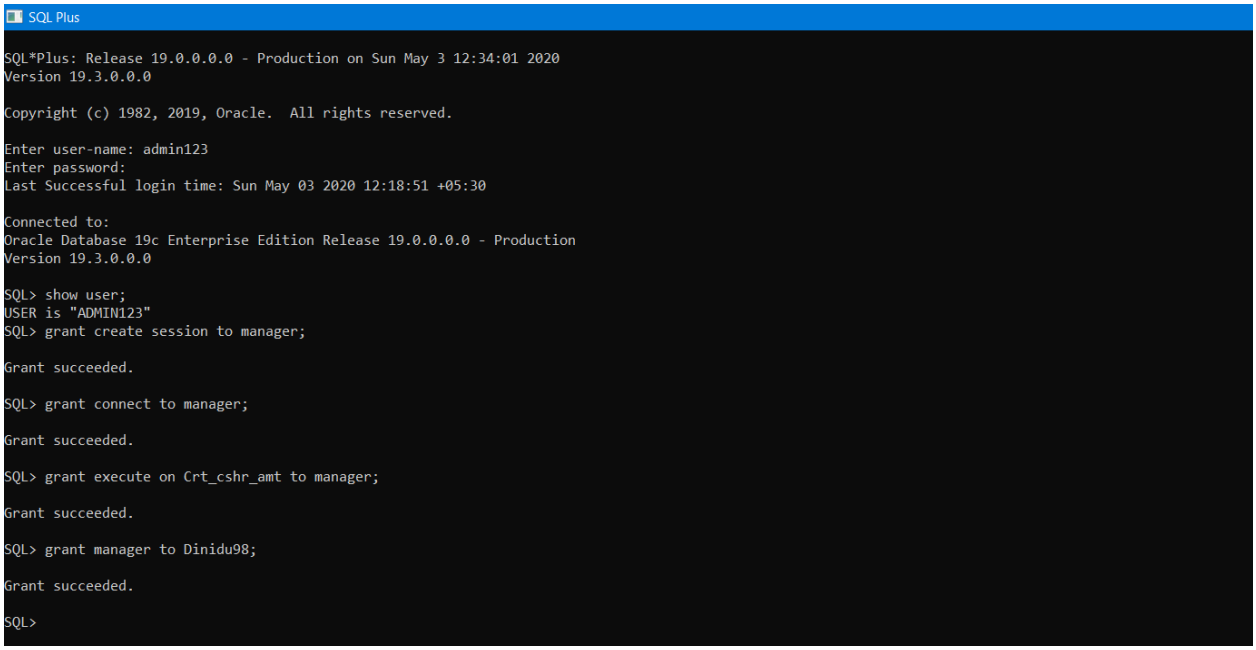
**SQL> grant execute on Crt_cshr_amt to manager;**

**Grant succeeded.**

**SQL> grant manager to Dinidu98;**

**Grant succeeded.**

**SQL>**




Grant the privileges to cashier to execute Crt_cshr_amt

**SQL> grant create session to cashier;**

**Grant succeeded.**

**SQL> grant connect to cashier;**

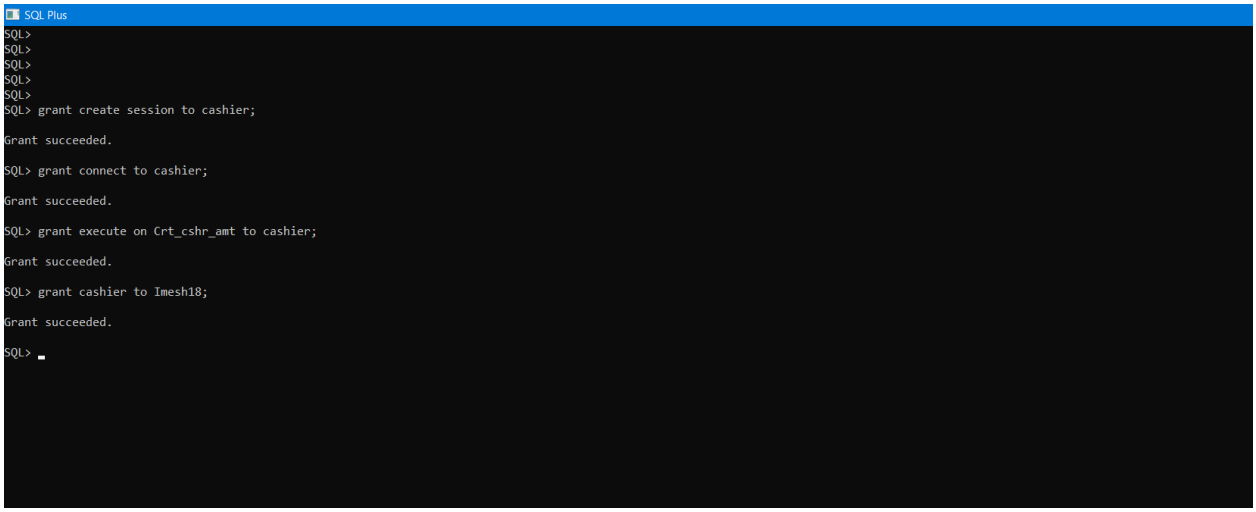
**Grant succeeded.**

**SQL> grant execute on Crt_cshr_amt to cashier;**

**Grant succeeded.**

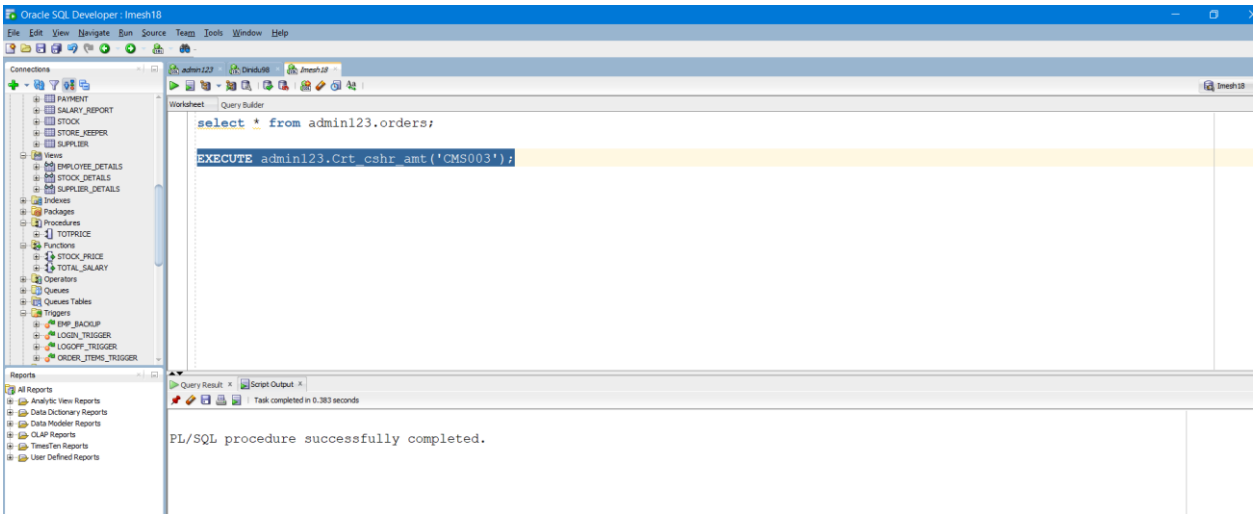
**SQL> grant cashier to Imesh18;**

**Grant succeeded.**



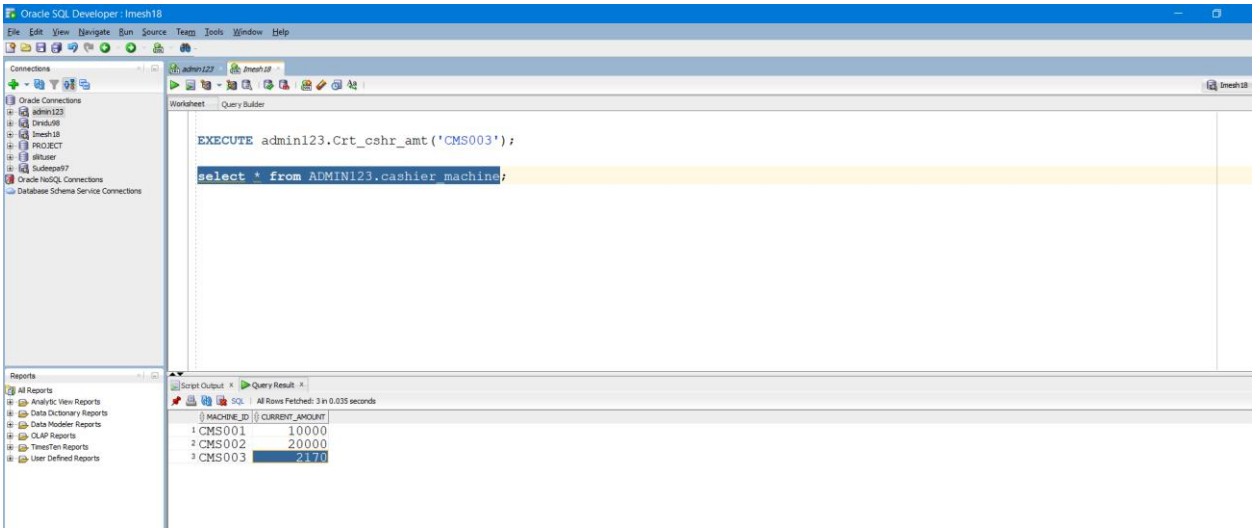

Now cashier can execute this function.

**EXECUTE admin123.Crt_cshr_amt('CMS003');**




When the user executes this procedure:

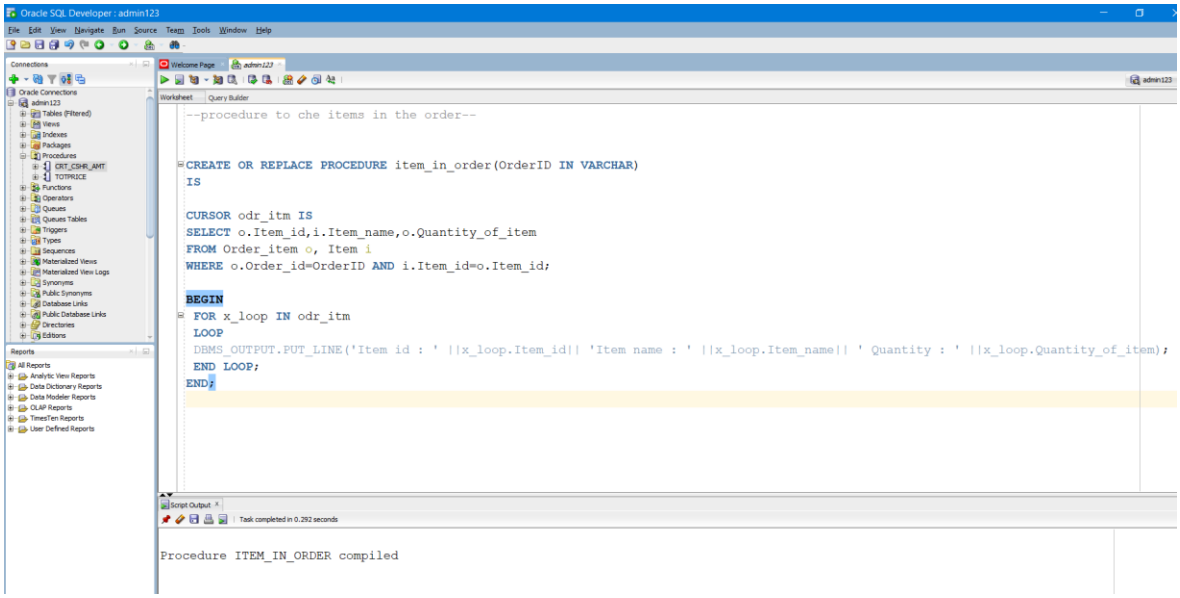
It updates the current amount of the given Machine_ids' table. Firstly, get the sum of payment amounts belongs to the given machine_id. Then it adds the current amount of cashier machine. After the display, that new current amount and update it on the table.




### 6.4.3 Create Procedure item_in_order

Admin creates a procedure called item_in_order and gives the execution permission to the manager and cashier.

```
CREATE OR REPLACE PROCEDURE item_in_order(OrderID IN VARCHAR)
IS
CURSOR odr_itm IS
SELECT o.Item_id,i.Item_name,o.Quantity_of_item
FROM Order_item o, Item i
WHERE o.Order_id=OrderID AND i.Item_id=o.Item_id;
BEGIN
 FOR x_loop IN odr_itm
 LOOP
 DBMS_OUTPUT.PUT_LINE('Item id : ' ||x_loop.Item_id|| 'Item name : '
||x_loop.Item_name|| ' Quantity : ' ||x_loop.Quantity_of_item);
 END LOOP;
END
```




Then we give the grant execute permission to the manager and cashier.

