FACULTY OF COMPUTING & INFORMATION TECHNOLOGY
KING ABDULAZIZ UNIVERSITY

كلية الحاسبات وتقنية المعلومات
جامعة الملك عبدالعزيز

CPIT:240 Database Project

Group 3

Flower store

Instructor: Tariq Ahmed

Name	Student ID
Ibrahim Rabeh Aljohani	
Nawaf Fahad Altamimi	
Ahmed abdulrhman Alhuthayfi	
Riyadh majed Alsulami	

Contents

Phase 1

Project Description

A Flower Shop Database that is designed to manage the shop across multiple branches in Jeddah city. The first entity it has is the Inventory, which maintains three key pieces of information: an identification number, a product reference, and the available quantity of each product. The Product entity stores details including a unique number to distinguish this product from the rest, name, type, price, and a brief description. Branch is another with details like an identification Unique name, location, and contact number. The Order component records the id number of each order and the date it was placed. Linked to orders is the OrderDetails entity which holds a unique serial number, contains the quantity of the product ordered, and it's price per unit. Customer details such as unique identifier, name, address, and contact number are stored in the Customer entity. Each customer's affiliation with the Loyalty Customer Program is tracked, with details like tag number added to the customer unique identifier to distinguish it from the rest, discount rate,,and the discount duration. Supplier details are recorded with values including an identification number, address, email, phone number, and company name. Lastly, the Employee entity holds details like an employee ID, name, email, phone number, salary, and position.

Operations

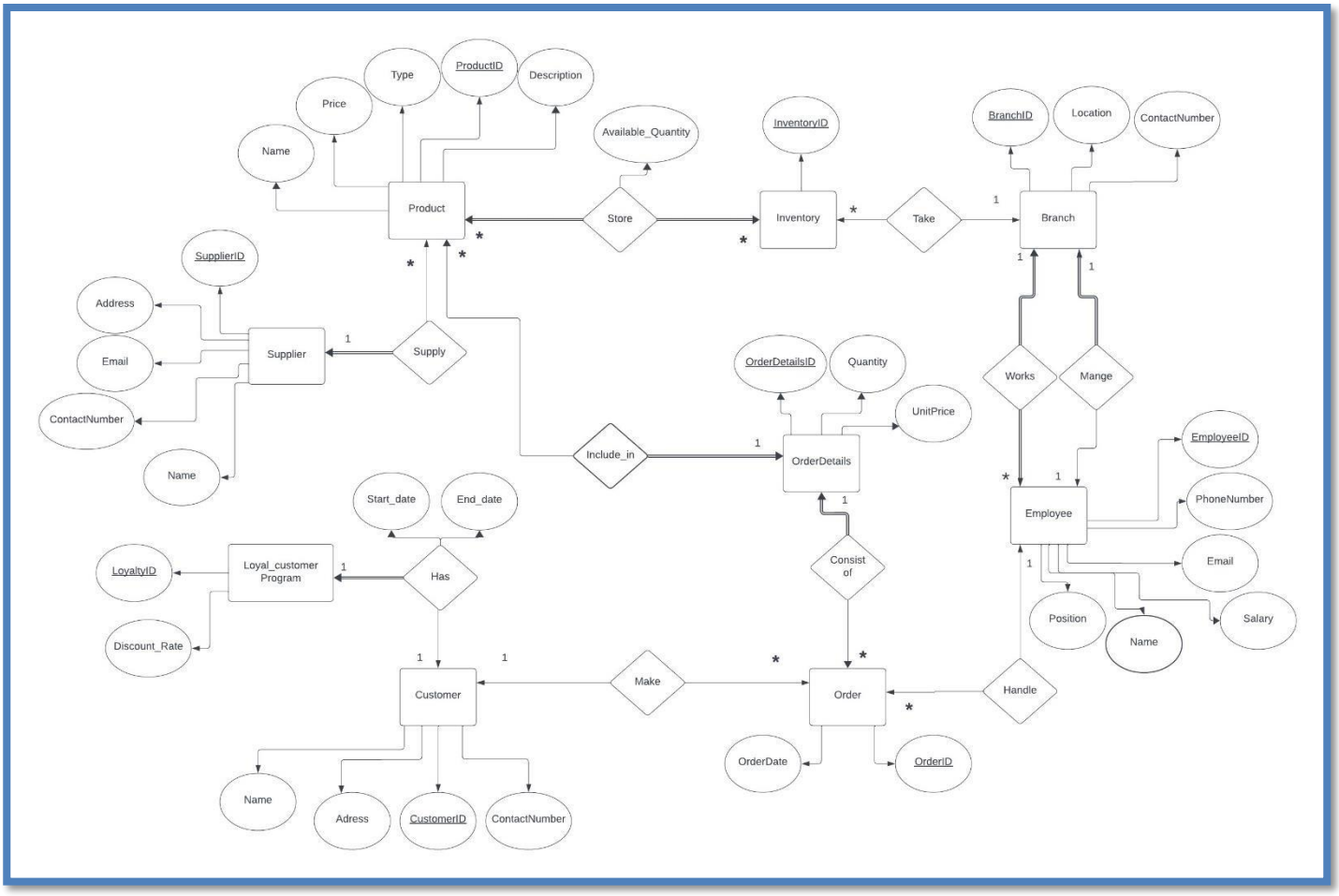
Each branch can hold multiple inventories and is also linked to the employees where each branch can have many employees, also at least one employee managing the branch. Employees are associated with handling multiple orders. Customers can place multiple orders, and these orders can contain many products each of which would be an entrie through the OrderDetails. The Product entity links to the inventory entity so that each Inventory can hold many products, and it connects with OrderDetails so many orderdetails entries can have one product, while also being supplied by one supplier. Suppliers have the capability to provide a number of products. Each product can appear in numerous order details. Lastly, each customer is part of loyalty program and they are entitled to benefits such as discounts.

Constraints and Rules for Each Entity

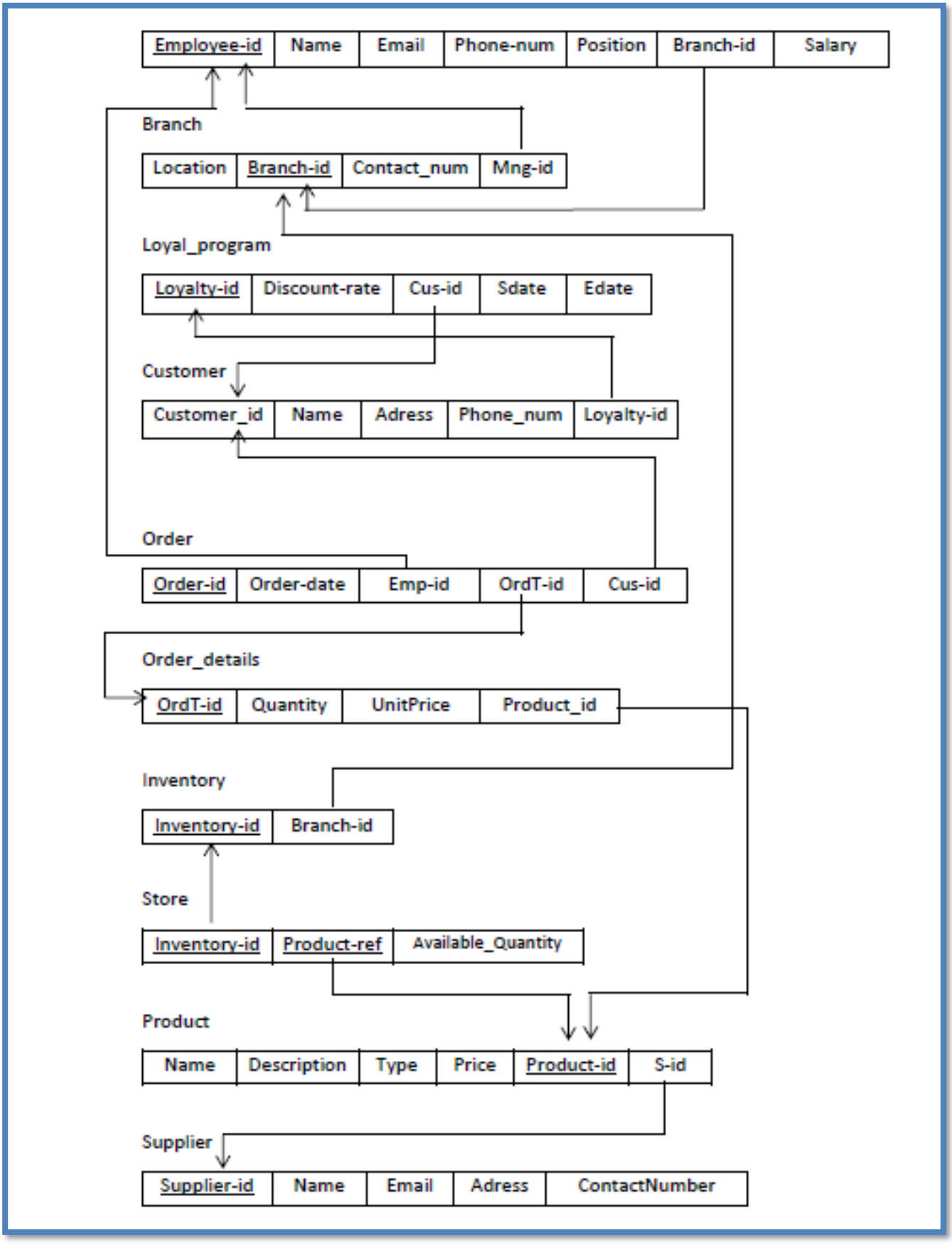
The Branch entity requires a unique location for each entry. Customer contact numbers are also unique in the Customer entity. The Loyalty Customer Program has specific rules where the discount rate must be within a 0% to 100% range and the program has a set duration marked by a start and end date, and each loyalty program is connected to one and only one customer. The Employee entity has a restriction that each employee can hold only one position within the company. The branch must have at least one manager.

Phase 2

ER diagram:



Relational Database diagram (RDB):



Normalization

To check this let us go through the tables in our database.

The Employee Table:

1. **All the attributes are atomic meaning they only hold 1 value = 1NF: True**
2. **There are no partial dependencies = 2NF: True**
3. **The non-prime attributes become directly (non-transitively) reliant on candidate keys. = 3NF: True.**

The Branch Table:

1. **All the attributes are atomic meaning they only hold 1 value = 1NF: True**
2. **There are no partial dependencies = 2NF: True**
3. **The non-prime attributes become directly (non-transitively) reliant on candidate keys. = 3NF: True.**

The Loyal_Program Table:

1. **All the attributes are atomic meaning they only hold 1 value = 1NF: True**
2. **There are no partial dependencies = 2NF: True**
3. **The non-prime attributes become directly (non-transitively) reliant on candidate keys. = 3NF: True.**

The Customer Table:

1. **All the attributes are atomic meaning they only hold 1 value = 1NF: True**
2. **There are no partial dependencies = 2NF: True**
3. **The non-prime attributes become directly (non-transitively) reliant on candidate keys. = 3NF: True.**

The Order Table:

1. **All the attributes are atomic meaning they only hold 1 value = 1NF: True**
2. **There are no partial dependencies = 2NF: True**
3. **The non-prime attributes become directly (non-transitively) reliant on candidate keys. = 3NF: True.**

The Order_details Table:

1. **All the attributes are atomic meaning they only hold 1 value = 1NF: True**
2. **There are no partial dependencies = 2NF: True**
3. **The non-prime attributes become directly (non-transitively) reliant on candidate keys. = 3NF: True.**

The Inventory Table:

1. **All the attributes are atomic meaning they only hold 1 value = 1NF: True**
2. **There are no partial dependencies = 2NF: True**
3. **The non-prime attributes become directly (non-transitively) reliant on candidate keys. = 3NF: True.**

The Product Table:

1. **All the attributes are atomic meaning they only hold 1 value = 1NF: True**
2. **There are no partial dependencies = 2NF: True**
3. **The non-prime attributes become directly (non-transitively) reliant on candidate keys. = 3NF: True.**

The Store Table:

1. **All the attributes are atomic meaning they only hold 1 value = 1NF: True**
2. **There are no partial dependencies = 2NF: True**
   The non-prime attributes become directly (non-transitively) reliant on candidate keys. = 3NF: True.