**SQL> conn admin123**

**Enter password:**

**Connected.**

**SQL> grant create session to manager;**

**Grant succeeded.**

**SQL> grant connect to manager;**

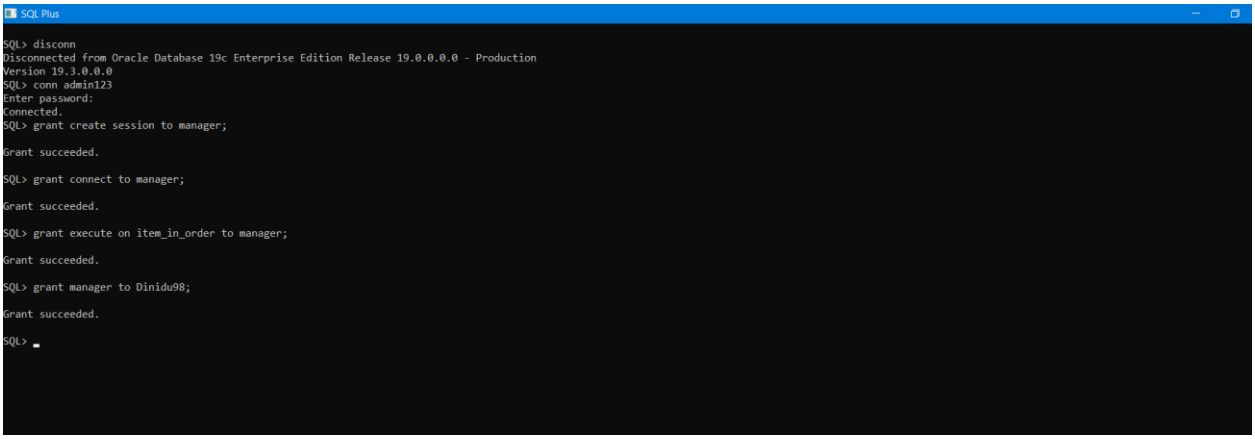
**Grant succeeded.**

**SQL> grant execute on item_in_order to manager;**

**Grant succeeded.**

**SQL> grant manager to Dinidu98;**

**Grant succeeded.**



**SQL> grant create session to cashier;**

**Grant succeeded.**

**SQL> grant connect to cashier;**

**Grant succeeded.**

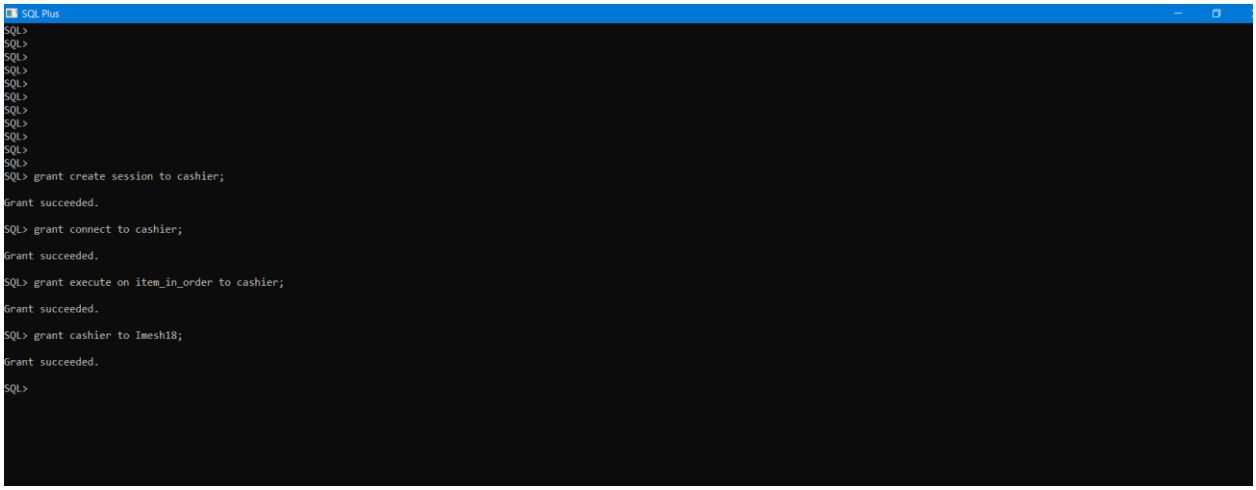
**SQL> grant execute on item_in_order to cashier;**

**Grant succeeded.**

**SQL> grant cashier to Imesh18;**

**Grant succeeded.**

**SQL>**

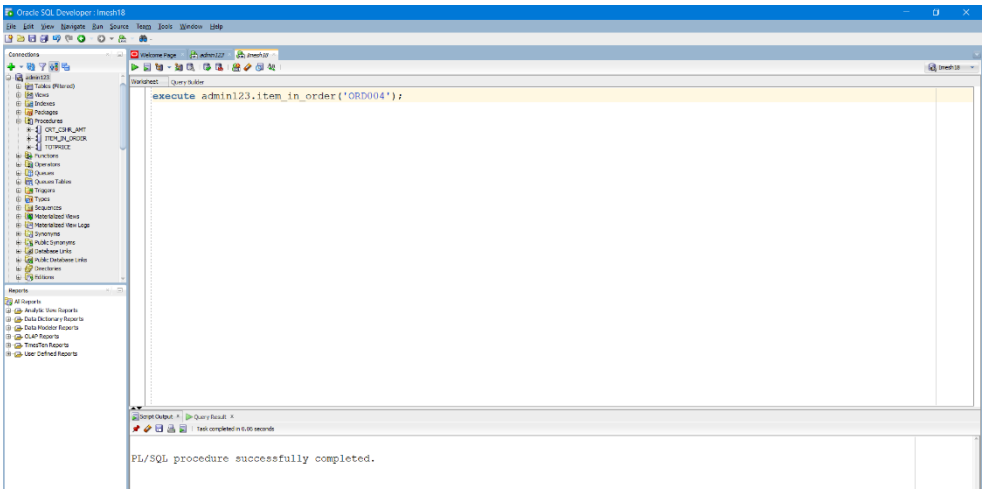



Then the cashier and manager can execute this procedure.

**execute admin123.item_in_order('ORD004');**

When executing this procedure:

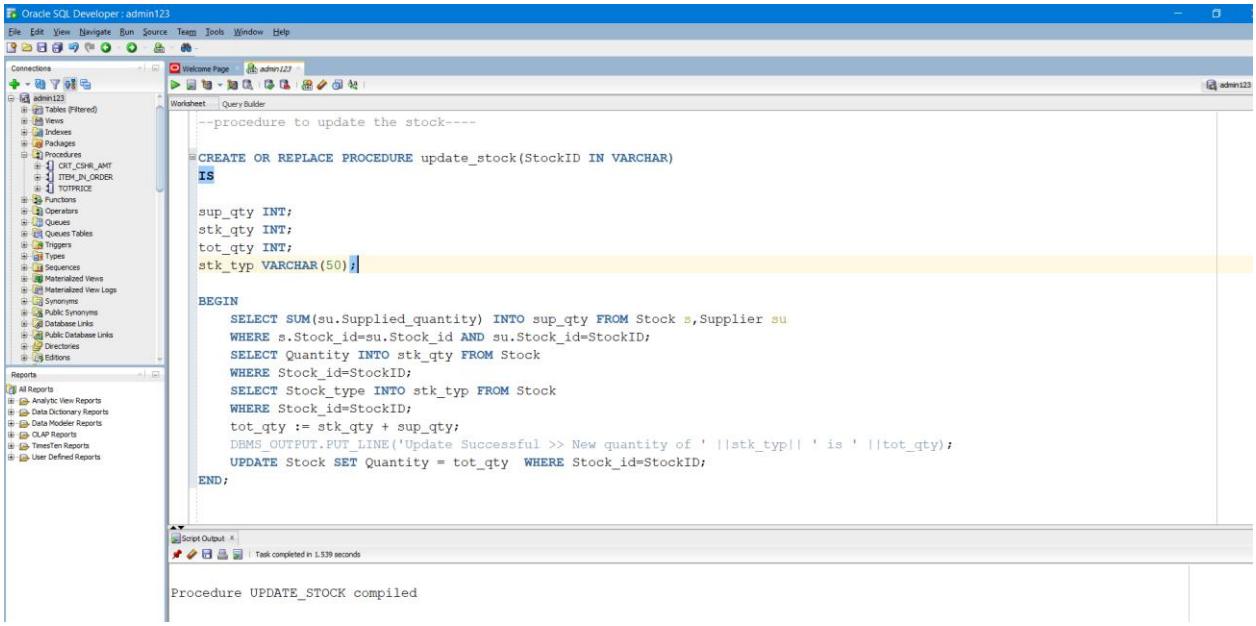
It displays some kind of detail in item and quantity for the given order_id.



### 6.4.4 Create Procedure update_stock

Admin creates a procedure to update stock and grant the execution permission to the manager and stock keeper

```
CREATE OR REPLACE PROCEDURE update_stock(StockID IN VARCHAR)
IS

sup_qty INT;
stk_qty INT;
tot_qty INT;
stk_typ VARCHAR(50);

BEGIN
   SELECT SUM(su.Supplied_quantity) INTO sup_qty FROM Stock s,Supplier su
   WHERE s.Stock_id=su.Stock_id AND su.Stock_id=StockID;
   SELECT Quantity INTO stk_qty FROM Stock
   WHERE Stock_id=StockID;
   SELECT Stock_type INTO stk_typ FROM Stock
   WHERE Stock_id=StockID;
   tot_qty := stk_qty + sup_qty;
   DBMS_OUTPUT.PUT_LINE('Update Successful >> New quantity of ' ||stk_typ|| ' is '
||tot_qty);
   UPDATE Stock SET Quantity = tot_qty  WHERE Stock_id=StockID;
END;
```




Then give the grant permission to execute this procedure to the manager and storekeeper.

**SQL> conn admin123**

**Enter password:**

**Connected.**

**SQL> grant create session to manager;**

**Grant succeeded.**

**SQL> grant connect to manager;**

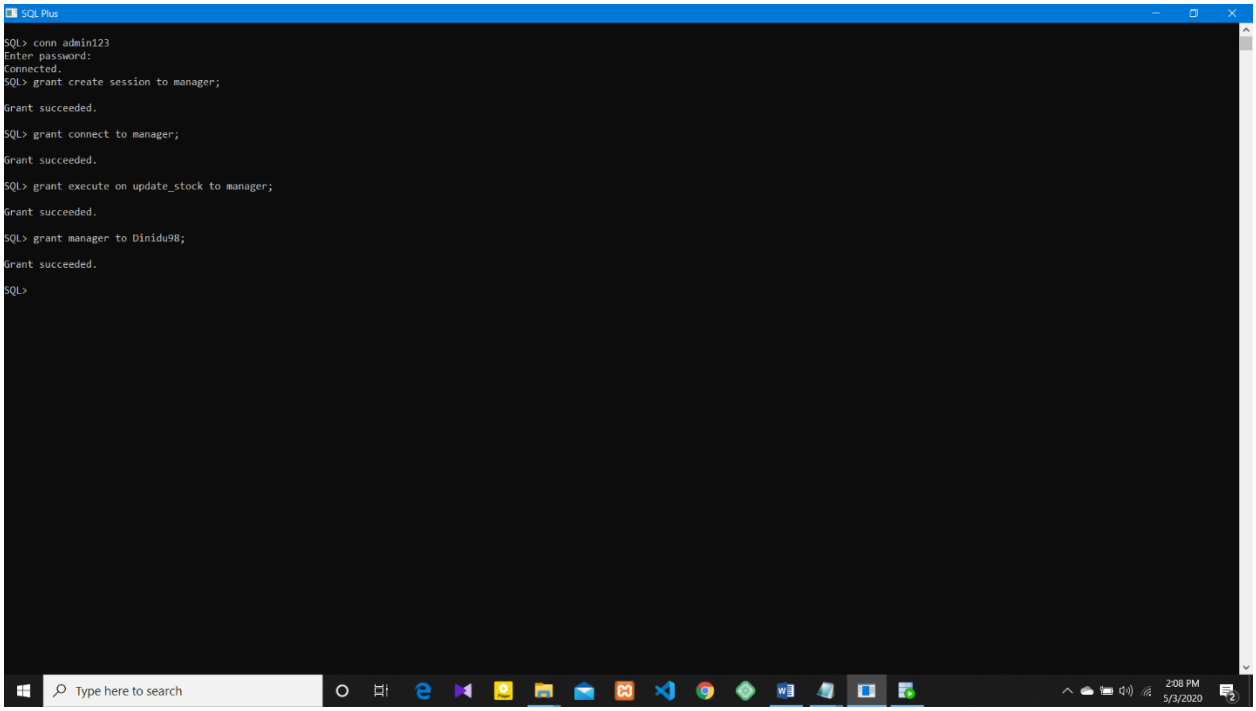
**Grant succeeded.**

**SQL> grant execute on update_stock to manager;**

**Grant succeeded.**

**SQL> grant manager to Dinidu98;**

**Grant succeeded.**




Give permission to the storekeeper.

**SQL>**

**SQL> grant create session to store_keeper;**

**Grant succeeded.**

**SQL> grant connect to store_keeper;**

**Grant succeeded.**

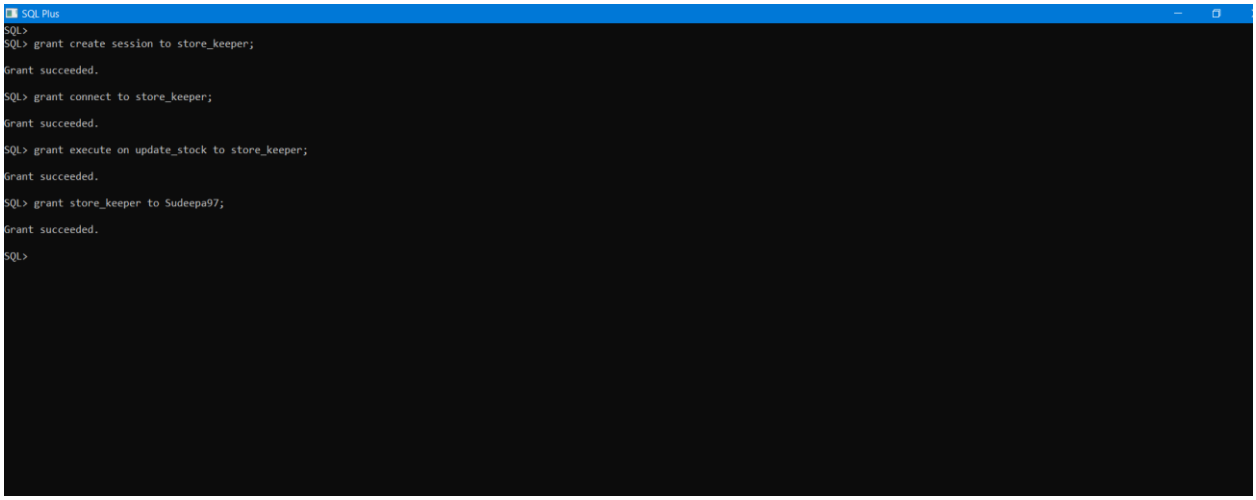
**SQL> grant execute on update_stock to store_keeper;**

**Grant succeeded.**

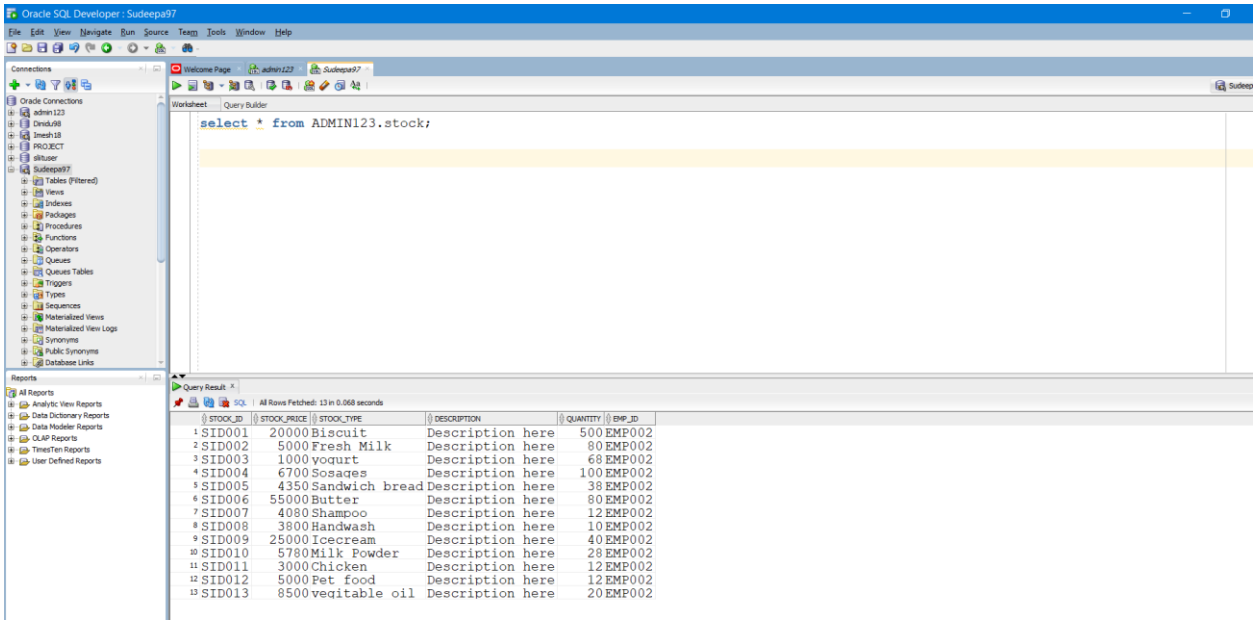
**SQL> grant store_keeper to Sudeepa97;**

**Grant succeeded.**

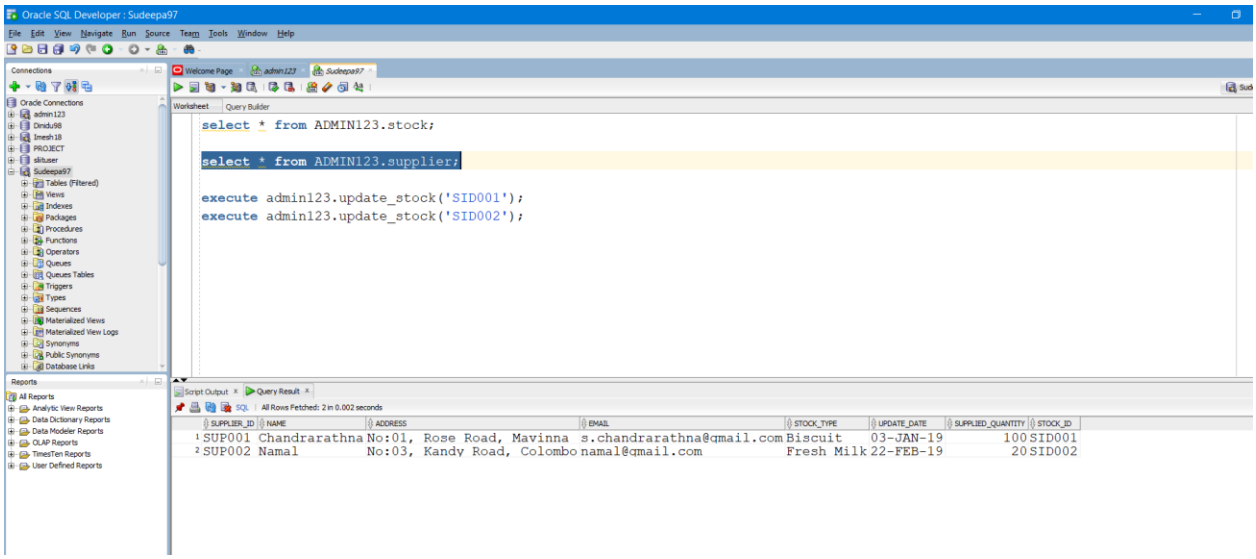
**SQL>**



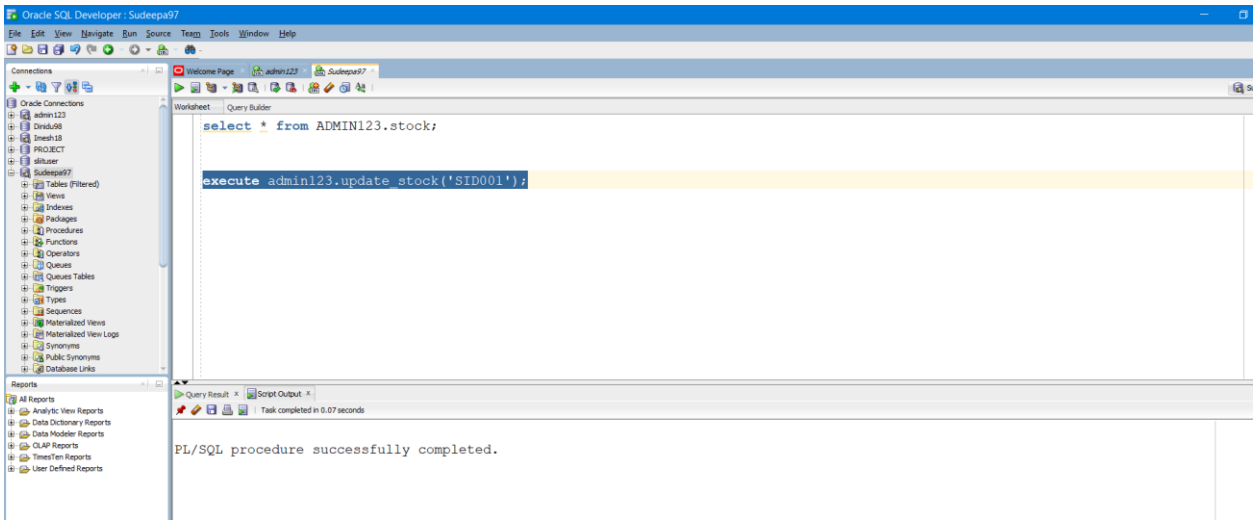

Before executing the procedure:



Supplier table details.




After executing the update_stock it automatically updates the stock with supplied quantity.

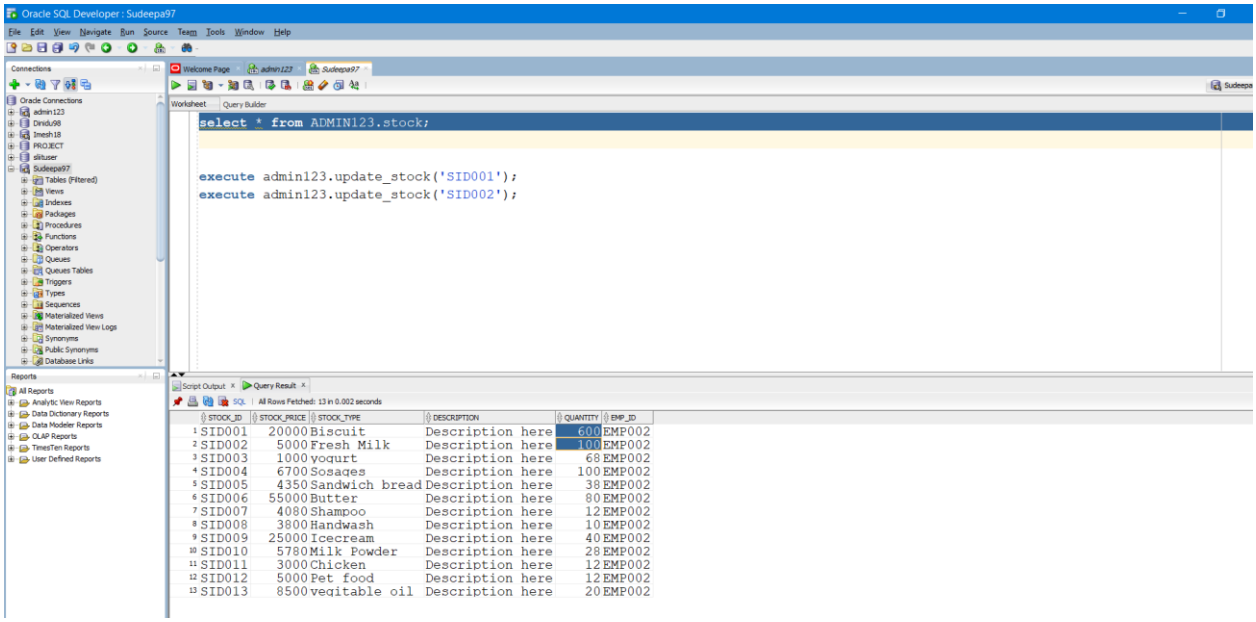



We can explain as it is following.

This procedure uses to update the stock quantity of given stock_id. Firstly, the procedure gets the sum of new quantities which belong to the given stock. Then add that sum to the current quantity of the given stock id, after displaying this new quantity and update it into the stock table.

**execute admin123.update_stock('SID001');**

**execute admin123.update_stock('SID002');**




This also can do by the manager of the supermarket as well.