The Supplier Table:

1. **All the attributes are atomic meaning they only hold 1 value = 1NF: True**
2. **There are no partial dependencies = 2NF: True**
3. **The non-prime attributes become directly (non-transitively) reliant on candidate keys. = 3NF: True.**

Finally, we can conclude that all the relation in the first, second, third normal form.

Phase 3

Task1.1 : Implementation (DDL)

All the text files are included in the ZIP file.

```
CREATE TABLE Employee (
   Employee_id NUMBER(7) PRIMARY KEY,
   Name VARCHAR2(25) NOT NULL,
   Email VARCHAR2(35) UNIQUE,
   Phone_num NUMBER(10) NOT NULL UNIQUE,
   Position VARCHAR2(15) NOT NULL,
   Branch_id NUMBER(5),
   Salary NUMBER(7, 2) NOT NULL);
CREATE TABLE Branch (
   Branch_id NUMBER(5) PRIMARY KEY,
   Location VARCHAR2(30) NOT NULL UNIQUE,
   Contact_num NUMBER(10) NOT NULL UNIQUE,
   Mng_id NUMBER(7));
ALTER TABLE Employee
ADD CONSTRAINT Works
FOREIGN KEY (Branch_id)
REFERENCES Branch(Branch_id);
ALTER TABLE Branch
ADD CONSTRAINT Mng_ID
FOREIGN KEY (Mng_id)
REFERENCES Employee(Employee_id);
```

```
CREATE TABLE Customer (
   Customer_id NUMBER(7) PRIMARY KEY,
   Name VARCHAR2(25) NOT NULL,
   Address VARCHAR(30) NOT NULL,
   Phone_num NUMBER(10) NOT NULL UNIQUE,
   Loyalty_id NUMBER(7));
CREATE TABLE Loyal_program (
   Loyalty_id NUMBER(7) PRIMARY KEY,
   Discount_rate NUMBER(5, 2) NOT NULL,
   Sdate DATE NOT NULL,
   Edate DATE NOT NULL,
   Cus_id NUMBER(7));
ALTER TABLE Customer
ADD CONSTRAINT Loyalty_ID
FOREIGN KEY (Loyalty_id)
REFERENCES Loyal_program(Loyalty_id);
ALTER TABLE Loyal_program
ADD CONSTRAINT Cus_ID_Has
FOREIGN KEY (Cus_id)
REFERENCES Customer(Customer_id);
CREATE TABLE Inventory (
 Inventory_id VARCHAR2(7) PRIMARY KEY,
 Branch_id NUMBER(5),
 CONSTRAINT Take_By_Branch_ID FOREIGN KEY (Branch_id)
 REFERENCES Branch(Branch_id));
CREATE TABLE Supplier (
   Supplier_id VARCHAR2(10) PRIMARY KEY,
   Name VARCHAR2(30) NOT NULL,
```

```
  Email VARCHAR2(50),
  Address VARCHAR2(50),
  ContactNumber NUMBER(10) NOT NULL UNIQUE);
CREATE TABLE Product (
 Product_id VARCHAR2(25) PRIMARY KEY,
 Name VARCHAR2(30) NOT NULL,
 Description VARCHAR2(100),
 Type VARCHAR2(50),
 Price NUMBER(6, 2) NOT NULL,
 S_id VARCHAR2(10),
 CONSTRAINT Supplier_ID FOREIGN KEY (S_id) REFERENCES Supplier(Supplier_id));
CREATE TABLE Store (
 Inventory_id VARCHAR2(7),
 Product_ref VARCHAR2(25),
 Available_Quantity NUMBER(5) NOT NULL,
 PRIMARY KEY (Inventory_id, Product_ref),
 CONSTRAINT Inventory_ID FOREIGN KEY (Inventory_id)
 REFERENCES Inventory(Inventory_id),
 CONSTRAINT Product_ID FOREIGN KEY (Product_ref) REFERENCES Product(Product_id));
CREATE TABLE Order_details (
 OrdT_id VARCHAR2(25) PRIMARY KEY,
 Quantity NUMBER(2) NOT NULL,
 UnitPrice NUMBER(6, 2) NOT NULL,
 Proudct_ID VARCHAR2(25),
 CONSTRAINT Include_In FOREIGN KEY (Proudct_ID) REFERENCES Product(Product_id));
CREATE TABLE Orders (
 Order_id VARCHAR2(25) PRIMARY KEY,
 Order_date DATE NOT NULL,
```

```
 Emp_id NUMBER(7),
 OrdT_id VARCHAR2(25),
 Cus_id NUMBER(7),
 CONSTRAINT Handle_By_EmID FOREIGN KEY (Emp_id) REFERENCES Employee(Employee_id),
 CONSTRAINT Make_By FOREIGN KEY (Cus_id) REFERENCES Customer(Customer_id),
 CONSTRAINT Consist_of FOREIGN KEY (OrdT_id) REFERENCES Order_details(OrdT_id));
);
```

Task1.2 : Implementation (DML)

All the text files are included in the ZIP

INSERT INTO Employee (Employee_id, Name, Email, Phone_num, Position, Branch_id, Salary) VALUES (100106, 'Ahmed Alharbi', 'Ahmed7arbi@example.com', 0500000001, 'Manager', null, 4500.00);

INSERT INTO Employee (Employee_id, Name, Email, Phone_num, Position, Branch_id, Salary) VALUES (100107, 'Ali AboBakr', 'AliBaker@example.com', 0500000002, 'Manager', null, 4500.00);

INSERT INTO Employee (Employee_id, Name, Email, Phone_num, Position, Branch_id, Salary) VALUES (100108, 'Mohmaed Hussein', 'MohamedH@example.com', 0500000003, 'Assistant', null, 3200.00);

INSERT INTO Employee (Employee_id, Name, Email, Phone_num, Position, Branch_id, Salary) VALUES (100109, ' Kai Ezra', 'KaiEzra@example.com', 0500000004, 'Assistant', null, 3200.00);

INSERT INTO Employee (Employee_id, Name, Email, Phone_num, Position, Branch_id, Salary) VALUES (100110, 'Renad Hajjaj', 'RenadH@example.com', 0500000005, 'Manager', null, 4500.00);

INSERT INTO Employee (Employee_id, Name, Email, Phone_num, Position, Branch_id, Salary) VALUES (100111, 'Omar Al-Sadiq', 'OmarSadiq@example.com', 0500000006, 'Manager', null, 4500.00);

INSERT INTO Employee (Employee_id, Name, Email, Phone_num, Position, Branch_id, Salary) VALUES (100112, 'Sara Muhand', 'MuhSara@example.com', 0500000007, 'Manager', null, 4500.00);

INSERT INTO Employee (Employee_id, Name, Email, Phone_num, Position, Branch_id, Salary) VALUES (100113, 'Saeed Al-Zahrani', 'SaeedZa7@example.com', 0500000008, 'Assistant', null, 3200.00);

INSERT INTO Employee (Employee_id, Name, Email, Phone_num, Position, Branch_id, Salary) VALUES (100114, 'Manaar', 'Manaar112@example.com', 0500000009, 'Assistant', null, 3200.00);

INSERT INTO Employee (Employee_id, Name, Email, Phone_num, Position, Branch_id, Salary) VALUES (100115, 'Mohamed Alghamdi', 'MAlghamdi@example.com', 0500000011, 'Assistant', null, 3200.00);

```
INSERT INTO Branch (Branch_id, Location, Contact_num, Mng_id) VALUES (10006, 'AlMarwah', 0510000000, 100106);

INSERT INTO Branch (Branch_id, Location, Contact_num, Mng_id) VALUES (10007, 'Obhur', 0520000000, 100107);

INSERT INTO Branch (Branch_id, Location, Contact_num, Mng_id) VALUES (10008, 'AlRehab', 0530000000, 100112);

INSERT INTO Branch (Branch_id, Location, Contact_num, Mng_id) VALUES (10009, 'AlHamdaniyah', 0540000000, 100110);

INSERT INTO Branch (Branch_id, Location, Contact_num, Mng_id) VALUES (10010, 'AsSalamah', 0550000000, 100107);


UPDATE Employee SET Branch_id = 10006 WHERE Employee_id = 100106;

UPDATE Employee SET Branch_id = 10007 WHERE Employee_id = 100107;

UPDATE Employee SET Branch_id = 10007 WHERE Employee_id = 100108;

UPDATE Employee SET Branch_id = 10009 WHERE Employee_id = 100109;

UPDATE Employee SET Branch_id = 10008 WHERE Employee_id = 100110;

UPDATE Employee SET Branch_id = 10009 WHERE Employee_id = 100111;

UPDATE Employee SET Branch_id = 10010 WHERE Employee_id = 100112;

UPDATE Employee SET Branch_id = 10008 WHERE Employee_id = 100113;

UPDATE Employee SET Branch_id = 10006 WHERE Employee_id = 100114;

UPDATE Employee SET Branch_id = 10010 WHERE Employee_id = 100115;


INSERT INTO Customer (Customer_id, Name, Address, Phone_num, Loyalty_id) VALUES (200206, 'Youssef Al-Mansouri', 'AlMarwah, Jeddah', 0500100000, Null);

INSERT INTO Customer (Customer_id, Name, Address, Phone_num, Loyalty_id) VALUES (200207, 'Amina Al-Faraj', 'AlSafa, Jeddah', 0500200000, Null);

INSERT INTO Customer (Customer_id, Name, Address, Phone_num, Loyalty_id) VALUES (200208, 'Sara Al-Najjar', 'AlBawadi, Jeddah', 0500300000, Null);

INSERT INTO Customer (Customer_id, Name, Address, Phone_num, Loyalty_id) VALUES (200209, 'Tariq AlOmari', 'Mishrifah, Jeddah', 0500400000, Null);
```

```
INSERT INTO Customer (Customer_id, Name, Address, Phone_num, Loyalty_id) VALUES
(200210, 'Yasmine Al-Ansari', 'Bryman, Jeddah', 0500500000, Null);

INSERT INTO Customer (Customer_id, Name, Address, Phone_num, Loyalty_id) VALUES
(200211, 'Fahad Al-Hariri', 'Obhur, Jeddah', 0500600000, Null);

INSERT INTO Customer (Customer_id, Name, Address, Phone_num, Loyalty_id) VALUES
(200212, 'Ahlam Al-Sharqawi', 'AlZahra, Jeddah', 0500700000, Null);


INSERT INTO Loyal_program (Loyalty_id, Discount_rate, Sdate, Edate, Cus_id) VALUES
(300311, 5.00, TO_DATE('2024-02-13', 'YYYY-MM-DD'), TO_DATE('2026-02-13', 'YYYY-
MM-DD'), 200211);

INSERT INTO Loyal_program (Loyalty_id, Discount_rate, Sdate, Edate, Cus_id) VALUES
(300307, 7.50, TO_DATE('2023-12-31', 'YYYY-MM-DD'), TO_DATE('2025-12-31', 'YYYY-
MM-DD'), 200207);

INSERT INTO Loyal_program (Loyalty_id, Discount_rate, Sdate, Edate, Cus_id) VALUES
(300308, 15.45, TO_DATE('2022-07-01', 'YYYY-MM-DD'), TO_DATE('2024-07-01', 'YYYY-
MM-DD'), 200208);

INSERT INTO Loyal_program (Loyalty_id, Discount_rate, Sdate, Edate, Cus_id) VALUES
(300312, 10.00, TO_DATE('2023-04-26', 'YYYY-MM-DD'), TO_DATE('2025-04-26', 'YYYY-
MM-DD'), 200212);

INSERT INTO Loyal_program (Loyalty_id, Discount_rate, Sdate, Edate, Cus_id) VALUES
(300310, 2.00, TO_DATE('2024-05-01', 'YYYY-MM-DD'), TO_DATE('2026-05-01', 'YYYY-
MM-DD'), 200210);


UPDATE Customer SET Loyalty_id = 300307 WHERE Customer_id = 200207;

UPDATE Customer SET Loyalty_id = 300308 WHERE Customer_id = 200208;

UPDATE Customer SET Loyalty_id = 300310 WHERE Customer_id = 200210;

UPDATE Customer SET Loyalty_id = 300311 WHERE Customer_id = 200211;

UPDATE Customer SET Loyalty_id = 300312 WHERE Customer_id = 200212;

INSERT INTO Inventory (Inventory_id, Branch_id) VALUES ('INV006', 10006);

INSERT INTO Inventory (Inventory_id, Branch_id) VALUES ('INV007', 10007);

INSERT INTO Inventory (Inventory_id, Branch_id) VALUES ('INV008', 10008);
```

```
INSERT INTO Inventory (Inventory_id, Branch_id) VALUES ('INV009', 10009);

INSERT INTO Inventory (Inventory_id, Branch_id) VALUES ('INV010', 10010);


INSERT INTO Supplier (Supplier_id, Name, Email, Address, ContactNumber) VALUES
('S1006', 'wardat farah lilzuhur', '-', 'Al Sharafeyah, Jeddah', 0500532114);

INSERT INTO Supplier (Supplier_id, Name, Email, Address, ContactNumber) VALUES
('S1007', 'Ghazal Flower', '-', 'Al-Hamra, Jeddah', 0501376120);

INSERT INTO Supplier (Supplier_id, Name, Email, Address, ContactNumber) VALUES
('S1008', 'warud alwahat alziraeia', '-', 'Al Rabia, Jeddah', 0566109997);

INSERT INTO Supplier (Supplier_id, Name, Email, Address, ContactNumber) VALUES
('S1009', 'Rose dreams of Natural flower', '-', 'Al-Manakh, Riyadh', 0508873638);

INSERT INTO Supplier (Supplier_id, Name, Email, Address, ContactNumber) VALUES
('S1010', 'Multiflora', '-', 'Al Sharafeyah, Jeddah', 0126144778);


INSERT INTO Product (Product_id, Name, Description, Type, Price, S_id) VALUES ('P1006',
'Spray Roses', 'Color:White/LightPeach NumberofStems:01 BunchHeight(cm):40
CountryofOrigin:Netherlands', 'A Single Stem', 10.00, 'S1010');

INSERT INTO Product (Product_id, Name, Description, Type, Price, S_id) VALUES ('P1007',
'Chrysanthemum', 'Color:White/Yellow NumberofStems:01 BunchHeight(cm):60
CountryofOrigin:Holland', 'A Single Stem', 10.00, 'S1010');

INSERT INTO Product (Product_id, Name, Description, Type, Price, S_id) VALUES ('P1008',
'Rose', 'Color:Color:White/Fushia/Red/Pink/Purple NumberofStems:05
BunchHeight(cm):50 CountryofOrigin:India', 'Bunch Of Stems', 15.00, 'S1007');

INSERT INTO Product (Product_id, Name, Description, Type, Price, S_id) VALUES ('P1009',
'Baby Roses', 'Color:Yellow/Purple/Pink/Red NumberofStems:05 BunchHeight(cm):50
CountryofOrigin:Kenya', 'Bunch Of Stems', 20.00, 'S1006');

INSERT INTO Product (Product_id, Name, Description, Type, Price, S_id) VALUES ('P1010',
'Lillium', 'Color:Fuchsia NumberofStems:07 BunchHeight(cm):40 CountryofOrigin:Kuwait',
'Bunch Of Stems', 30.00, 'S1009');

INSERT INTO Product (Product_id, Name, Description, Type, Price, S_id) VALUES ('P1011',
'Rose Bouquet', 'Color:White/Fushia/Red/Pink/Purple NumberofStems:50
BunchHeight(cm):50 Country of Origin :Kenya', 'Bouquet', 120.00, 'S1007');
```