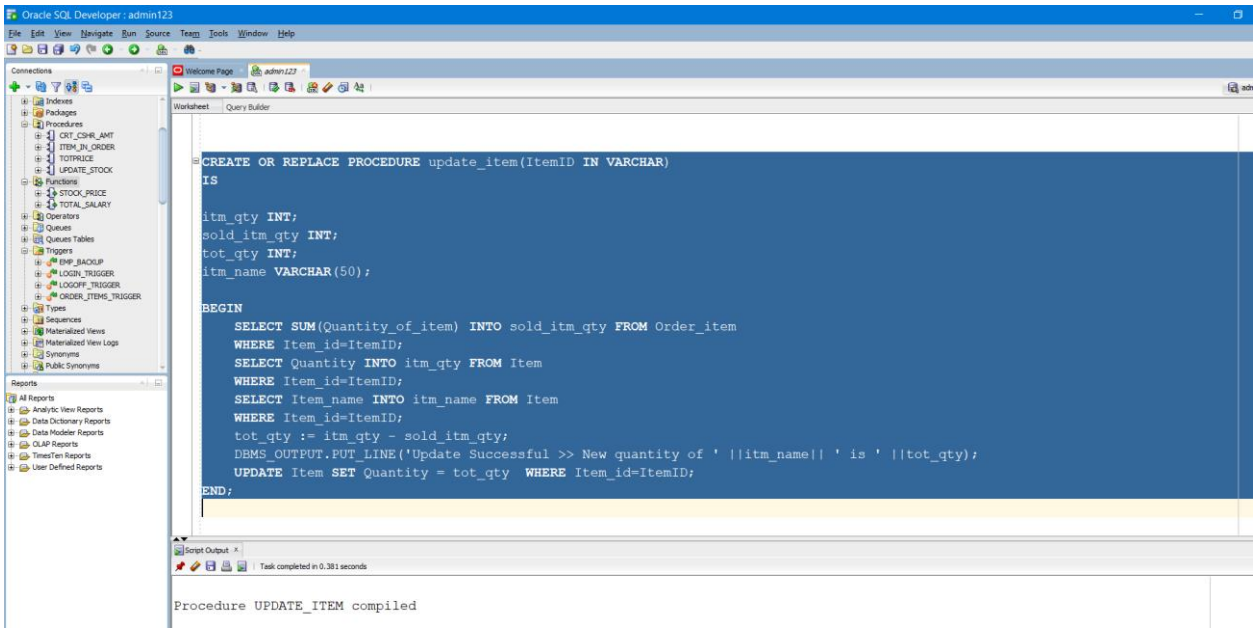
### 6.4.5 Create Procedure update_item

Create a procedure called update_item to update item quantity.

Admin create this procedure and grant the execute permission to manager and storekeeper;

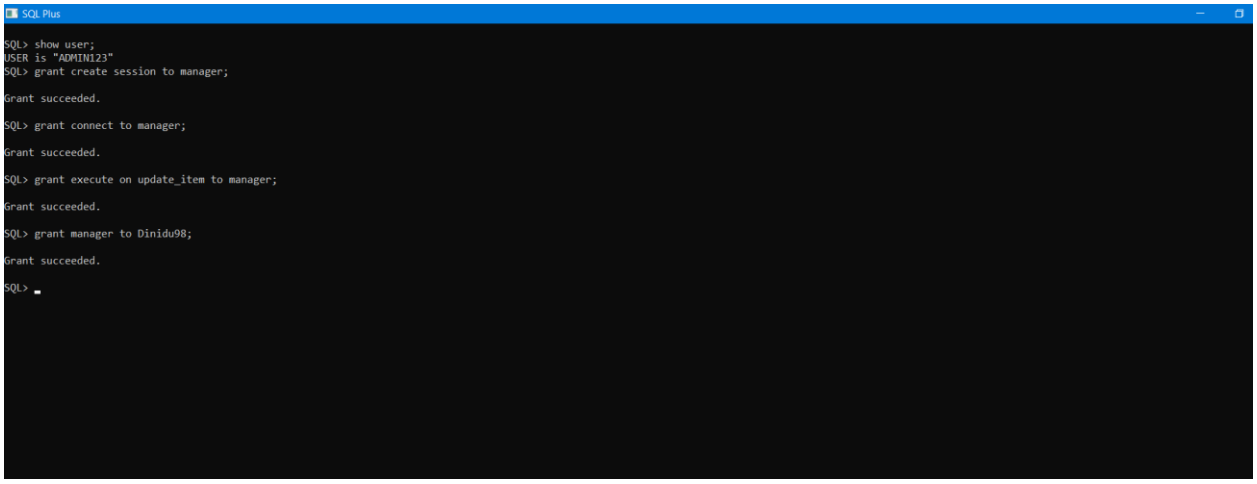
```
CREATE OR REPLACE PROCEDURE update_item(ItemID IN VARCHAR)
IS

itm_qty INT;
sold_itm_qty INT;
tot_qty INT;
itm_name VARCHAR(50);

BEGIN
   SELECT SUM(Quantity_of_item) INTO sold_itm_qty FROM Order_item
   WHERE Item_id=ItemID;
   SELECT Quantity INTO itm_qty FROM Item
   WHERE Item_id=ItemID;
   SELECT Item_name INTO itm_name FROM Item
```

```
    WHERE Item_id=ItemID;
    tot_qty := itm_qty - sold_itm_qty;
    DBMS_OUTPUT.PUT_LINE('Update Successful >> New quantity of ' ||itm_name|| '
        is ' ||tot_qty);
    UPDATE Item SET Quantity = tot_qty  WHERE Item_id=ItemID;
END;
```




Grant the execute permission to store_keeper and manager.

```
SQL> show user;
USER is "ADMIN123"
SQL> grant create session to manager;
Grant succeeded.
SQL> grant connect to manager;
Grant succeeded.
SQL> grant execute on update_item to manager;
Grant succeeded.
SQL> grant manager to Dinidu98;
Grant succeeded.
```




**SQL> grant create session to store_keeper;**

**Grant succeeded.**

**SQL> grant connect to store_keeper;**

**Grant succeeded.**

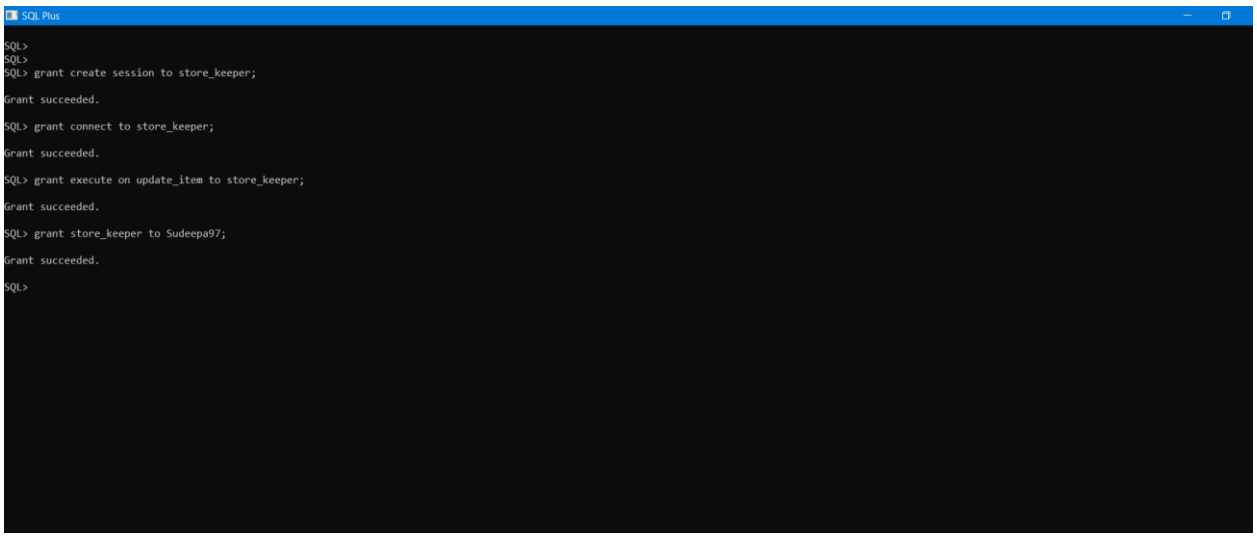
**SQL> grant execute on update_item to store_keeper;**

**Grant succeeded.**

**SQL> grant store_keeper to Sudeepa97;**

**Grant succeeded.**

**SQL>**

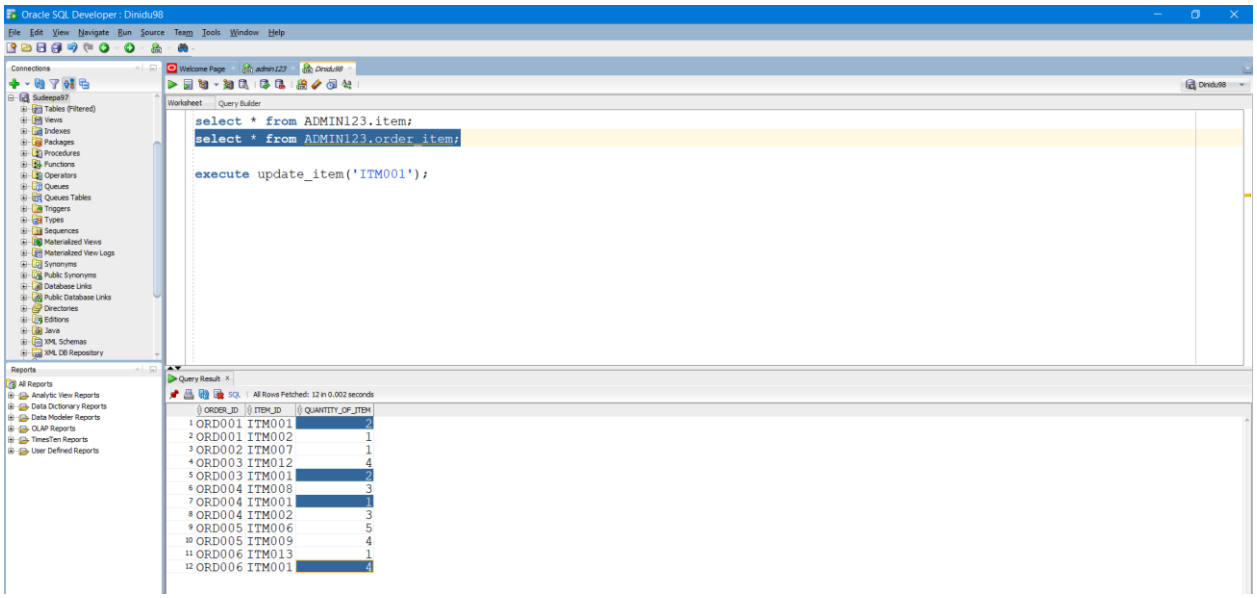


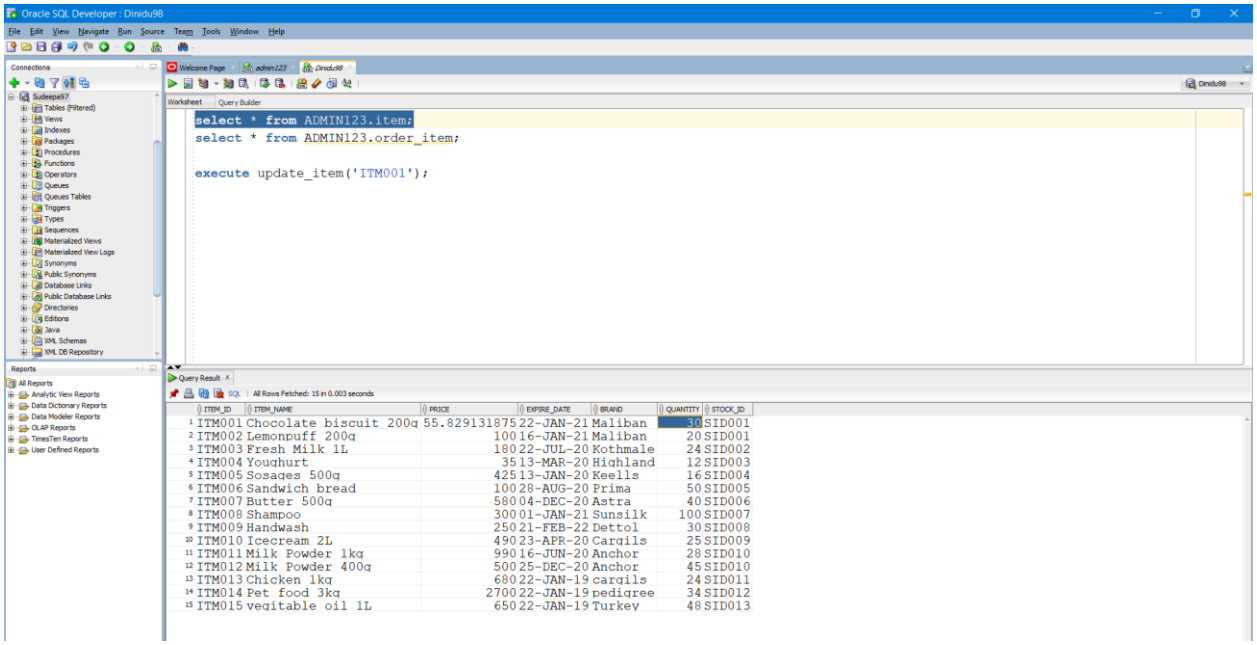

Now manager and store_keeper can execute this procedure.

So, let us see how the manager executes this. To execute this, we should give the select, update permission to the manager.

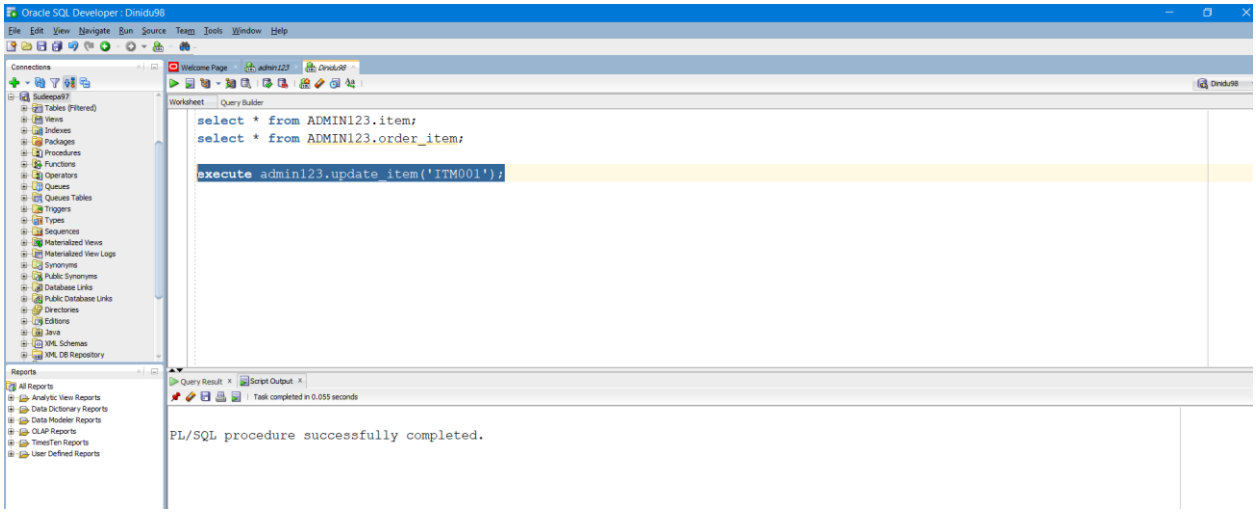
```
SQL Plus

SQL> grant create session to manager;

Grant succeeded.

SQL> grant connect to manager;

Grant succeeded.

SQL> grant select, update on Order_item to manager;

Grant succeeded.

SQL> grant manager to Dinidu98;

Grant succeeded.

SQL>
```

Before we execute the Item table and order_item table shown as follows.



The available quantity of ITM001 is 30. 9 Items are already ordered by customers.

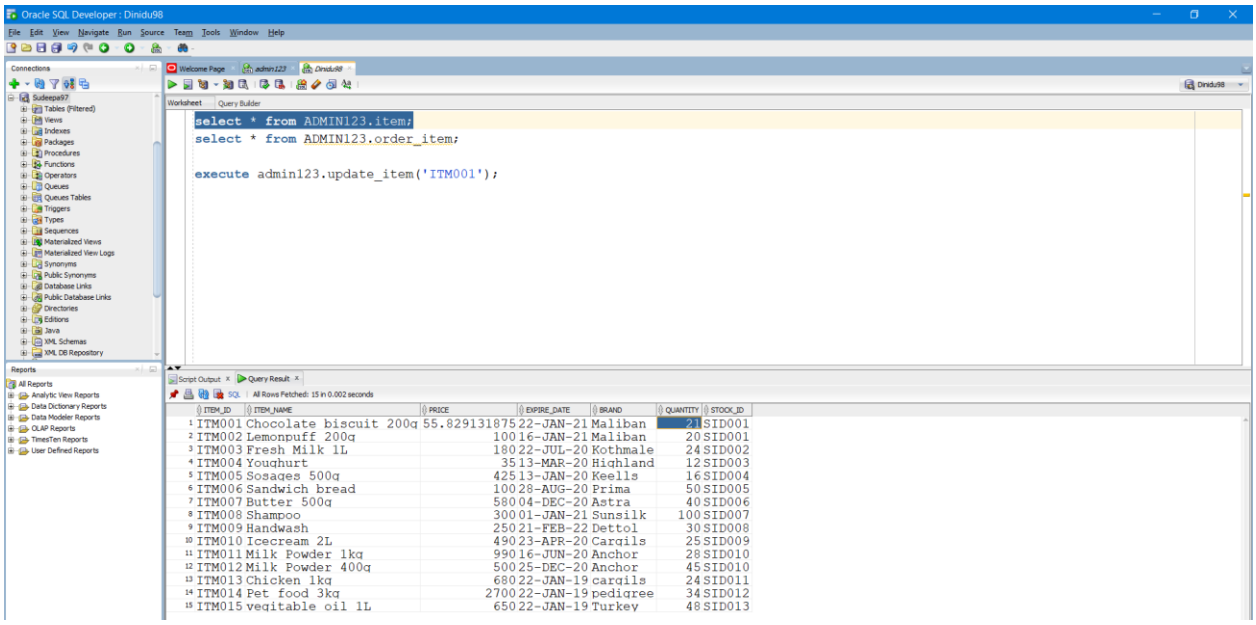



After executing the update_item procedure, it gives the quantity of the rest of the items. Basically,

New quantity of Items = Available Items – No of order Items

X = 30 – 9

X = 21



## 6.5 Additional Security Implementations

### 6.5.1 Enabling AUDIT_TRAIL and setting it up with distinct privileges

**Enter user-name: admin123**

**Enter password:**

**Last Successful login time: Mon Apr 20 2020 21:04:33 +05:30**

**Connected to:**

**Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 -Production**

**Version 19.3.0.0.0**

**SQL> ALTER SYSTEM SET AUDIT_TRAIL=XML, EXTENDED SCOPE=SPFILE;**

**System altered.**

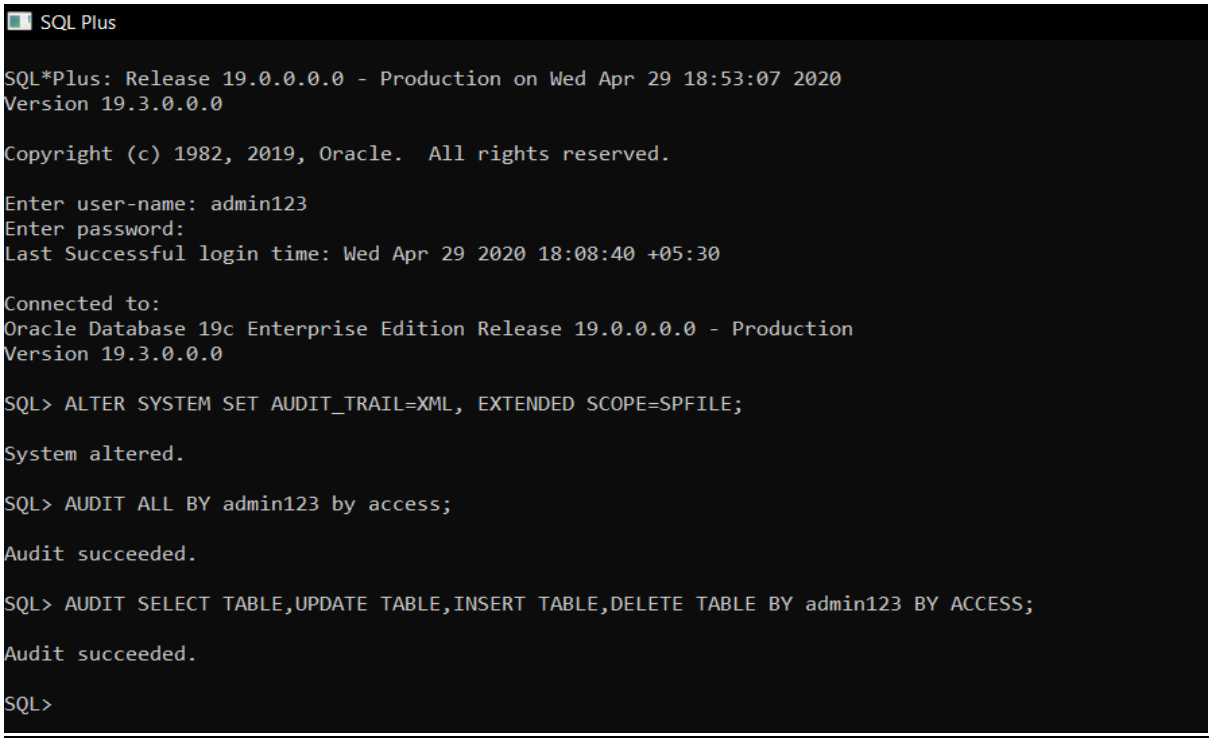
**SQL> AUDIT ALL BY admin123 by access;**

**Audit succeeded.**

**SQL> AUDIT SELECT TABLE, UPDATE TABLE, INSERT TABLE, DELETE TABLE BY admin123 BY ACCESS;**

**Audit succeeded.**

**SQL>**

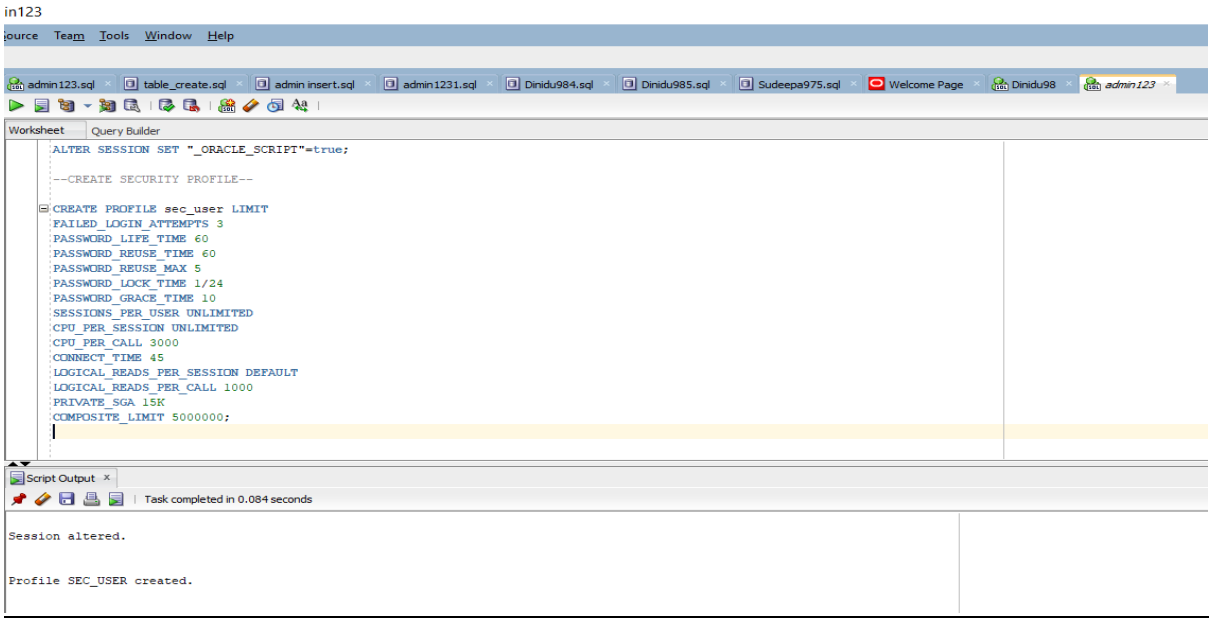


### 6.5.2 Create a security profile and lock the profile after failing to login with password

First, log in to the system by giving the password and create a security profile for a particular user can provide the wrong password three attempts.

Create a security profile

```
ALTER SESSION SET "_ORACLE_SCRIPT"=TRUE;

CREATE PROFILE sec_user LIMIT
FAILED_LOGIN_ATTEMPTS 3
PASSWORD_LIFE_TIME 60
PASSWORD_REUSE_TIME 60
PASSWORD_REUSE_MAX 5
PASSWORD_LOCK_TIME 1/24
PASSWORD_GRACE_TIME 10
SESSIONS_PER_USER UNLIMITED
CPU_PER_SESSION UNLIMITED
CPU_PER_CALL 3000
CONNECT_TIME 45
LOGICAL_READS_PER_SESSION DEFAULT
LOGICAL_READS_PER_CALL 1000
PRIVATE_SGA 15K
COMPOSITE_LIMIT 5000000;
```




After creating the security profile (sec_user), change the profile of the user.