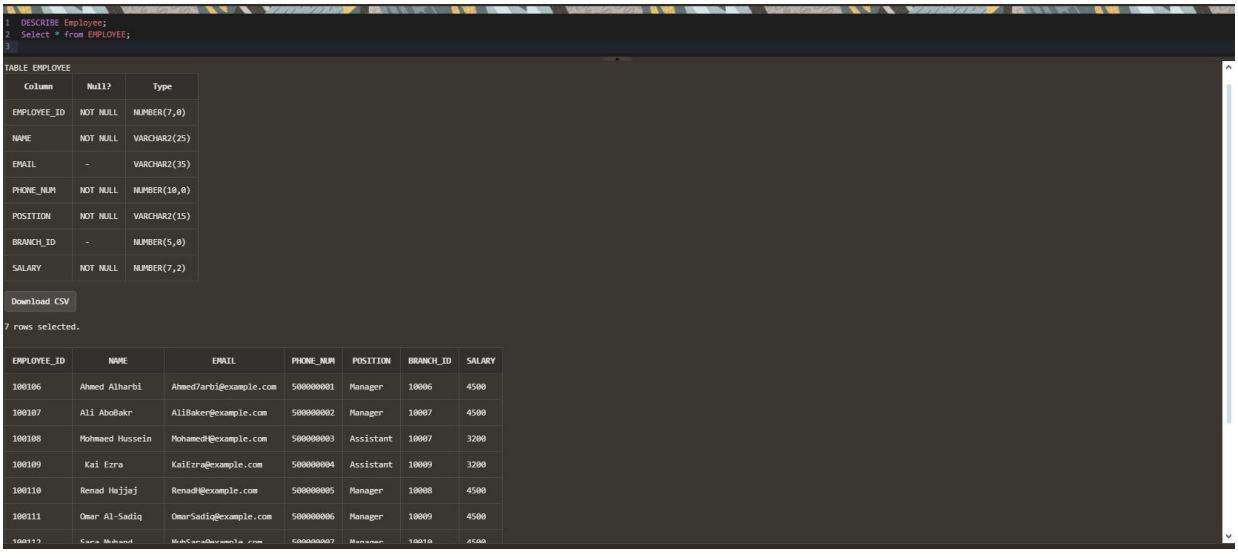
```
INSERT INTO Product (Product_id, Name, Description, Type, Price, S_id) VALUES ('P1012',
'Gerbera', 'Color:OffWhite NumberofStems:10 BunchHeight(cm):45
CountryofOrigin:Holland','Bunch Of Stems', 25.00, 'S1008');


Insert into Store (Inventory_id, Product_ref, Available_Quantity) VALUES ('INV006', 'P1006',
25);

Insert into Store (Inventory_id, Product_ref, Available_Quantity) VALUES ('INV006', 'P1010',
30);

Insert into Store (Inventory_id, Product_ref, Available_Quantity) VALUES ('INV006', 'P1009',
15);

Insert into Store (Inventory_id, Product_ref, Available_Quantity) VALUES ('INV007', 'P1008',
25);

Insert into Store (Inventory_id, Product_ref, Available_Quantity) VALUES ('INV007', 'P1011',
32);

Insert into Store (Inventory_id, Product_ref, Available_Quantity) VALUES ('INV008', 'P1006',
30);

Insert into Store (Inventory_id, Product_ref, Available_Quantity) VALUES ('INV008', 'P1008',
27);

Insert into Store (Inventory_id, Product_ref, Available_Quantity) VALUES ('INV008', 'P1010',
13);

Insert into Store (Inventory_id, Product_ref, Available_Quantity) VALUES ('INV008', 'P1012',
27);

Insert into Store (Inventory_id, Product_ref, Available_Quantity) VALUES ('INV009', 'P1006',
43);

Insert into Store (Inventory_id, Product_ref, Available_Quantity) VALUES ('INV009', 'P1007',
28);

Insert into Store (Inventory_id, Product_ref, Available_Quantity) VALUES ('INV009', 'P1008',
51);

Insert into Store (Inventory_id, Product_ref, Available_Quantity) VALUES ('INV009', 'P1011',
15);

Insert into Store (Inventory_id, Product_ref, Available_Quantity) VALUES ('INV009', 'P1012',
23);
```

```
Insert into Store (Inventory_id, Product_ref, Available_Quantity) VALUES ('INV010', 'P1008', 46);

Insert into Store (Inventory_id, Product_ref, Available_Quantity) VALUES ('INV010', 'P1011', 9);

Insert into Store (Inventory_id, Product_ref, Available_Quantity) VALUES ('INV010', 'P1012', 32);


INSERT INTO Order_details (OrdT_id, Quantity, UnitPrice, Proudct_ID) VALUES ('OD1006', 4, 40.00, 'P1006');

INSERT INTO Order_details (OrdT_id, Quantity, UnitPrice, Proudct_ID) VALUES ('OD1007', 1, 114.00, 'P1011');

INSERT INTO Order_details (OrdT_id, Quantity, UnitPrice, Proudct_ID) VALUES ('OD1008', 4, 92.50, 'P1012');

INSERT INTO Order_details (OrdT_id, Quantity, UnitPrice, Proudct_ID) VALUES ('OD1009', 3, 60.00, 'P1009');

INSERT INTO Order_details (OrdT_id, Quantity, UnitPrice, Proudct_ID) VALUES ('OD1010', 7, 102.90, 'P1008');


INSERT INTO Orders (Order_id, Order_date, Emp_id, OrdT_id, Cus_id) VALUES ('O1006', TO_DATE('2024-05-25', 'YYYY-MM-DD'), 100113, 'OD1006', 200206);

INSERT INTO Orders (Order_id, Order_date, Emp_id, OrdT_id, Cus_id) VALUES ('O1007', TO_DATE('2024-05-13', 'YYYY-MM-DD'), 100108, 'OD1007', 200211);

INSERT INTO Orders (Order_id, Order_date, Emp_id, OrdT_id, Cus_id) VALUES ('O1008', TO_DATE('2024-07-05', 'YYYY-MM-DD'), 100113, 'OD1008', 200207);

INSERT INTO Orders (Order_id, Order_date, Emp_id, OrdT_id, Cus_id) VALUES ('O1009', TO_DATE('2024-06-01', 'YYYY-MM-DD'), 100113, 'OD1009', 200209);

INSERT INTO Orders (Order_id, Order_date, Emp_id, OrdT_id, Cus_id) VALUES ('O1010', TO_DATE('2024-05-15', 'YYYY-MM-DD'), 100115, 'OD1010', 200212);
```

Task1.3 : Description of Tables

This image shows the structure of the 'Employee' table in the database. Each row represents an employee record, and the columns contain specific information about each employee.

- **EMPLOYEE_ID**: This column stores a unique employee ID number (contain of a NUMBER 7 digits). It is **NOT NULL**, meaning every employee must have a unique identifier in this table.
- **NAME**: This column stores the employee's name (contain of VARCHAR2 data type with a maximum length of 25 characters). The name field is also **NOT NULL**, ensuring all employees have their names recorded.
- **EMAIL**: This column stores the employee's email address (contain of VARCHAR2 data type with a maximum length of 35 characters to accommodate most email addresses). While some organizations might require employee email addresses, we can assume the EMAIL field is **NULL** for cases where an email isn't provided (according to the Manager of the Company).
- **PHONE NUM**: This column stores the employee's phone number (likely a NUMBER data type with 10 digits).
- **POSITION**: This column stores the employee's job title (contain of VARCHAR2 data type with a maximum length of 15 characters). the POSITION field is **NOT NULL** as all employees must have assigned job titles.
- **BRANCH_ID**: column might indicate the branch the employee is assigned to, and this column stores a foreign key referencing the 'Branch' table. (contain of NUMBER data type with 5 digits).
- **SALARY**: This column stores the employee's salary (contain of NUMBER data type with 7 digits and 2 decimal places). Salaries are **NOT NULL** as all employees presumably have a designated salary.
  the image also shows a table populated with ten rows, each representing an em-



```
1  DESCRIBE Employee;
2  Select * from EMPLOYEE;
3
```

TABLE EMPLOYEE

Column	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(7,0)
NAME	NOT NULL	VARCHAR2(25)
EMAIL	-	VARCHAR2(35)
PHONE_NUM	NOT NULL	NUMBER(10,0)
POSITION	NOT NULL	VARCHAR2(15)
BRANCH_ID	-	NUMBER(5,0)
SALARY	NOT NULL	NUMBER(7,2)

Download CSV

7 rows selected.