Changing the profile of a User

**SQL>**

**SQL> connect admin123**

**Enter password:**

**Connected.**

**SQL>**

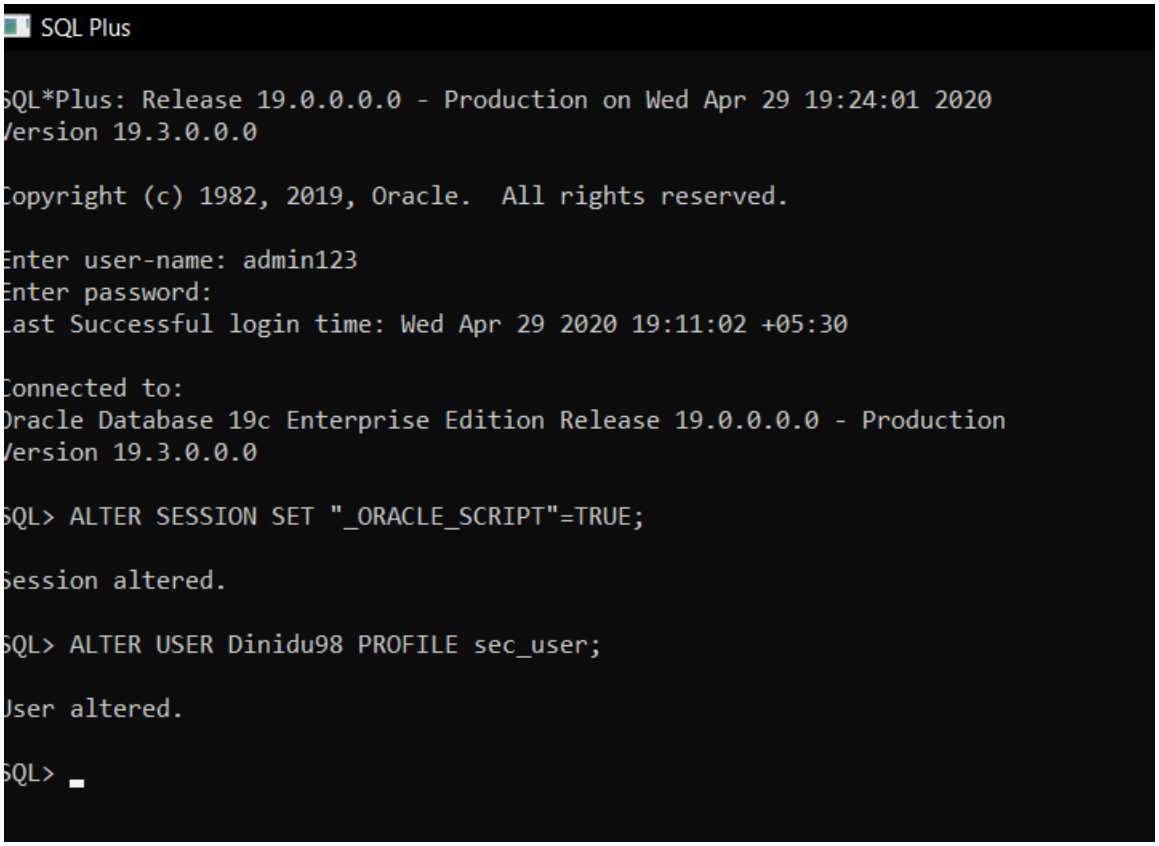
**SQL> alter session set "_ORACLE_SCRIPT"=true;**

**Session altered.**

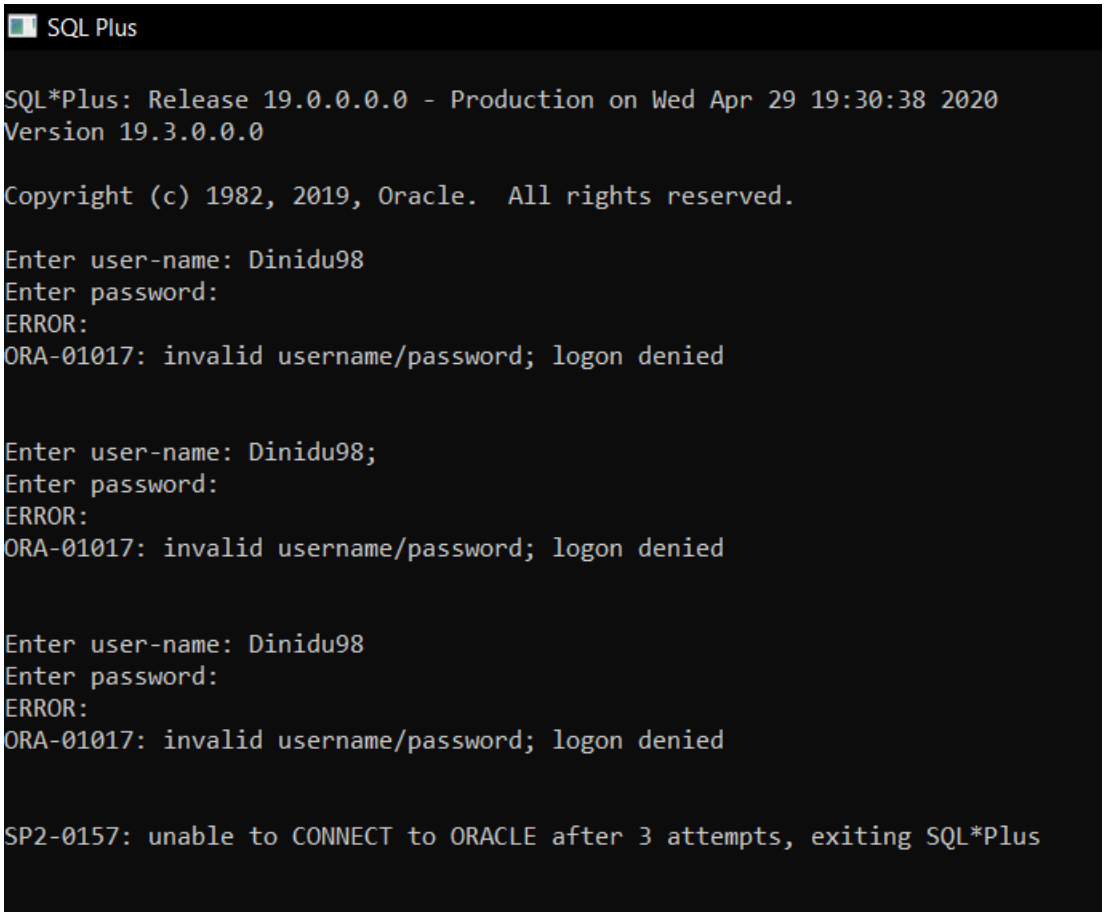
**SQL> alter user Dinidu98 profile sec_user;**

**User altered.**

**SQL>**




After checking the profile activities by trying to login “dinidu98” with an invalid password three times. In the fourth attempt, the profile is going to be locked.





We can unlock the user by login as a system account and run this command.

**SQL> alter user Dinidu98 identified by Dinidu#$1998 account unlock;**

**User altered.**

**SQL> CONNECT**

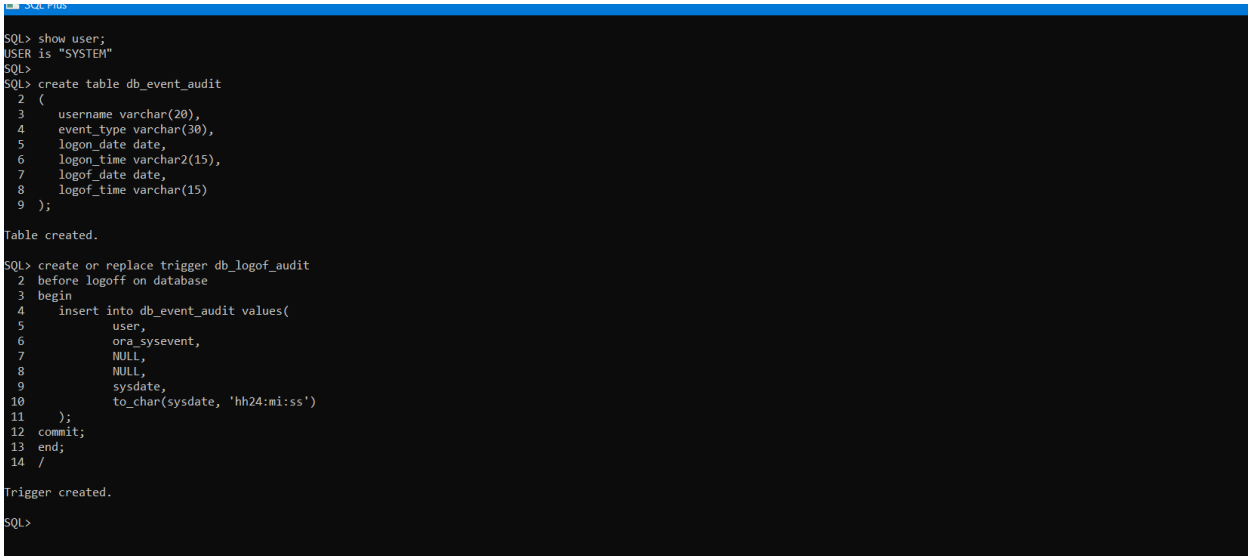
**Enter user-name: Dinidu98**

**Enter password:**

**Connected.**

### 6.5.3 Create a database audit event to detect all the user's logon and logoff details.

Implementing this can identify the user who logs into the system according to the time, date, and event. First, we create a new database table to store the audited data called db_event_audit. This creates the system user of the database, and it detects all the logon and logoff behaviors of all the users in the database.



```
SQL> show user;
USER is "SYSTEM"
SQL>
SQL> create table db_event_audit
  2  (
  3     username varchar(20),
  4     event_type varchar(30),
  5     logon_date date,
  6     logon_time varchar2(15),
  7     logof_date date,
  8     logof_time varchar(15)
  9  );

Table created.

SQL> create or replace trigger db_logof_audit
  2  before logoff on database
  3  begin
  4     insert into db_event_audit values(
  5             user,
  6             ora_sysevent,
  7             NULL,
  8             NULL,
  9             sysdate,
 10             to_char(sysdate, 'hh24:mi:ss')
 11     );
 12  commit;
 13  end;
 14  /

Trigger created.

SQL>
```

Creating the db_logof_audit trigger.

```
SQL> create or replace trigger db_logof_audit
  2  before logoff on database
  3  begin
  4     insert into db_event_audit values(
  5             user,
  6             ora_sysevent,
  7             NULL,
  8             NULL,
  9             sysdate,
 10              to_char(sysdate, 'hh24:mi:ss'),
 11        );
 12     commit;
 13     end;
```

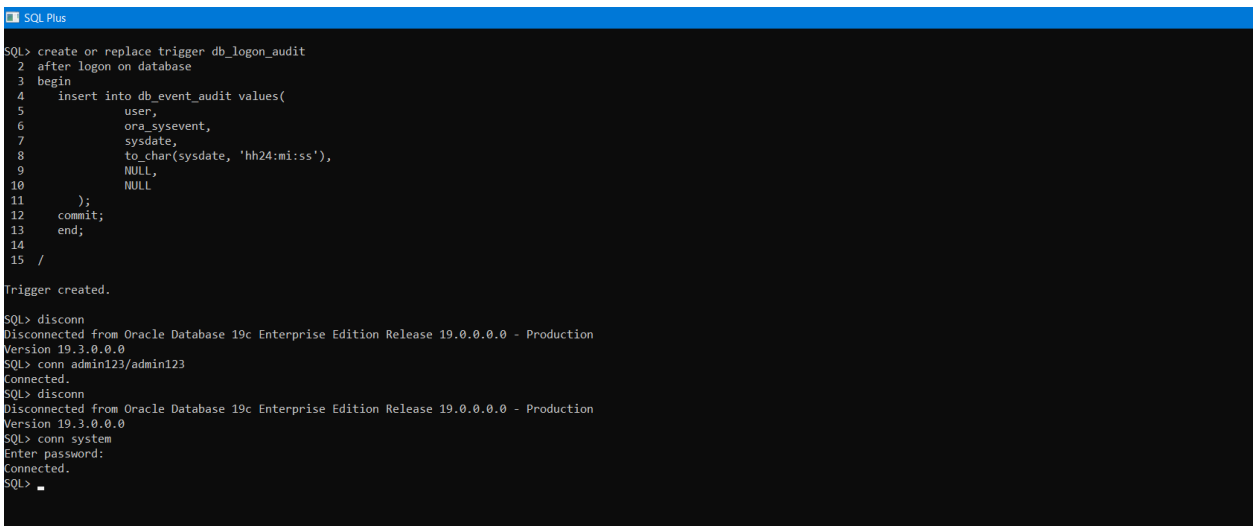
```
 14
 15  /
```

Trigger created.

After we create a trigger db_logon_audit.

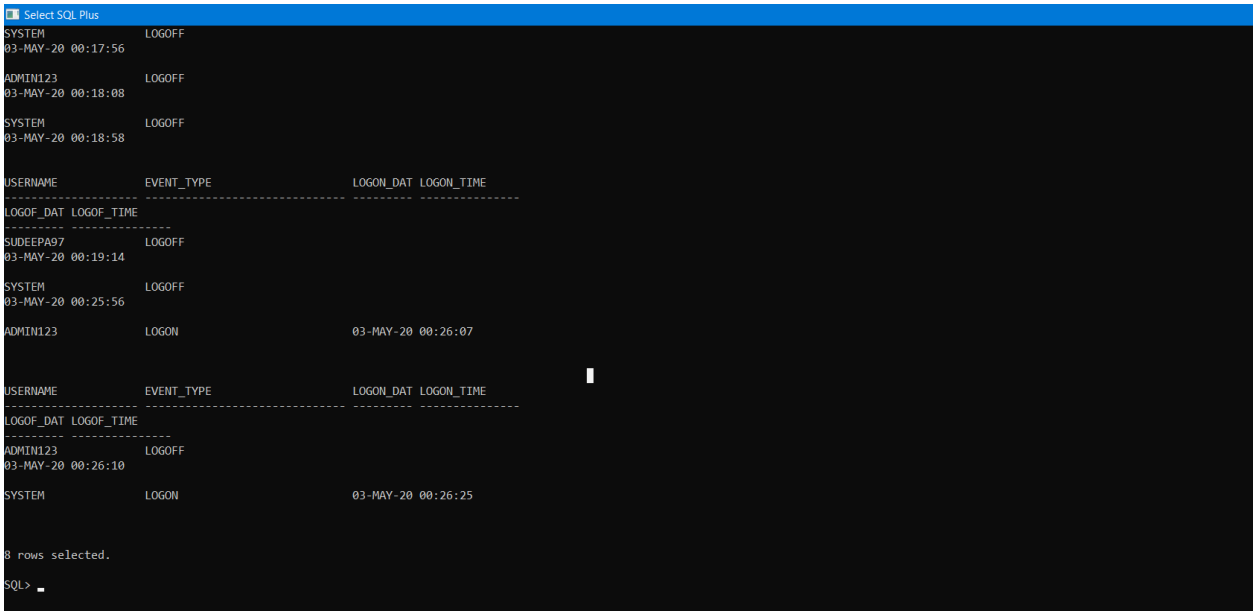
```
SQL> create or replace trigger db_logon_audit
  2  after logon on database
  3  begin
  4     insert into db_event_audit values(
  5               user,
  6               ora_sysevent,
  7               sysdate,
  8               to_char(sysdate, 'hh24:mi:ss'),
  9               NULL,
 10                NULL
 11        );
 12     commit;
 13     end;
 14
 15  /
```

Trigger created.