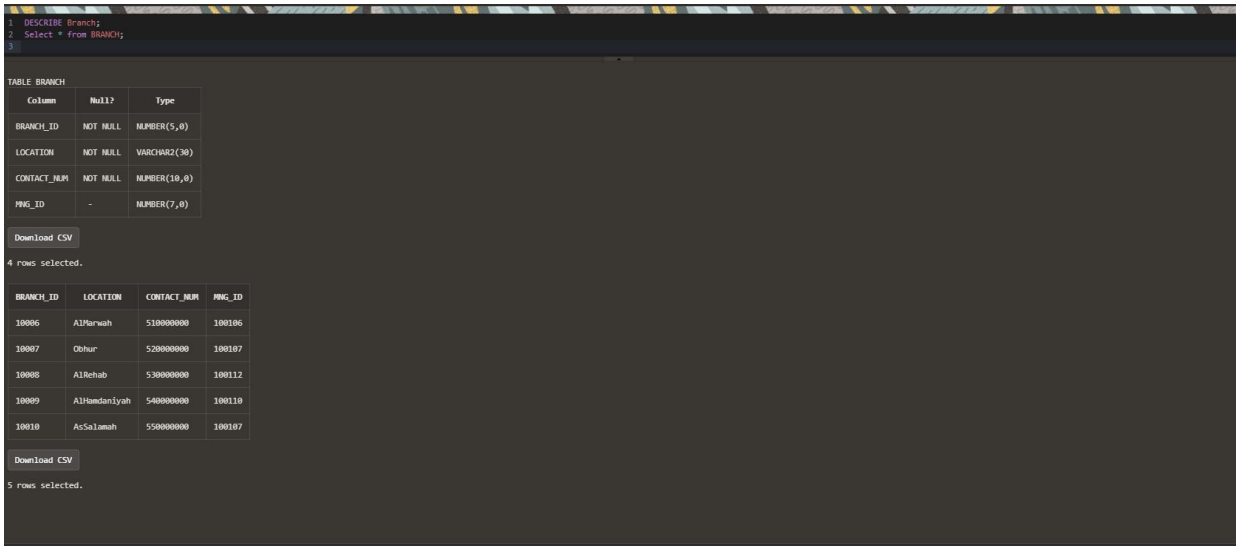
EMPLOYEE_ID	NAME	EMAIL	PHONE_NUM	POSITION	BRANCH_ID	SALARY
100106	Ahmed Alharbi	Ahmed7arbi@example.com	500000001	Manager	10006	4500
100107	Ali AboBakr	AliBaker@example.com	500000002	Manager	10007	4500
100108	Mohmaed Hussein	Mohamed#@example.com	500000003	Assistant	10007	3200
100109	Kai Ezra	KaiEzra@example.com	500000004	Assistant	10009	3200
100110	Renad Hajjaj	Renad#@example.com	500000005	Manager	10008	4500
100111	Omar Al-Sadiq	OmarSadiq@example.com	500000006	Manager	10009	4500
100112	Sara Mubarak	MubSara@example.com	500000007	Manager	10010	4500

ployee with their respective details.

This table image depicts the structure of the 'Branch' table within the database. Each row represents a branch location, and the columns contain specific details about each branch.

- **BRANCH_ID:** This column stores a unique identifier for each branch (contain of NUMBER data type with 5 digits). It is designated as NOT NULL, meaning every branch must have a distinct ID within the table.
- **LOCATION:** This column stores the physical location of the branch (contain of VARCHAR2 data type with 30 characters). The LOCATION field is also NOT NULL, ensuring all branches have their locations recorded.
- **CONTACT_NUM:** This column stores the phone number for the branch (contain of NUMBER data type 10 digits). Like LOCATION, CONTACT_NUM is also designated NOT NULL.
- **MNG_ID:** This column acts as a foreign key referencing the 'Employee' table. It contains a NUMBER data type. Its store the Manager of the branch.

And the image show table populated with five rows, each representing a branch with its respective details. This data can be used to analyze the distribution of branches, their locations, contact information, and the managers assigned to them. It provides valuable insights into the structure and organization of the company's branches.



```
1  DESCRIBE Branch;
2  Select * from BRANCH;
3
```

TABLE BRANCH

Column	Null?	Type
BRANCH_ID	NOT NULL	NUMBER(5,0)
LOCATION	NOT NULL	VARCHAR2(30)
CONTACT_NUM	NOT NULL	NUMBER(10,0)
MNG_ID	-	NUMBER(7,0)

Download CSV

4 rows selected.

BRANCH_ID	LOCATION	CONTACT_NUM	MNG_ID
10006	AlMarwah	510000000	100106
10007	Obhur	520000000	100107
10008	AlRehab	530000000	100112
10009	AlHamdaniyah	540000000	100110
10010	AsSalamah	550000000	100107

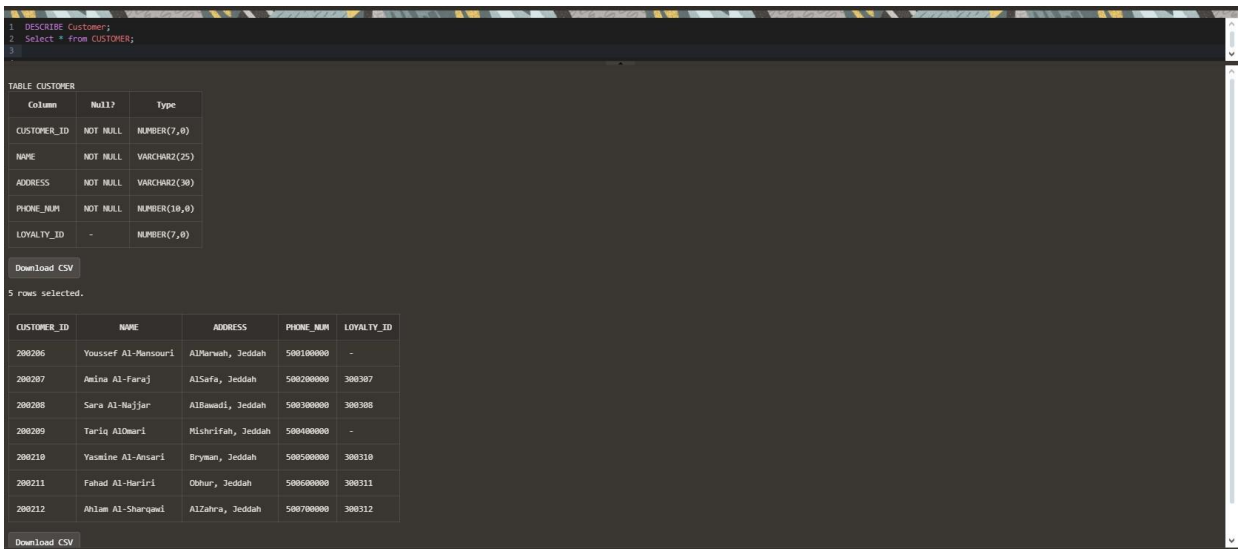
Download CSV

5 rows selected.

This table represents the 'Customer' table within your database. Each row stores information about a single customer. Here's a breakdown of the columns:

- **Customer_id:** This column stores a unique customer ID number (contain of type NUMBER with 7 digits). The PRIMARY KEY constraint ensures each customer has a distinct identifier within the table.
- **Name:** This column stores the customer's name (contain of type VARCHAR2 with a maximum length of 25 characters). The NOT NULL constraint ensures that every customer record has a name entered.
- **Address:** This column stores the customer's address (contain of type VARCHAR2 with a maximum length of 30 characters). Similar to the Name column, it's also designated NOT NULL, mandating that every customer has an address recorded.
- **Phone_num:** This column stores the customer's phone number (contain of type NUMBER with 10 digits). The NOT NULL constraint ensures every customer record has a phone number entered, and the UNIQUE constraint guarantees that no two customers share the same phone number.
- **Loyalty_id:** This column stores a customer loyalty program ID number (of type NUMBER with 7 digits). It acts as a foreign key referencing the 'Loyal_program' table. This relationship helps maintain data integrity by ensuring Loyalty_id values in the Customer table correspond to existing entries in the Loyal_program table.

The other table populated with seven rows, each representing a customer with their respective details. This data can be utilized to analyze customer demographics, address distribution, loyalty program participation, and more, providing insights into the customer base of the business.



```
1  DESCRIBE Customer;
2  Select * from CUSTOMER;
3
```

TABLE CUSTOMER

Column	Null?	Type
CUSTOMER_ID	NOT NULL	NUMBER(7,0)
NAME	NOT NULL	VARCHAR2(25)
ADDRESS	NOT NULL	VARCHAR2(30)
PHONE_NUM	NOT NULL	NUMBER(10,0)
LOYALTY_ID	-	NUMBER(7,0)

Download CSV

5 rows selected.

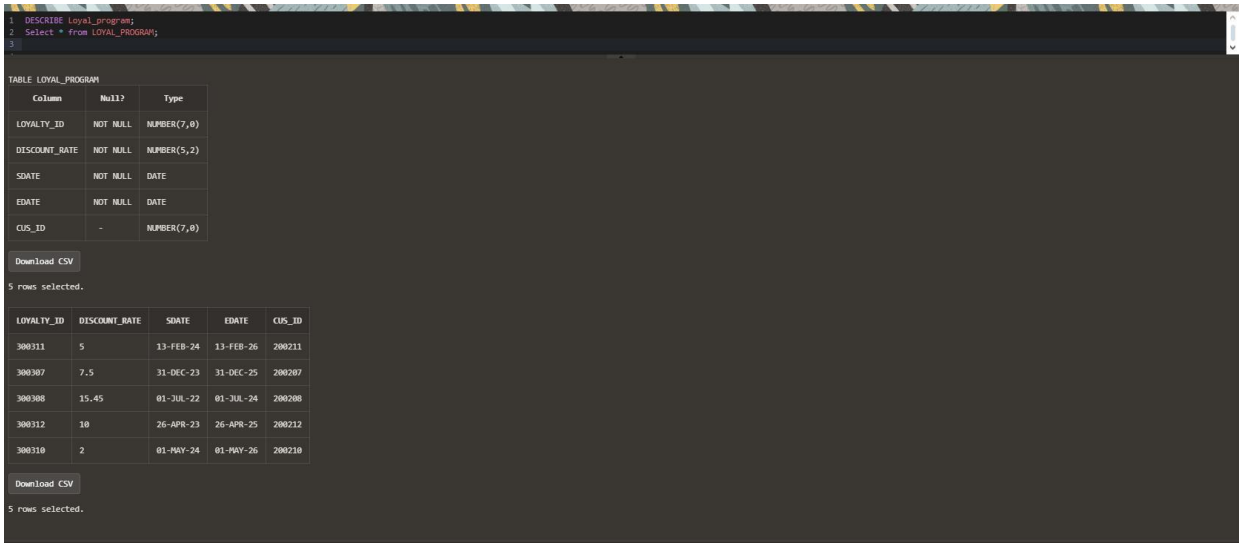
CUSTOMER_ID	NAME	ADDRESS	PHONE_NUM	LOYALTY_ID
200206	Youssef Al-Mansouri	AlMarwah, Jeddah	500100000	-
200207	Amina Al-Faraj	AlSafa, Jeddah	500200000	300307
200208	Sara Al-Najjar	AlBawadi, Jeddah	500300000	300308
200209	Tariq AlOmari	Mishrifah, Jeddah	500400000	-
200210	Yasmine Al-Ansari	Bryman, Jeddah	500500000	300310
200211	Fahad Al-Hariri	Obhur, Jeddah	500600000	300311
200212	Ahlam Al-Sharqawi	AlZahra, Jeddah	500700000	300312

Download CSV

And here a define of the structure of the 'Loyal_program' table, Each row represents a loyalty program membership. Here's a breakdown of the columns and their constraints:

- **Loyalty_id:** This column stores a unique identifier for each loyalty program record (Its a NUMBER data type with 7 digits). The PRIMARY KEY constraint ensures that each loyalty program has a distinct ID within this table.
- **Discount_rate:** This column stores the discount rate offered to loyalty program members (Its a NUMBER data type with 5 digits and 2 decimal places to represent percentages). The NOT NULL constraint guarantees that a discount rate is specified for every loyalty program entry.
- **Sdate:** This column stores the start date of the loyalty program membership (DATE data type). The NOT NULL constraint ensures a start date is recorded for every program membership.
- **Edate:** This column stores the end date of the loyalty program membership (DATE data type). The NOT NULL constraint ensures an end date is recorded for every program membership.
- **Cus_id:** This column stores a foreign key referencing the 'Customer' table. (It contains a NUMBER data type with 7 digits) and establishes a link between customer records and their loyalty program memberships. The NOT NULL constraint ensures a customer_ID is linked to every loyalty program record. This relationship helps maintain data integrity by ensuring Cus_id values in the Loyal_program table correspond to existing customer IDs in the 'Customer' table.

the Loyal_program table populated with five rows, each representing a loyalty program entry with its respective details. The Cus_id column establishes a relationship with the Customer table, allowing for tracking which customers are enrolled in which loyalty programs and the associated discount rates and dates.



```
1  DESCRIBE Loyal_program;
2  Select * from LOYAL_PROGRAM;
3
```

TABLE LOYAL_PROGRAM

Column	Null?	Type
LOYALTY_ID	NOT NULL	NUMBER(7,0)
DISCOUNT_RATE	NOT NULL	NUMBER(5,2)
SDATE	NOT NULL	DATE
EDATE	NOT NULL	DATE
CUS_ID	-	NUMBER(7,0)

Download CSV

5 rows selected.

LOYALTY_ID	DISCOUNT_RATE	SDATE	EDATE	CUS_ID
300311	5	13-FEB-24	13-FEB-26	200211
300307	7.5	31-DEC-23	31-DEC-25	200207
300308	15.45	01-JUL-22	01-JUL-24	200208
300312	10	26-APR-23	26-APR-25	200212
300310	2	01-MAY-24	01-MAY-26	200210

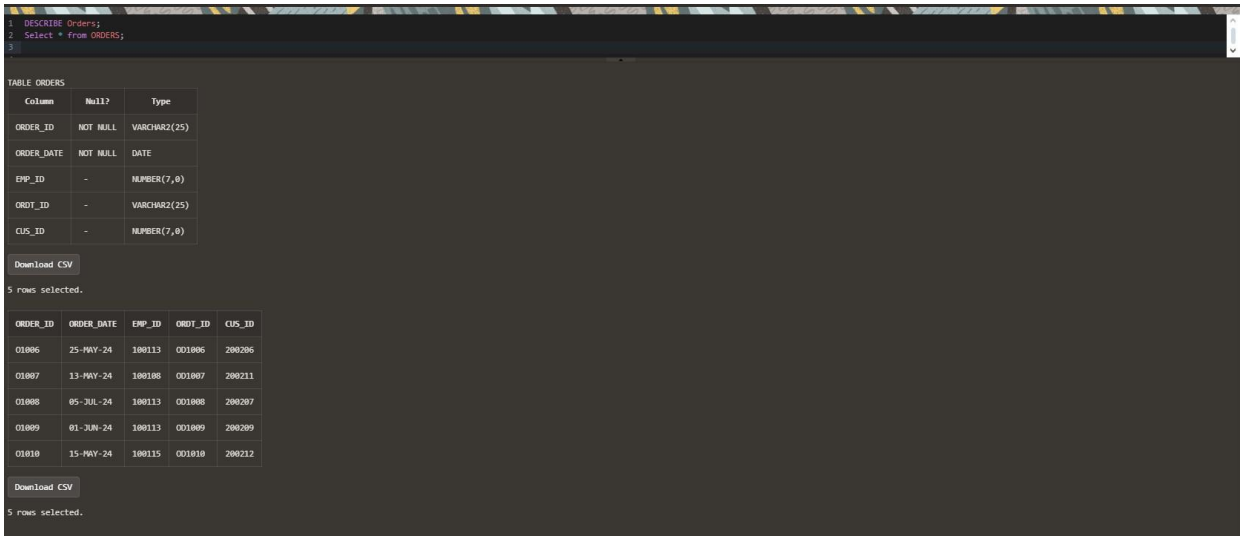
Download CSV

5 rows selected.

This code image defines the structure of the 'Orders' table, each row represents a single order placed by a customer. And here's a breakdown of the columns and their constraints:

- **Order_id:** This column stores a unique identifier for each order (VARCHAR2 data type with a maximum length of 25 characters). The PRIMARY KEY constraint ensures that each order record has a distinct identifier within the table.
- **Order_date:**This column stores the date the order was placed (DATE data type). The NOT NULL constraint mandates that a date is recorded for every order.
- **Emp_id:** This column stores a foreign key referencing the 'Employee' table. (It contains a NUMBER data type with 7 digits) and establishes a link between the employee who processed the order and the order itself. The NOT NULL constraint ensures an employee ID is linked to every order record. This relationship helps maintain data integrity by ensuring Emp_id values in the Orders table correspond to existing employee IDs in the 'Employee' table.
- **OrdT_id:** This column stores a foreign key referencing another table, named 'Order_details'. It's a VARCHAR2 data type with a maximum length of 25 characters. The NOT NULL constraint ensures an order details ID is linked to every order record.
- **Cus_id:** This column stores a foreign key referencing the 'Customer' table. (It contains a NUMBER data type with 7 digits) and establishes a link between the customer who placed the order and the order itself. The NOT NULL constraint ensures a customer ID is linked to every order record. This relationship helps maintain data integrity by ensuring Cus_id values in the Orders table correspond to existing customer IDs in the 'Customer' table.

the Orders table have a five rows, each representing an order with its respective details. This data can be used to analyze order dates, employee performance in handling orders, types of orders placed, and customer order patterns, providing insights into the company's sales operations.



```
1  DESCRIBE Orders;
2  Select * from ORDERS;
3
```

TABLE ORDERS

Column	Null?	Type
ORDER_ID	NOT NULL	VARCHAR2(25)
ORDER_DATE	NOT NULL	DATE
EMP_ID	-	NUMBER(7,0)
ORDT_ID	-	VARCHAR2(25)
CUS_ID	-	NUMBER(7,0)

Download CSV

5 rows selected.

ORDER_ID	ORDER_DATE	EMP_ID	ORDT_ID	CUS_ID
O1006	25-MAY-24	100113	OD1006	200206
O1007	13-MAY-24	100108	OD1007	200211
O1008	05-JUL-24	100113	OD1008	200207
O1009	01-JUN-24	100113	OD1009	200209
O1010	15-MAY-24	100115	OD1010	200212

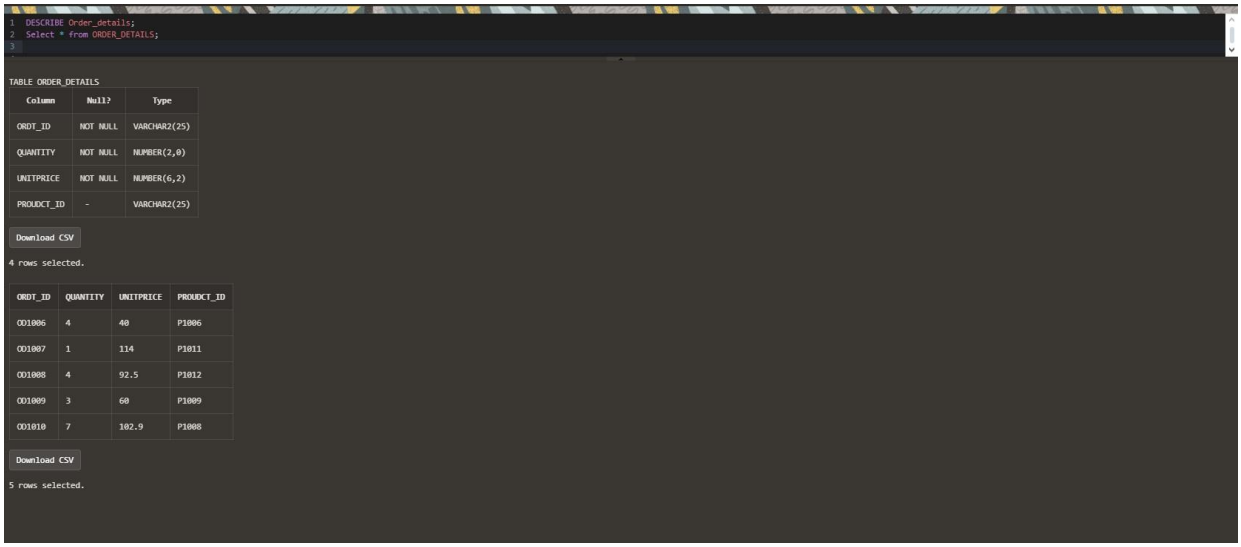
Download CSV

5 rows selected.

This image shows a table structure, named 'Order_details', represents the details of individual items within customer orders stored in your database. Here's a breakdown of the columns and their constraints:

- **OrdT_id:** This column likely stores a unique identifier for each order detail record (its a VARCHAR2 data type with a maximum length of 25 characters). The PRIMARY KEY constraint ensures that each record has a distinct identifier within this table.
- **Quantity:** This column stores the quantity of a particular product ordered (its a NUMBER data type with 2 digits). The NOT NULL constraint ensures a quantity is specified for every order detail record.
- **UnitPrice:** This column stores the unit price of the product at the time the order was placed (its a NUMBER data type with 6 digits and 2 decimal places). The NOT NULL constraint ensures a unit price is recorded for every order detail record.
- **Proudct_ID:** This column stores a foreign key referencing the 'Product' table. (Its contains a VARCHAR2 data type with a maximum length of 25 characters) and establishes a link between product information and the specific items included in an order. The NOT NULL constraint ensures a product ID is linked to every order detail record. This relationship helps maintain data integrity by ensuring Proudct_ID values correspond to existing product entries in the 'Product' table.

the Order_details table have a five rows, each representing an order detail with its respective details. This data can be used to analyze the quantity and pricing of products ordered, understand the composition of each order, and calculate total order values, providing insights into the company's sales transactions.



```
1  DESCRIBE Order_details;
2  Select * from ORDER_DETAILS;
3
```

TABLE ORDER_DETAILS

Column	Null?	Type
ORDT_ID	NOT NULL	VARCHAR2(25)
QUANTITY	NOT NULL	NUMBER(2,0)
UNITPRICE	NOT NULL	NUMBER(6,2)
PROUDCT_ID	-	VARCHAR2(25)

Download CSV

4 rows selected.

ORDT_ID	QUANTITY	UNITPRICE	PROUDCT_ID
OD1006	4	40	P1006
OD1007	1	114	P1011
OD1008	4	92.5	P1012
OD1009	3	60	P1009
OD1010	7	102.9	P1008

Download CSV

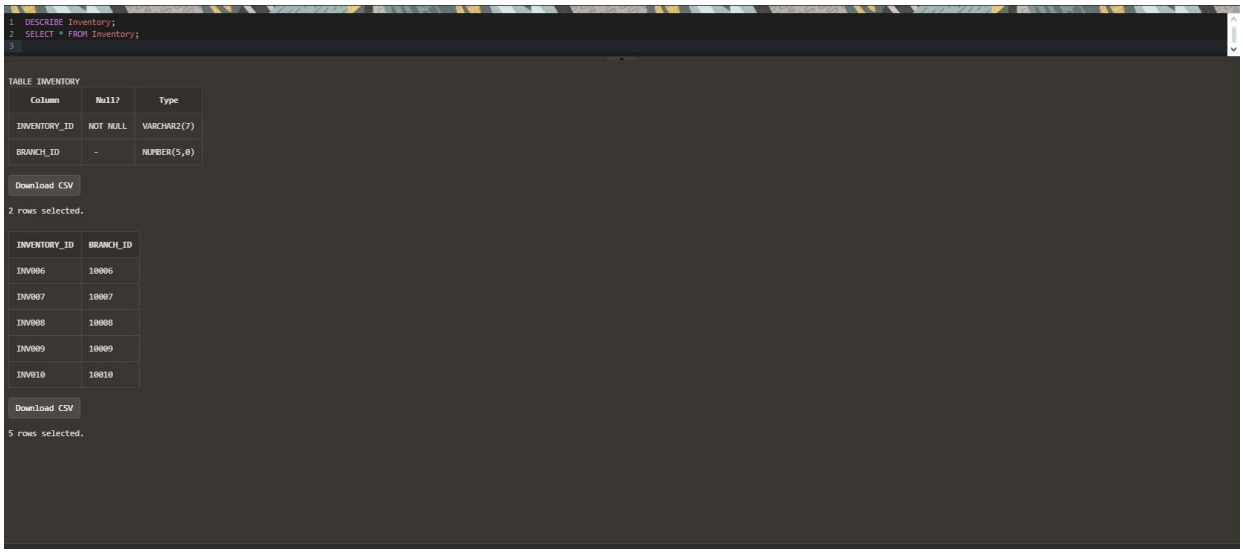
5 rows selected.