After we disconnect, the user can connect with a different user account.

The results are shown below.



# 7. Identified transaction in the supermarket management system

**Transaction 01**

Set a transaction when updating the salary report table.

Case Scenario:

Assume manager is going to update the salary_report table. The manager updates one record in the table, for example, update the salary of EMP001 for a particular month. Also, he updates other employees' salary amount to the table, for instance, EMP002, EMP003, according to the relevant month. After updating some records, if the recorded salary amount for EMP002 has recorded as wrong data.

| | |
|---|---|
| T0 | COMMIT; |
| T1 | SET TRANSACTION NAME 'update_sal'; |
| T2 | UPDATE Salary_report<br>SET Total_salary = 7000<br>WHERE Emp_id = 'EMP001' AND<br>Month = 'JAN'; |
| T3 | SAVEPOINT after_emp001_sal_jan; |
| T4 | UPDATE Salary_report<br>SET Total_salary = 14000<br>WHERE Emp_id = 'EMP002' AND<br>Month = 'FEB'; |
| T5 | SAVEPOINT after_emp002_sal_feb; |
| T6 | ROLLBACK TO SAVEPOINT<br>after_emp001_sal_jan; |

| | |
|---|---|
| T7 | UPDATE Salary_report<br>SET Total_salary = 15500<br>WHERE Emp_id = 'EMP002' AND<br>Month = 'FEB'; |
| T8 | ROLLBACK; |
| T9 | SET TRANSACTION NAME 'update_sal_2'; |
| T10 | UPDATE Salary_report<br>SET Total_salary = 7500<br>WHERE Emp_id = 'EMP001' AND<br>Month = 'JAN'; |
| T11 | UPDATE Salary_report<br>SET Total_salary = 14500<br>WHERE Emp_id = 'EMP002' AND<br>Month = 'FEB'; |
| T12 | COMMIT; |

Explanation of transaction:

The manager can able to rollback transaction status to the previous save point (after_emp001_sal_jan) before updating the EMP001 salary amount. Then the manager can update the new salary amount and commit the transaction.

**Transaction 02**

Case scenario:

Suppose a customer has bought items 5000.00 by debit card or credit card. the transaction involves,

1. Read the current balance of the customer card
2. Reduce 5000.00 from the customer's card
3. Read the total of the transaction
4. Add 5000.00 to the current sum
5. Update and read the balance of the supermarket database with the new balance

Explanation of transaction:
futher, this is what happens in the customer's accounts,
older_blance=customer.blance
new_blance = older_blance - 5000.00
customer.blance = new_blance

And this is what happens in the customer's account,
older_blance = supermarket.blance
new_blance = older_blance _ 5000.00
supermarket.blance = new_blance

## 8.Explain about recovery mechanisms of the database

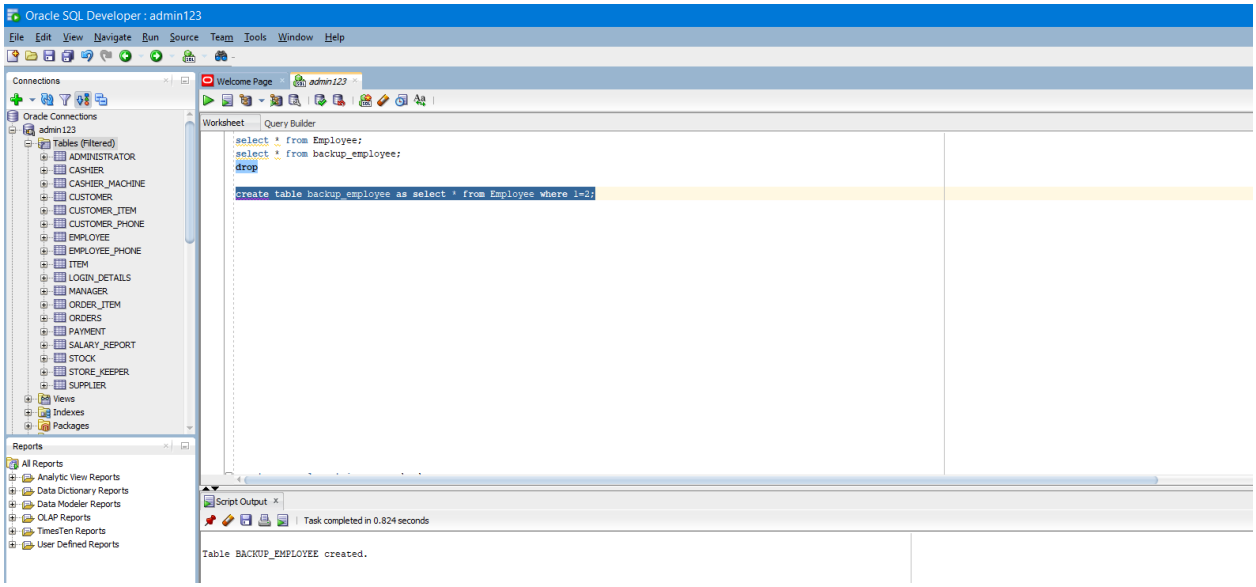
This is the set of methods, mechanisms, and concepts involved in protecting the database against data crashed and data loss. This can be happening by two main reasons because of media failure or human errors.

Backup and recovery is a significant part when implementing a proper database. It will help to recover all the table data if the database is compromised by an attacker or malware or crashed the database. All the tables in the supermarket system should backup as a separate backup table and should store it separately.

Create a backup table for each database table, and we can back up all the data which have inserted, updated, and also deleted in the main table. We can do this by creating a trigger.

To illustrate this process, we create a new table called backup_employee, which stores all the data in the employee table.

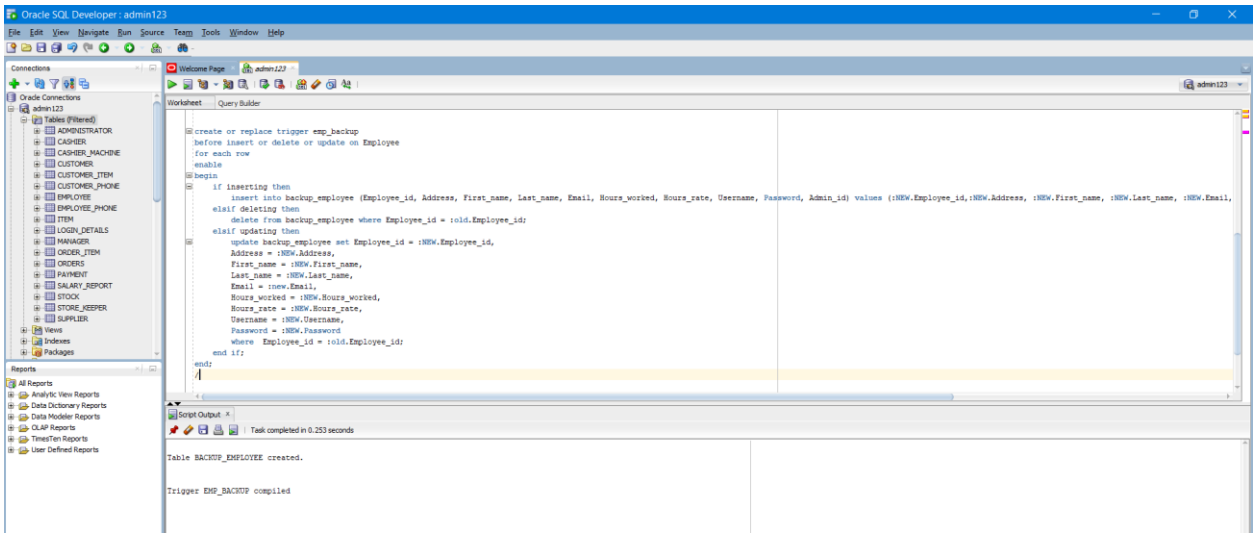
**create table backup_employee as select * from Employee where 1=2;**



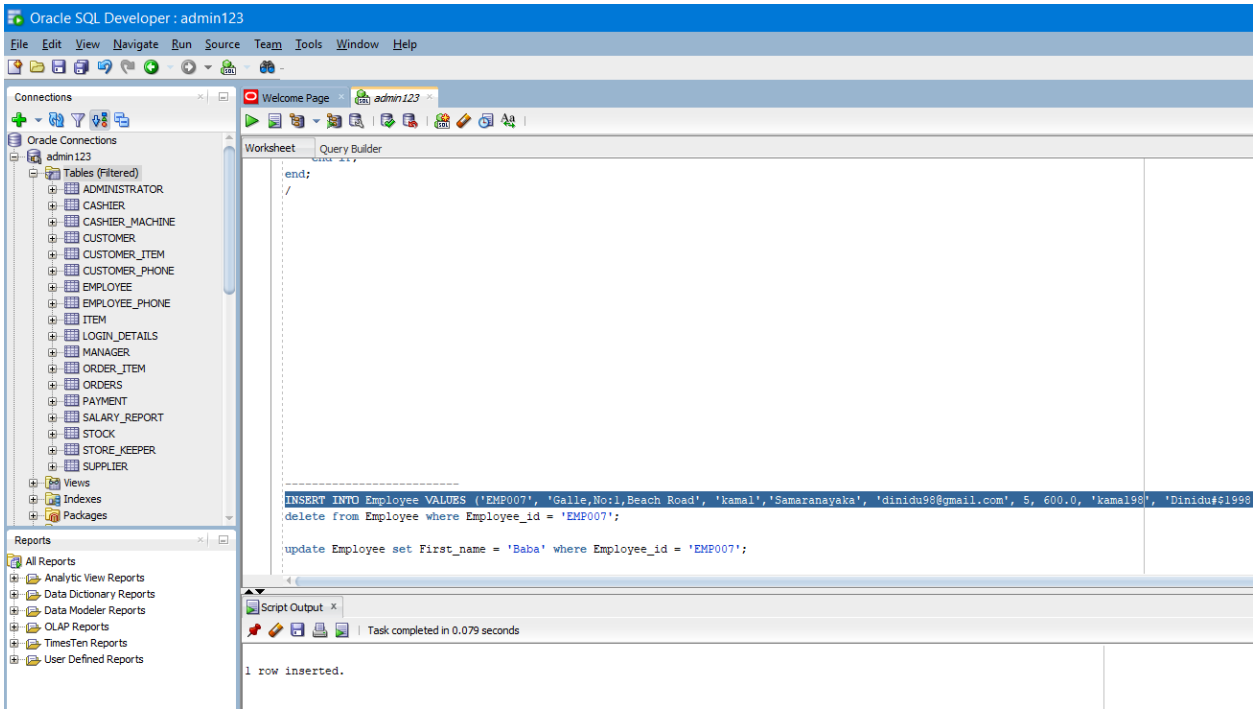

After creating the table, we create a trigger to insert, update, and delete the backup employee table.

```
create or replace trigger emp_backup
before insert or delete or update on Employee
for each row
enable
begin
   if inserting then
        insert into backup_employee (Employee_id, Address, First_name, Last_name,
        Email, Hours_worked, Hours_rate, Username, Password, Admin_id)
        values (:NEW.Employee_id,:NEW.Address, :NEW.First_name,
        :NEW.Last_name, :NEW.Email, :NEW.Hours_worked, :NEW.Hours_rate,
        :NEW.Username, :NEW.Password, :NEW.Admin_id);
   elsif deleting then
      delete from backup_employee where Employee_id = :old.Employee_id;
   elsif updating then
      update backup_employee set Employee_id = :NEW.Employee_id,
      Address = :NEW.Address,
      First_name = :NEW.First_name,
      Last_name = :NEW.Last_name,
      Email = :new.Email,
```

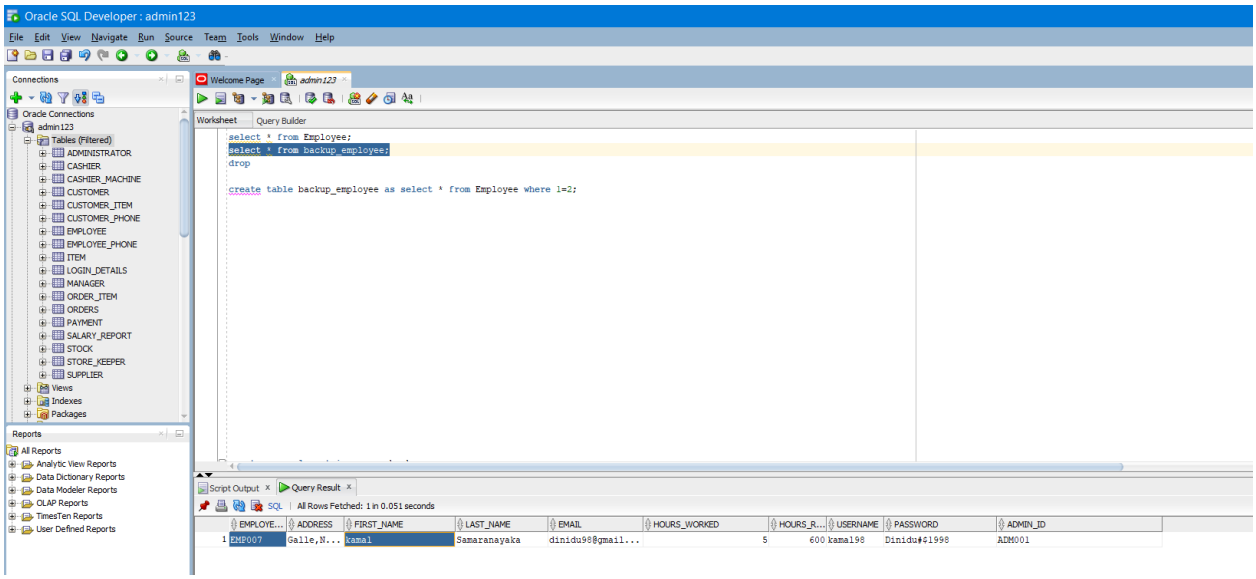
```
        Hours_worked = :NEW.Hours_worked,
        Hours_rate = :NEW.Hours_rate,
        Username = :NEW.Username,
        Password = :NEW.Password
        where  Employee_id = :old.Employee_id;
    end if;
end;
```

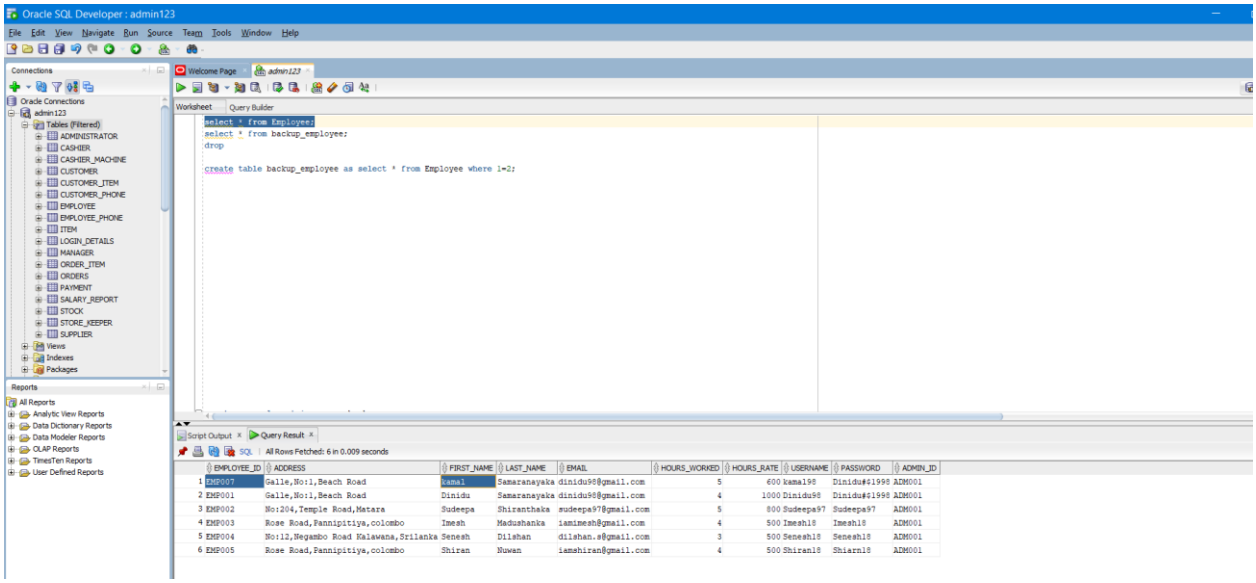



To check this trigger works and store it to the backup table, we insert a record to the table.

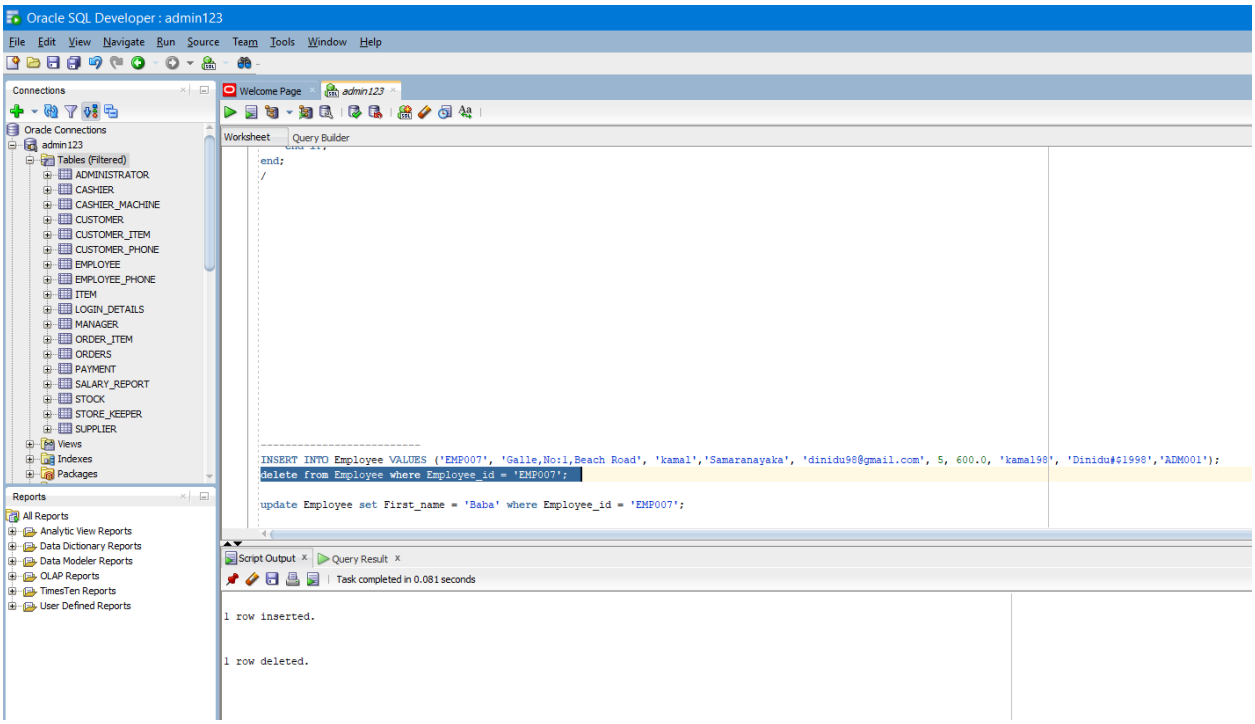


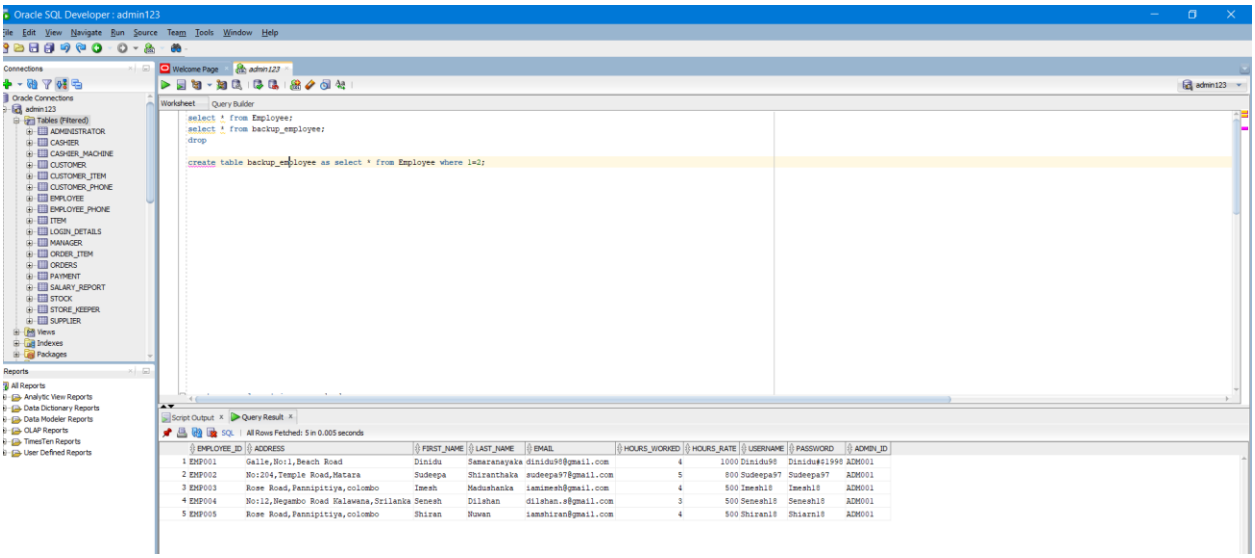
The employee table has inserted with a new record, and the backup table also has inserted.

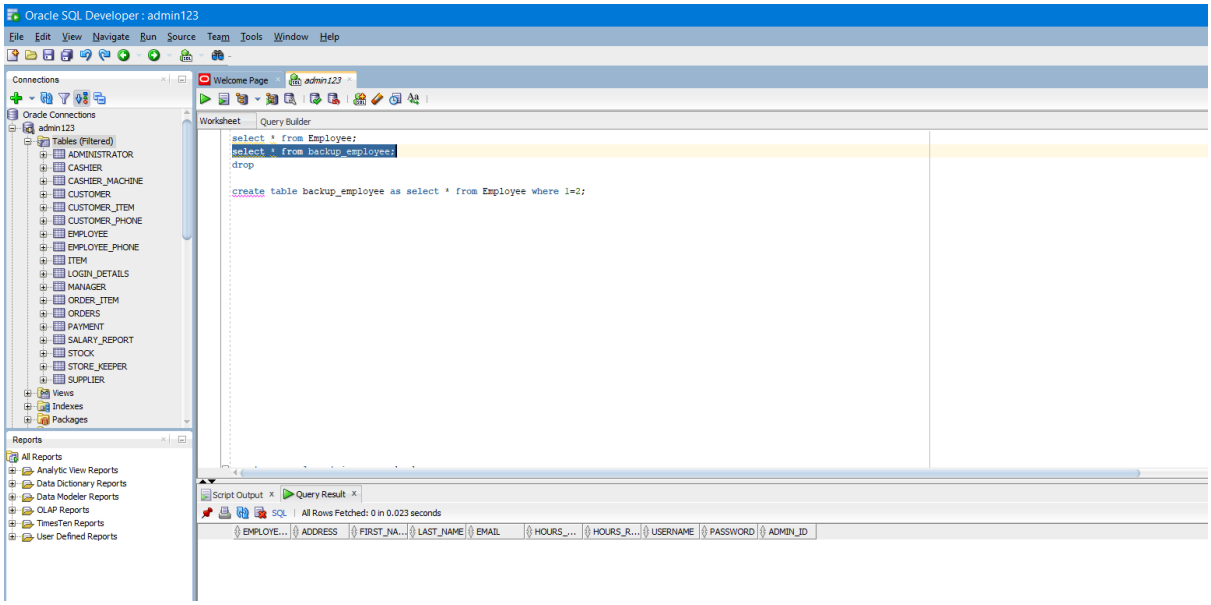


Let’s delete this record.






Also, we can use other backup and recovery techniques such as RMAN.

**Recovery Manager (RMAN)**

We can use backup database data by using RMAN or Recovery Manager in Oracle. This is an integrated recovery server which is provided by the oracle database. We can use this mechanism to backup data because of its offers many facilities such as backup more than one data file to disk or tape, backup achieved records to disk space or tape, restore data file from disk.

**% rman**

**RMAN> CONNECT TARGET SYS@prod**

**target database Password: password**

**connected to target database: PROD (DBID=39525561)**

**% rman**

**RMAN> CONNECT TARGET /**

**connected to target database: PROD (DBID=39525561)**

We can use the above commands to connect to RMAN. We can use this technique to keep backup files.

# 9. Conclusion

The database management system or DBMS is a vital tool for future implementations of business organizations. The main features offered by DBMS are query ability, backup, security, and computation, which are the need for a corporate system. Every company needs a proper, well planned, effective DBMS for profit-making. Because not having an adequate DBMS it can lead to many security problems not only that, but also it can lead to many financial issues too.

Supermarket Management System has to do with making appropriate efforts to stop the rising problem to all manual supermarket operations to enhance the operation of such a supermarket. In this project, the software or system that can be used to aid all supermarkets that are still operating manually has been successfully developed.