This table structure, named 'Inventory', represents the inventory management system within your database. Each row likely represents a specific item or product stored in a particular branch.

Here's a breakdown of the columns and their constraints:

- **Inventory_id:** This column likely stores a unique identifier for each inventory item (VARCHAR2 data type with a maximum length of 7 characters). The PRIMARY KEY constraint ensures that each record has a distinct identifier within the inventory table.
- **Branch_id:** This column stores a foreign key referencing the 'Branch' table. (Its contains a NUMBER data type with 5 digits) and establishes a link between the branch location and the inventory items stored there. The NOT NULL constraint ensures a branch ID is linked to every inventory record. This relationship helps maintain data integrity by ensuring Branch_id values in the Inventory table correspond to existing branch entries in the 'Branch' table.

the Inventory table have a five rows, each representing an inventory entry with its respective details. This data can be used to track inventory levels at each branch, monitor stock availability, and manage inventory distribution across different branches, providing insights into the company's inventory management practices.



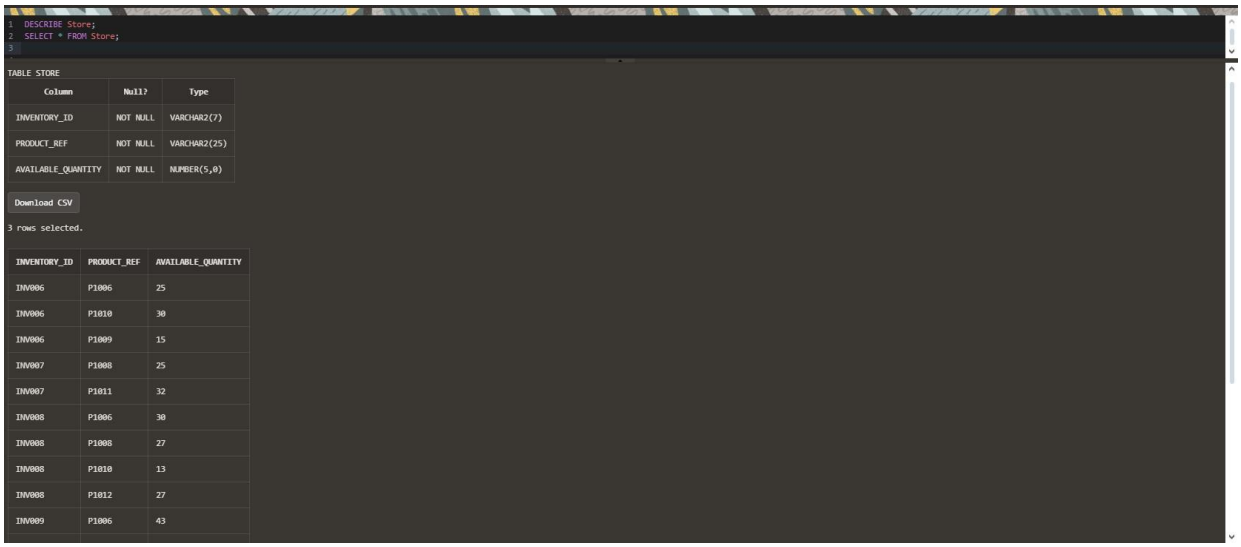
This table structure, named 'Store', represents the product inventory within your database. Each row likely represents the quantity of a specific product available at a particular store location.

Here's a breakdown of the columns and constraints:

- **Inventory_id:** This column likely stores a foreign key referencing the 'Inventory' table. (It's a VARCHAR2 data type with a maximum length of 7 characters). The NOT NULL constraint ensures a valid inventory ID is linked to every record in the Store table. This relationship helps maintain data integrity by referencing existing inventory items from the 'Inventory' table.
- **Product_ref:** This column likely stores a foreign key referencing the 'Product' table. (It's a VARCHAR2 data type with a maximum length of 25 characters). The NOT NULL constraint ensures a valid product reference is linked to every record in the Store table. This relationship helps maintain data integrity by referencing existing products from the 'Product' table.
- **Available_Quantity:** This column stores the number of units currently available for that specific product in the store (a NUMBER data type with 5 digits). The NOT NULL constraint ensures a quantity is recorded for every inventory record.

Primary Key: **(Inventory_id, Product_ref):** The table defines a composite primary key consisting of both Inventory_id and Product_ref columns. This means a combination of a specific inventory ID and a specific product reference must be unique to identify each record in the Store table. This composite approach ensures a single product within a particular inventory location cannot have duplicate entries.

the Store table with data indicating the available quantity of each product in different inventories. This data can be used to manage inventory levels, track product availability across stores, and optimize stock replenishment processes, providing insights into the company's inventory management practices and store operations.



```
1  DESCRIBE Store;
2  SELECT * FROM Store;
3
```

TABLE STORE

Column	Null?	Type
INVENTORY_ID	NOT NULL	VARCHAR2(7)
PRODUCT_REF	NOT NULL	VARCHAR2(25)
AVAILABLE_QUANTITY	NOT NULL	NUMBER(5,0)

Download CSV

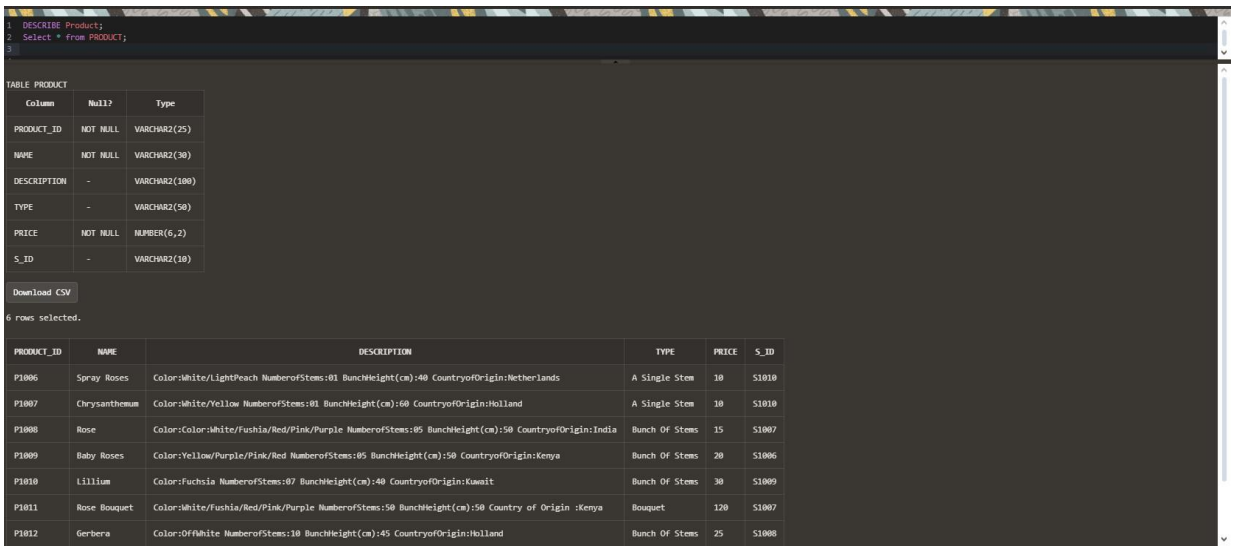
3 rows selected.

INVENTORY_ID	PRODUCT_REF	AVAILABLE_QUANTITY
INV006	P1006	25
INV006	P1010	30
INV006	P1009	15
INV007	P1008	25
INV007	P1011	32
INV008	P1006	30
INV008	P1008	27
INV008	P1010	13
INV008	P1012	27
INV009	P1006	43

This image show a table structure, named 'Product', represents the product information within your database. Each row likely represents a single product offered by your company. Here's a breakdown of the columns and their constraints:

- **Product_id:** This column stores a unique identifier for each product (its a VARCHAR2 data type with a maximum length of 25 characters). The PRIMARY KEY constraint ensures that each product record has a distinct identifier within the table.
- **Name:** This column stores the name of the product (its a VARCHAR2 data type with a maximum length of 30 characters). The NOT NULL constraint ensures a product name is entered for every record.
- **Description:** This column stores a description of the product (its a VARCHAR2 data type with a maximum length of 100 characters). This column might allow null values (empty descriptions) depending on your data needs.
- **Type:** This column stores the product type (its a VARCHAR2 data type with a maximum length of 50 characters). This column might allow null values (empty types) depending on your data needs.
- **Price:** This column stores the price of the product (its a NUMBER data type with 6 digits and 2 decimal places). The NOT NULL constraint ensures a price is entered for every product record.
- **S_id:** This column stores a foreign key referencing the 'Supplier' table. (It's a contains a VARCHAR2 data type with a maximum length of 10 characters) and establishes a link between the product and the supplier who provides it. The NOT NULL constraint ensures a supplier ID is linked to every product record. This relationship helps maintain data integrity by ensuring S_id values in the Product table correspond to existing supplier entries in the 'Supplier' table.

The Product table populated with seven rows, each representing a product with its respective details. This data can be used to track product information, manage inventory, analyze product pricing, and monitor supplier relationships, providing insights into the company's product offerings and sourcing practices.



```
1  DESCRIBE Product;
2  Select * from PRODUCT;
3
```

TABLE PRODUCT

Column	Null?	Type
PRODUCT_ID	NOT NULL	VARCHAR2(25)
NAME	NOT NULL	VARCHAR2(30)
DESCRIPTION	-	VARCHAR2(100)
TYPE	-	VARCHAR2(50)
PRICE	NOT NULL	NUMBER(6,2)
S_ID	-	VARCHAR2(10)

Download CSV

6 rows selected.

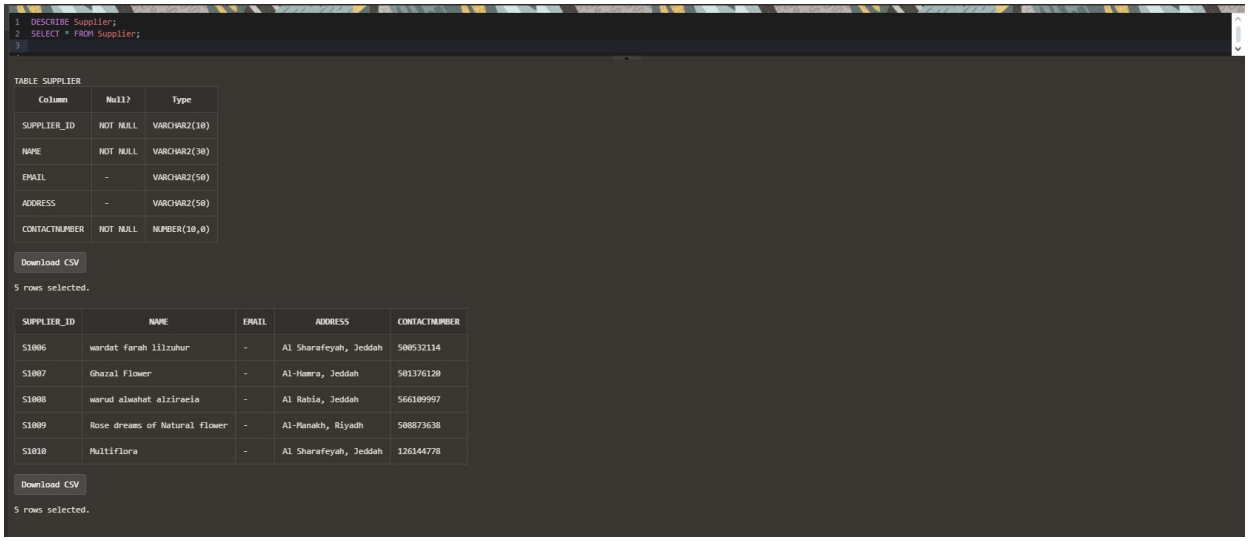
PRODUCT_ID	NAME	DESCRIPTION	TYPE	PRICE	S_ID
P1006	Spray Roses	Color:White/LightPeach NumberofStems:01 BunchHeight(cm):40 CountryofOrigin:Netherlands	A Single Stem	10	S1010
P1007	Chrysanthemum	Color:White/Yellow NumberofStems:01 BunchHeight(cm):60 CountryofOrigin:Holland	A Single Stem	10	S1010
P1008	Rose	Color:Color:White/Fushia/Red/Pink/Purple NumberofStems:05 BunchHeight(cm):50 CountryofOrigin:India	Bunch Of Stems	15	S1007
P1009	Baby Roses	Color:Yellow/Purple/Pink/Red NumberofStems:05 BunchHeight(cm):50 CountryofOrigin:Kenya	Bunch Of Stems	20	S1006
P1010	Lillium	Color:Fuchsia NumberofStems:07 BunchHeight(cm):40 CountryofOrigin:Kuwait	Bunch Of Stems	30	S1009
P1011	Rose Bouquet	Color:White/Fushia/Red/Pink/Purple NumberofStems:50 BunchHeight(cm):50 Country of Origin :Kenya	Bouquet	120	S1007
P1012	Gerbera	Color:OffWhite NumberofStems:10 BunchHeight(cm):45 CountryofOrigin:Holland	Bunch Of Stems	25	S1008

This table structure, named 'Supplier', represents the supplier information within your database. Each row likely represents a company or vendor from which you purchase products.

Here's a breakdown of the columns and their constraints:

- **Supplier_id:** This column stores a unique identifier for each supplier (its a VARCHAR2 data type with a maximum length of 10 characters). The PRIMARY KEY constraint ensures that each supplier record has a distinct identifier within the table.
- **Name:** This column stores the name of the supplier company (its a VARCHAR2 data type with a maximum length of 30 characters). The NOT NULL constraint ensures a supplier name is entered for every record.
- **Email:** This column stores the email address of the supplier (its a VARCHAR2 data type with a maximum length of 50 characters). This column might allow null values (empty emails) depending on your data needs.
- **Address:** This column stores the address of the supplier (its a VARCHAR2 data type with a maximum length of 50 characters.
- **ContactNumber:** This column stores the contact phone number for the supplier (likely a NUMBER data type with 10 digits). The NOT NULL constraint ensures a contact number is entered for every supplier record. The UNIQUE constraint ensures that no two suppliers share the same contact number.

the Supplier table with data representing different suppliers along with their contact information. This data can be used to manage supplier relationships, track supplier details, and facilitate communication with suppliers, providing insights into the company's supplier network and procurement processes.



```
1  DESCRIBE Supplier;
2  SELECT * FROM Supplier;
3
```

TABLE SUPPLIER

Column	Null?	Type
SUPPLIER_ID	NOT NULL	VARCHAR2(10)
NAME	NOT NULL	VARCHAR2(30)
EMAIL	-	VARCHAR2(50)
ADDRESS	-	VARCHAR2(50)
CONTACTNUMBER	NOT NULL	NUMBER(10,0)

Download CSV

5 rows selected.

SUPPLIER_ID	NAME	EMAIL	ADDRESS	CONTACTNUMBER
S1006	wardat farah lilzuhur	-	Al Sharafeyah, Jeddah	500532114
S1007	Ghazal Flower	-	Al-Hamra, Jeddah	501376120
S1008	warud alwahat alziraeia	-	Al Rabia, Jeddah	566109997
S1009	Rose dreams of Natural flower	-	Al-Manakh, Riyadh	508873638
S1010	Multiflora	-	Al Sharafeyah, Jeddah	126144778

Download CSV

5 rows selected.

Task2 : relational algebra and SQL statements

Building complex queries, i.e. using the **join** condition to obtain data from multiple tables. (At least 3 queries)

Query 1: Products and their inventory in a specific branch

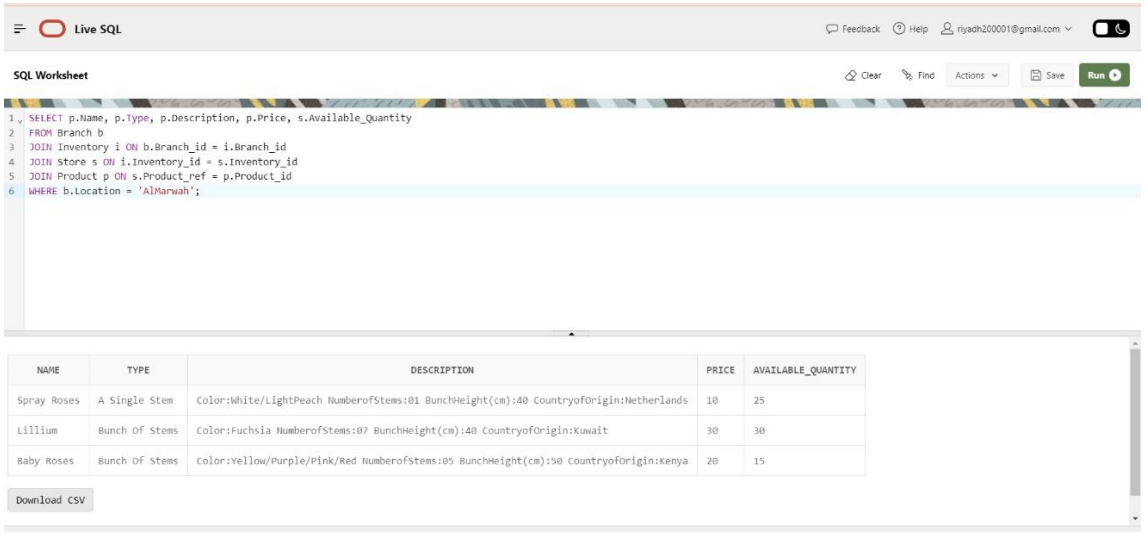
Definition:

This query retrieves a list of all products available at the 'AlMarwah' branch, including their names, types, descriptions, prices, and available quantities in the inventory. It combines information from several tables: Branch, Inventory, Store, and Product, focusing specifically on a given branch location.

Relational algebra:

$$\pi Name,\ Type,\ Description,\ Price,\ Available\_Quantity(\sigma Location = \text{'AlMarwah'}(Branch \bowtie Branch\_id = Branch\_id Inventory \bowtie Inventory\_id = Inventory\_id Store \bowtie Product\_ref = Product\_id Product))$$

SQL query:



Query 2: orders with customers and employee details

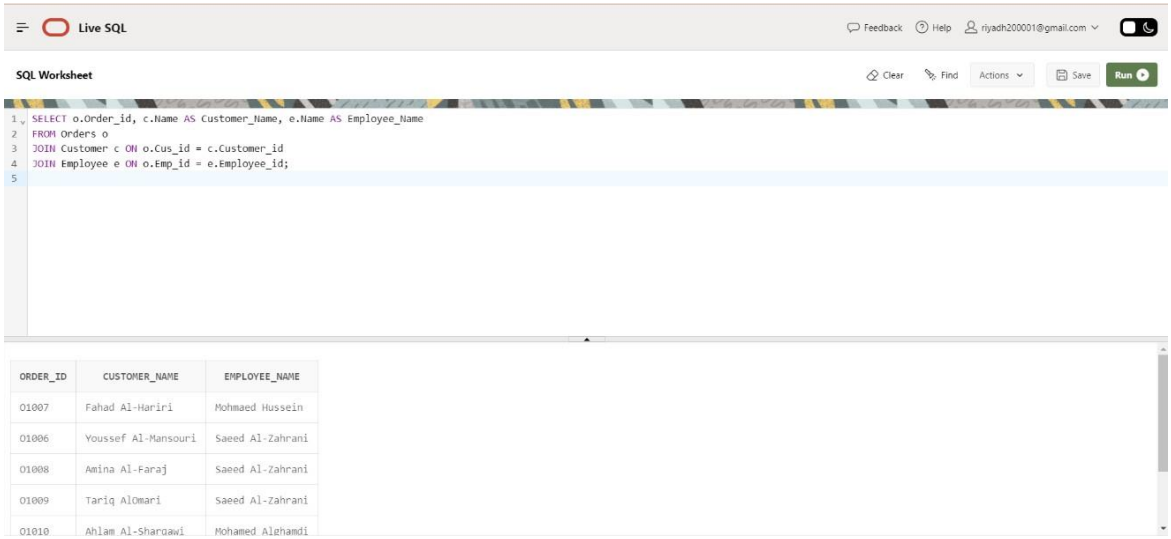
Definition:

This query fetches details of each order, including the order ID, the name of the customer who placed the order, and the name of the employee who handled the order. It links Orders, Customer, and Employee tables to provide comprehensive details about who placed each order and who managed it

Relational algebra:

πOrder_id, Customer_Name, Employee_Name(Orders⋈Cus_id = Customer_idCustomer⋈Emp_id = Employee_idEmployee)

SQL query:



.

Query 3: customer loyalty program details

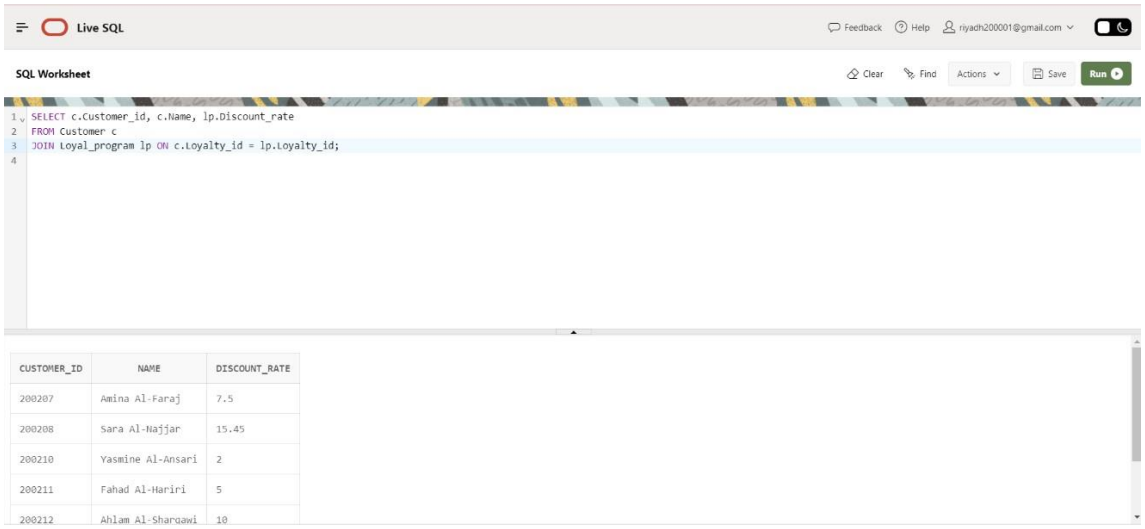
Definition:

Shows customer IDs, names, and their associated discount rates from the loyalty program. Useful for understanding how loyalty benefits are distributed among customers.

Relational Algebra:

πCustomer_id, Name, Discount_rate(Customer⋈Loyalty_id = Loyalty_idLoyal_program)πCustomer_id, Name, Discount_rate(Customer⋈Loyalty_id = Loyalty_idLoyal_program)

SQL statement:



Query 4: Supplier Products and Contact Details

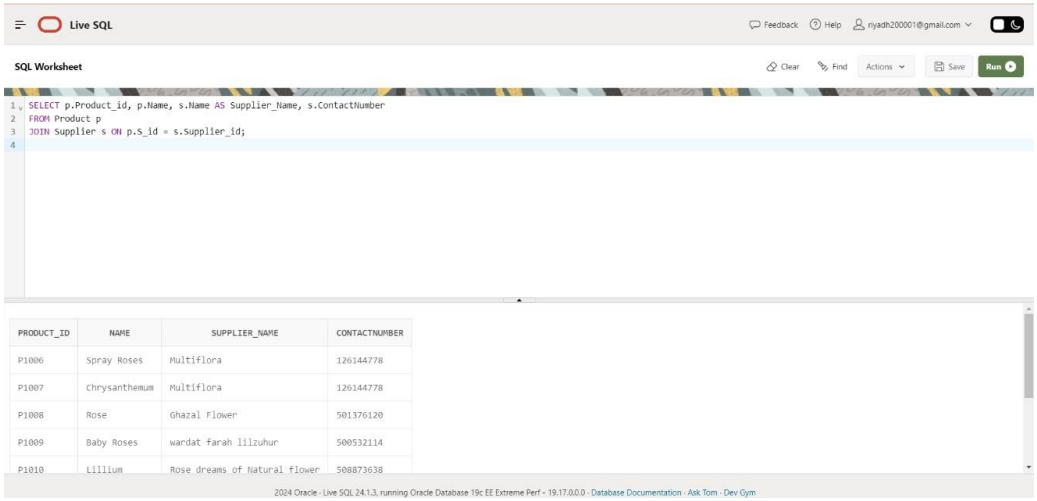
Definition:

Lists all products with their supplier names and contact details. Facilitates easy communication with suppliers and tracks product sourcing.

Relational Algebra:

πProduct_id, Name, Supplier_Name, ContactNumber(Product⋈S_id = Supplier_idSupplier)π Product_id, Name, Supplier_Name, ContactNumber(Product⋈S_id = Supplier_idSupplier)

SQL statement:



Query 5 : Employees and their branch details

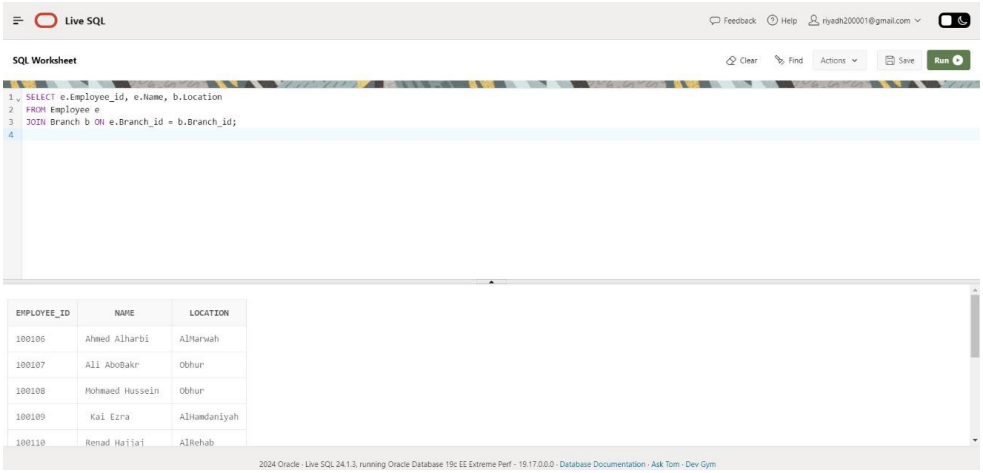
Definition:

Provides a list of all employees along with the locations of the branches they work at. Essential for managing staffing across various locations

Relational Algebra:

πEmployee_id, Name, Location(Employee⋈Employee.Branch_id=Branch.Branch_idBranch)

SQL statement:



Building queries which include aggregate functions. (At least 3 queries)

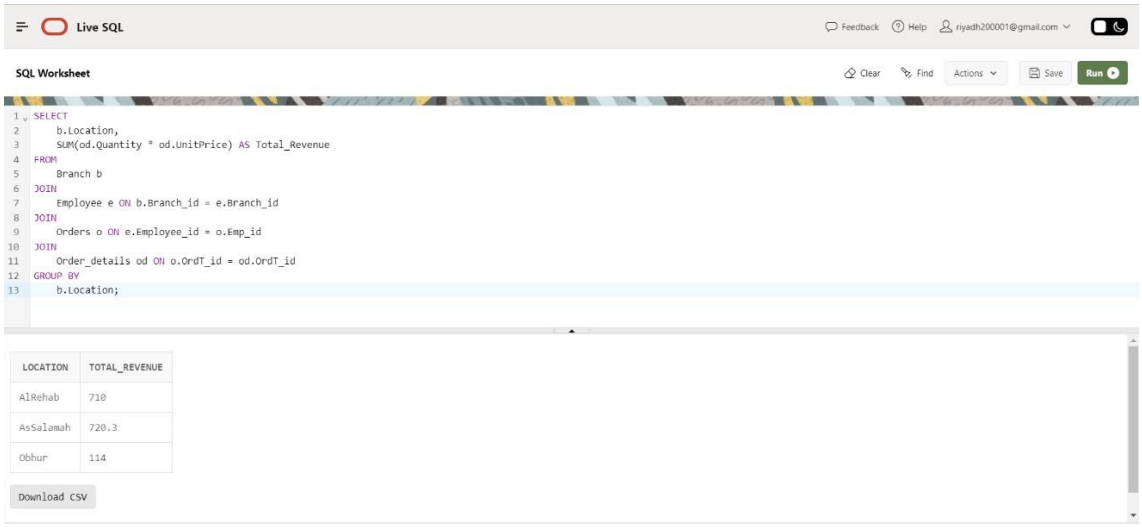
Query 1: Supplier Products and Contact Details

Definition :

this SQL statement retrieves the total revenue generated by each branch by joining multiple tables (Branch, Employee, Orders, and Order_details), aggregating the revenue for each branch, and grouping the result by branch location.

Relational Algebra:

γLocation, SUM(od.Quantity×od.UnitPrice)→Total_Revenue(πLocation, Total_Revenue (Branch b⋈b.Branch_id=e.Branch_id Employee e⋈e.Employee_id=o.Emp_id Orders o⋈o.OrdT_id=od.OrdT_id Order_details od))SQL statement:



Query 2: Total Salary Expense Per Branch

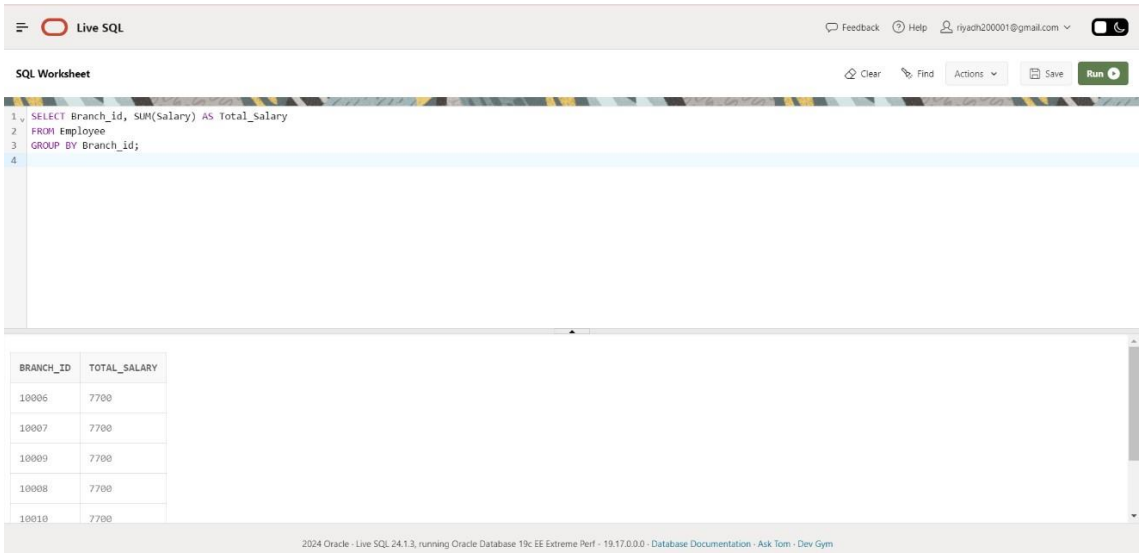
Definition:

This query aggregates salaries within each branch, summarizing total salary expenditure to aid in financial planning and budget allocations.

Relational Algebra:

GBranch_id(SUM($Salary$))(Employee)GBranch_id(SUM(Salary))(Employee)

SQL statement:



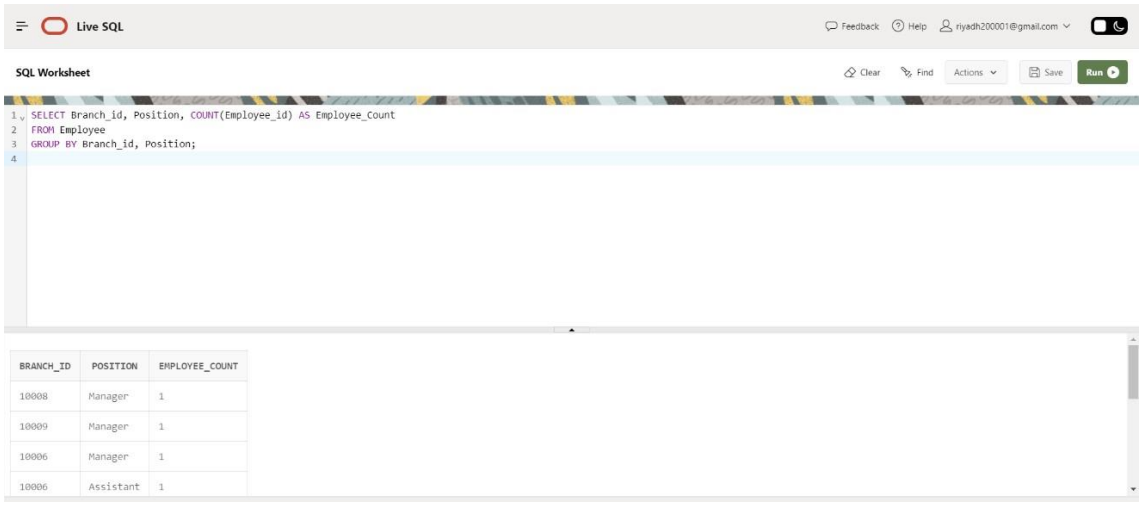
Query 3: Number of Employees Per Position at Each Branch

Definition:

This query highlights the distribution of employees by position at each branch, assisting HR in managing staff levels and training needs.

Relational Algebra: G_{Branch_id, Position} (COUNT(Employee_id))(Employee)

SQL statement:



Query 4: Average Salary of Employees by Position

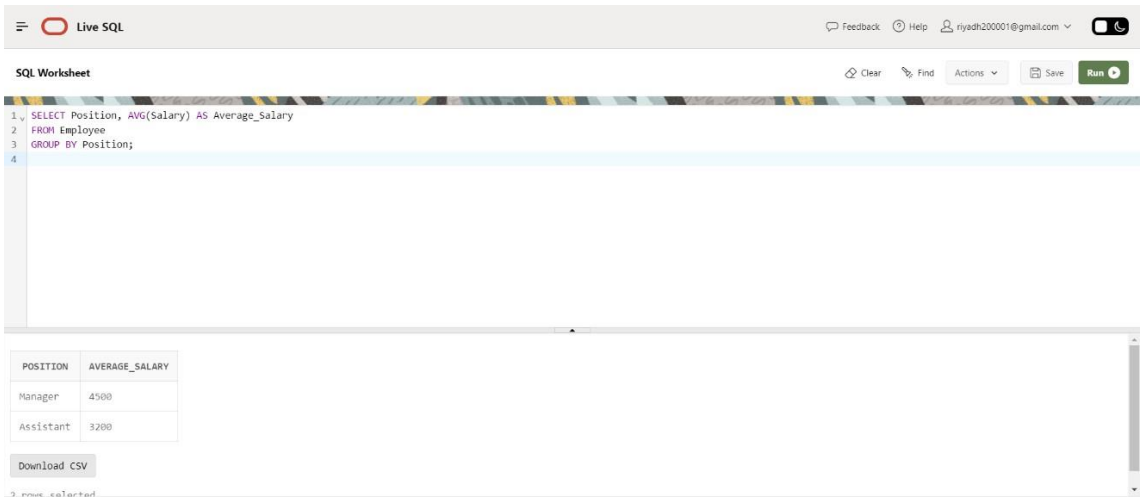
Definition:

This query provides benchmarks for average salaries by position, useful for payroll reviews and ensuring competitive compensation.

Relational Algebra:

G_{Position} (AVG(Salary))(Employee)

SQL statement:



Query 5: Maximum and Minimum Unit Price of Products in Each Product Type

Definition:

This query assesses the price range within each product type, offering insights for marketing strategies and inventory management.

Relational Algebra:

G_{Type} (MAX(Price), MIN(Price))(Product)

SQL statement: