Database Project

Reuven Chiche 328944517

&

Eyal Seckbach 324863539

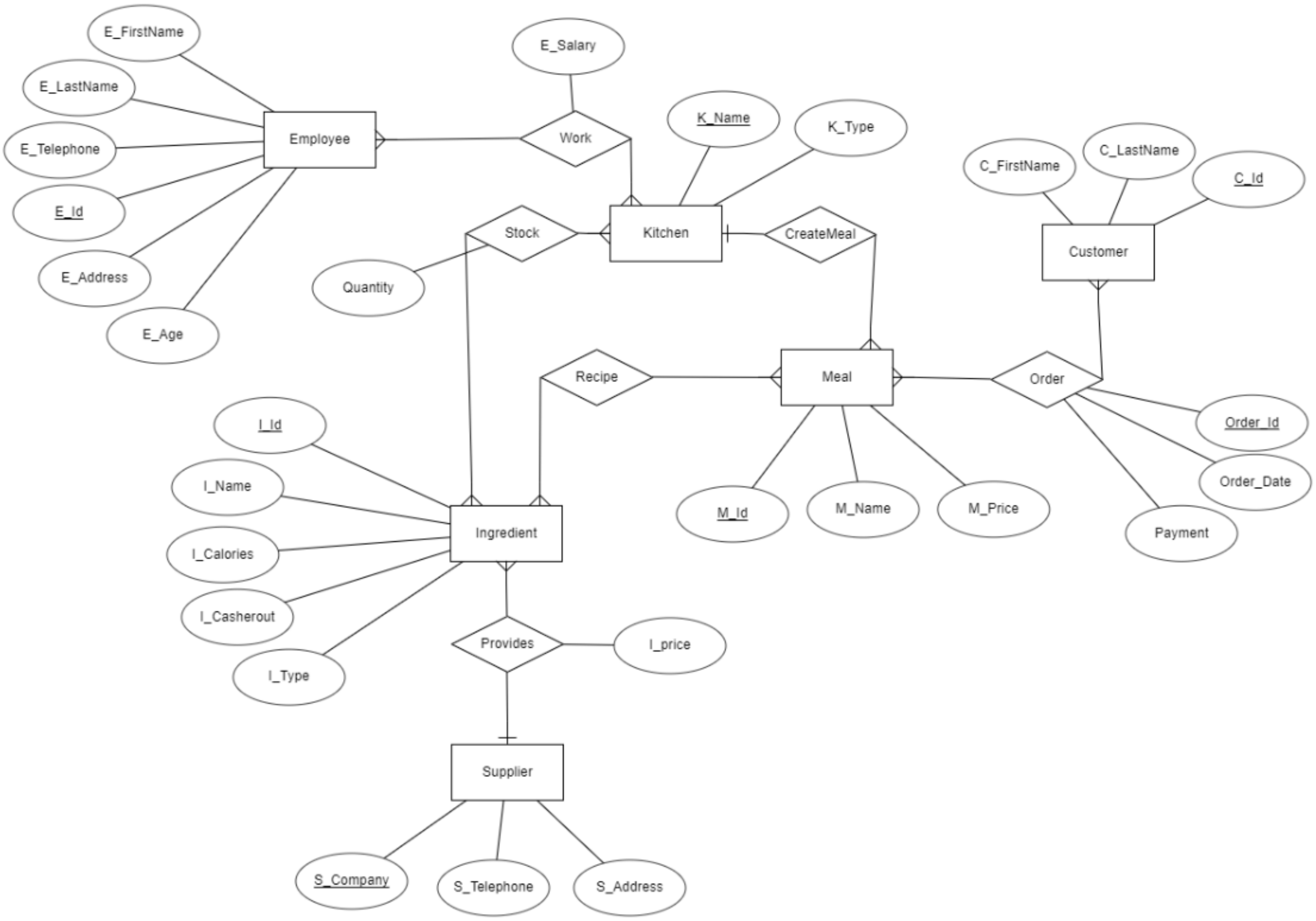
Table of Contents

Description of the organization

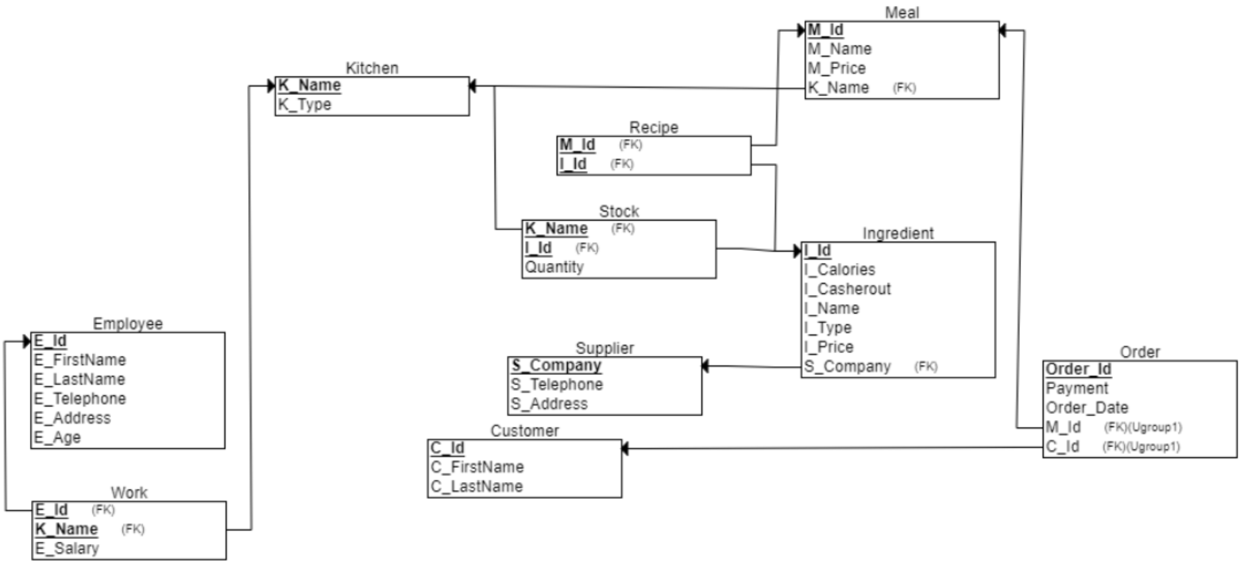
Our cafeteria database project is being developed to streamline the operations of the cafeteria. The database will include information about the inventory of ingredients, suppliers, and their prices. In addition, the database will store employees information, such as their names, address, telephone and age. and also customers' information such as their orders and payment methods. The aim of the project is to improve efficiency and provide a better customer experience by ensuring that meals are prepared in a timely manner and that the cafeteria is well-stocked with fresh ingredients. Once implemented, the cafeteria database project will be an essential tool for managing the day-to-day operations of the cafeteria.

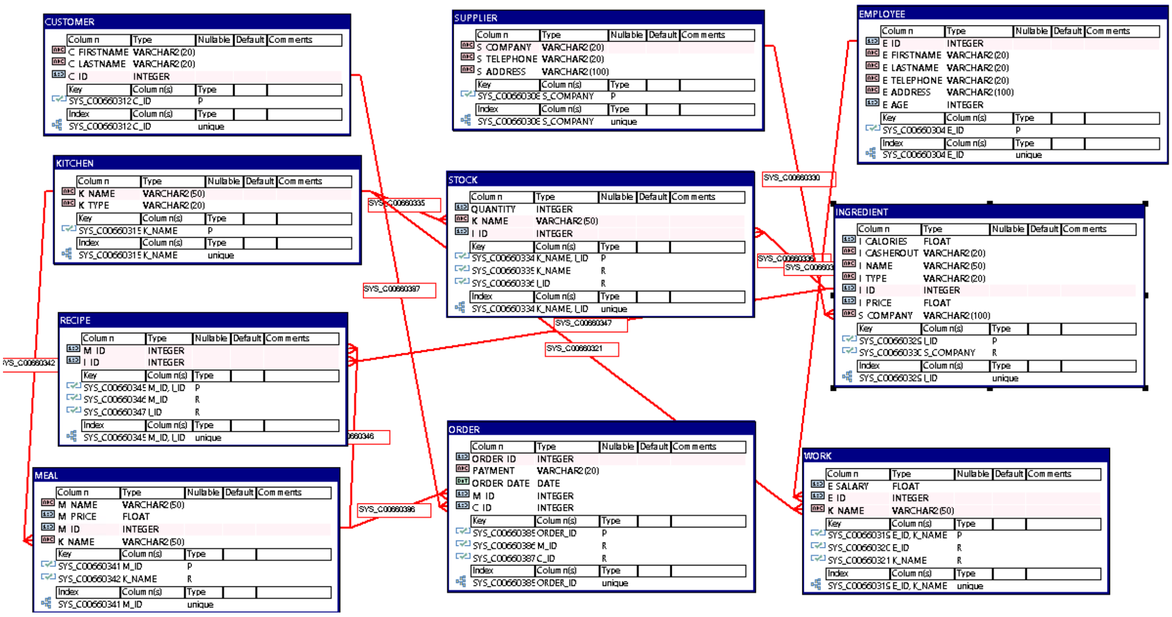
This cafeteria is a place where customers can come and order meals prepared by the kitchen staff. The kitchen serves a variety of cuisine types and prepares meals using ingredients provided by different supplier companies. Customers can pay for their orders using different payment methods. The cafeteria keeps track of the stock of ingredients and ensures that there is enough quantity available to prepare the meals. The employees working in the kitchen are paid salaries. The cafeteria maintains a customer database and keeps track of the orders made by each customer. Each meal has a unique identifier and the recipe for each meal is recorded, specifying the ingredients used.

ERD chart


E_FirstName
E_LastName
E_Telephone
E_Id
E_Address
E_Age
Employee
E_Salary
Work
K_Name
K_Type
Kitchen
Stock
Quantity
CreateMeal
C_FirstName
C_LastName
C_Id
Customer
Recipe
Meal
Order
Order_Id
Order_Date
Payment
M_Id
M_Name
M_Price
I_Id
I_Name
I_Calories
I_Casherout
I_Type
Ingredient
Provides
I_price
Supplier
S_Company
S_Telephone
S_Address

DSD chart





CUSTOMER

Column	Type	Nullable	Default	Comments
C FIRSTNAME	VARCHAR2 (20)			
C LASTNAME	VARCHAR2 (20)			
C ID	INTEGER			

Key	Column(s)	Type		
SYS_C00660312	C_ID	P		

Index	Column(s)	Type		
SYS_C00660312	C_ID	unique		

SUPPLIER

Column	Type	Nullable	Default	Comments
S COMPANY	VARCHAR2 (20)			
S TELEPHONE	VARCHAR2 (20)			
S ADDRESS	VARCHAR2 (100)			

Key	Column(s)	Type		
SYS_C0066030E	S_COMPANY	P		

Index	Column(s)	Type		
SYS_C0066030E	S_COMPANY	unique		

EMPLOYEE

Column	Type	Nullable	Default	Comments
E ID	INTEGER			
E FIRSTNAME	VARCHAR2 (20)			
E LASTNAME	VARCHAR2 (20)			
E TELEPHONE	VARCHAR2 (20)			
E ADDRESS	VARCHAR2 (100)			
E AGE	INTEGER			

Key	Column(s)	Type		
SYS_C00660304	E_ID	P		

Index	Column(s)	Type		
SYS_C00660304	E_ID	unique		

KITCHEN

Column	Type	Nullable	Default	Comments
K NAME	VARCHAR2 (50)			
K TYPE	VARCHAR2 (20)			

Key	Column(s)	Type		
SYS_C0066031E	K_NAME	P		

Index	Column(s)	Type		
SYS_C0066031E	K_NAME	unique		

STOCK

Column	Type	Nullable	Default	Comments
QUANTITY	INTEGER			
K NAME	VARCHAR2 (50)			
I ID	INTEGER			

Key	Column(s)	Type		
SYS_C00660334	K_NAME, I_ID	P		
SYS_C00660335	K_NAME	R		
SYS_C0066033E	I_ID	R		

Index	Column(s)	Type		
SYS_C00660334	K_NAME, I_ID	unique		

INGREDIENT

Column	Type	Nullable	Default	Comments
I CALORIES	FLOAT			
I CASHEROUT	VARCHAR2 (20)			
I NAME	VARCHAR2 (50)			
I TYPE	VARCHAR2 (20)			
I ID	INTEGER			
I PRICE	FLOAT			
S COMPANY	VARCHAR2 (100)			

Key	Column(s)	Type		
SYS_C0066032S	I_ID	P		
SYS_C0066033C	S_COMPANY	R		

Index	Column(s)	Type		
SYS_C0066032S	I_ID	unique		

RECIPE

Column	Type	Nullable	Default	Comments
M ID	INTEGER			
I ID	INTEGER			

Key	Column(s)	Type		
SYS_C0066034E	M_ID, I_ID	P		
SYS_C0066034E	M_ID	R		
SYS_C00660347	I_ID	R		

Index	Column(s)	Type		
SYS_C0066034E	M_ID, I_ID	unique		

MEAL

Column	Type	Nullable	Default	Comments
M NAME	VARCHAR2 (50)			
M PRICE	FLOAT			
M ID	INTEGER			
K NAME	VARCHAR2 (50)			

Key	Column(s)	Type		
SYS_C00660341	M_ID	P		
SYS_C00660342	K_NAME	R		

Index	Column(s)	Type		
SYS_C00660341	M_ID	unique		

ORDER

Column	Type	Nullable	Default	Comments
ORDER ID	INTEGER			
PAYMENT	VARCHAR2 (20)			
ORDER DATE	DATE			
M ID	INTEGER			
C ID	INTEGER			

Key	Column(s)	Type		
SYS_C0066038E	ORDER_ID	P		
SYS_C0066038E	M_ID	R		
SYS_C00660387	C_ID	R		

Index	Column(s)	Type		
SYS_C0066038E	ORDER_ID	unique		

WORK

Column	Type	Nullable	Default	Comments
E SALARY	FLOAT			
E ID	INTEGER			
K NAME	VARCHAR2 (50)			

Key	Column(s)	Type		
SYS_C0066031S	E_ID, K_NAME	P		
SYS_C0066032C	E_ID	R		
SYS_C00660321	K_NAME	R		

Index	Column(s)	Type		
SYS_C0066031S	E_ID, K_NAME	unique		

Description of the entities:

Employee

The "Employee" entity represents the employees of the cafeteria who can work in the cafeteria. The purpose of this entity is to store information about employees in the cafeteria. By storing information about each employee, such as their unique identifier, first and last names, telephone number, address, and age.

E_Id (INT, NOT NULL)	**This is the unique identifier for each employee**
E_FirstName **(VARCHAR, NOT NULL)**	**This is the first name of the employee**
E_LastName **(VARCHAR, NOT NULL)**	**This is the last name of the employee**
E_Telephone **(VARCHAR, NOT NULL)**	**This is the telephone number of the employee**
E_Address **(VARCHAR, NOT NULL)**	**This is the address of the employee**
E_Age **(INT, NOT NULL)**	**This is the age of the employee**

Supplier

The "Supplier" entity represents the suppliers who can supply ingredients for the kitchens. The purpose of this entity is to store information such as the supplier's name, telephone number, and address so that the cafeteria can manage and track the different suppliers that can supply the kitchens.

S_Company **(VARCHAR, NOT NULL)**	**This is the name of the supplier company**
S_Telephone **(VARCHAR, NOT NULL)**	**This is the telephone number of the supplier company**
S_Address **(VARCHAR, NOT NULL)**	**This is the address of the supplier company**

Customer

The "Customer" entity represents the people who place orders for food. The purpose of this entity is to store information about the customers such as their first and last name, and a unique identifier for each customer.

C_FirstName **(VARCHAR, NOT NULL)**	**This is the first name of the customer**
C_LastName **(VARCHAR, NOT NULL)**	**This is the last name of the customer**
C_Id **(INT, NOT NULL)**	**This is the unique identifier for each customer**

Kitchen

The "Kitchen" entity represents the different kitchens in the cafeteria that prepare meals. The purpose of this entity is to store information about each kitchen, such as its name and the type of cuisine it serves, so that the cafeteria can manage and track the different types of meals it offers.

K_Name **(VARCHAR, NOT NULL)**	**This is the name of the kitchen**
K_Type **(VARCHAR, NOT NULL)**	**This is the type of cuisine served by the kitchen**

Work

The "Work" entity represents the employees who work in the different kitchens in the restaurant. The purpose of this entity is to store information about the employees' salaries, the unique identifier for each employee, and the name of the kitchen where they work, so that the cafeteria can manage employee information and track which employees work in which kitchens.

E_Salary **(FLOAT, NOT NULL)**	**This is the salary of the employee**
E_Id **(INT, NOT NULL)**	**This is the unique identifier for each employee**
K_Name **(VARCHAR, NOT NULL)**	**This is the name of the kitchen where the employee works**

Ingredient

The "Ingredient" entity represents the different ingredients used to prepare the meals. The purpose of this entity is to store information about each ingredient, such as its unique identifier, name, type, calories, casherout certification, price, and the supplier company that provides the ingredient, so that the cafeteria can manage its inventory and track where ingredients come from.

I_Id (INT, NOT NULL)	**This is the unique identifier for each ingredient**
I_Calories (FLOAT, NOT NULL)	**This is the number of calories in the ingredient**
I_Casherout (VARCHAR, NOT NULL)	**This is the casherout certification of the ingredient**
I_Name (VARCHAR, NOT NULL)	**This is the name of the ingredient**
I_Type (VARCHAR, NOT NULL)	**This is the type of the ingredient**
I_Price (FLOAT, NOT NULL)	**This is the price of the ingredient**
S_Company (VARCHAR, NOT NULL)	**This is the name of the supplier company that provides the ingredient**

Meal

The "Meal" entity represents the different meals that are served in the cafeteria. The purpose of this entity is to store information about each meal, such as its unique identifier, name, price, and the kitchen where it is prepared so that the cafeteria can manage its menu.

M_Name **(VARCHAR, NOT NULL)**	**This is the name of the meal**
M_Price **(FLOAT, NOT NULL)**	**This is the price of the meal**
M_Id **(INT, NOT NULL)**	**This is the unique identifier for each meal**
K_Name **(VARCHAR, NOT NULL)**	**The name of the kitchen where the meal is prepared**

Order

The "Order" entity represents the orders made by customers for meals. The purpose of this entity is to store information about each order, such as the unique identifier for each order, the payment type for the order, the date the order was made, the meal that was ordered, and the customer who made the order, so that the cafeteria can manage its orders.

Order_Id (INT, NOT NULL)	**The unique identifier for each order**
Payment (VARCHAR, NOT NULL)	**The payment type for the order**
Order_Date (DATE, NOT NULL)	**The date on which the order was made**
M_Id (INT, NOT NULL)	**The unique identifier for the meal that was ordered**
C_Id (INT, NOT NULL)	**The unique identifier for the customer who made the order**

Stock

The "Stock" entity represents the stock of ingredients available in the restaurant's different kitchens. The purpose of this entity is to store information about the quantity of each ingredient in stock, the name of the kitchen where the ingredient is used, and the unique identifier for each ingredient so that the cafeteria can manage its inventory.

Quantity (INT, NOT NULL)	**The quantity of the ingredient in stock**
K_Name (VARCHAR, NOT NULL)	**The name of the kitchen where the ingredient is used**
I_Id (INT, NOT NULL)	**The unique identifier for the ingredient**

Recipe

The "Recipe" entity represents the ingredients used to prepare each meal. The purpose of this entity is to store information about the unique identifier for each meal and the unique identifier for each ingredient used in the recipe so that the cafeteria can manage its recipes.

M_Id (INT, NOT NULL)	**The unique identifier for the meal**
I_Id (INT, NOT NULL)	**The unique identifier for the ingredient used in the recipe**

Description of the relationships

Create meal

"Create meal" is a many-to-one relationship that signifies that each meal was created by only one kitchen. This relationship helps us keep track of which kitchen was responsible for preparing a particular meal.

Provide

The "Provide" relationship is a many-to-one relationship that tells us that each ingredient is supplied by only one supplier to the kitchens. This relationship is essential in maintaining a record of the suppliers and their provided ingredients for the cafeteria.

Work, Stock, Recipe, Order

These relationships are many-to-many relationships, and have been explained above.

Scripts to create tables

Create

Creating all the entities.

```
CREATE TABLE Employee
(
   E_Id         INT     NOT NULL,
   E_FirstName VARCHAR NOT NULL,
   E_LastName  VARCHAR NOT NULL,
   E_Telephone VARCHAR NOT NULL,
   E_Address   VARCHAR NOT NULL,
   E_Age        INT     NOT NULL,
   PRIMARY KEY (E_Id)
);
```

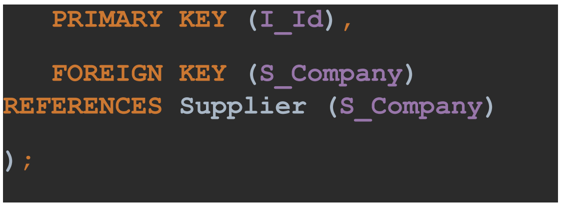
```
CREATE TABLE Supplier
(
   S_Company   VARCHAR NOT NULL,
   S_Telephone VARCHAR NOT NULL,
   S_Address   VARCHAR NOT NULL,
   PRIMARY KEY (S_Company)
);
```

```
CREATE TABLE Customer
(
   C_FirstName VARCHAR NOT NULL,
   C_LastName  VARCHAR NOT NULL,
   C_Id         INT     NOT NULL,
   PRIMARY KEY (C_Id)
);
```

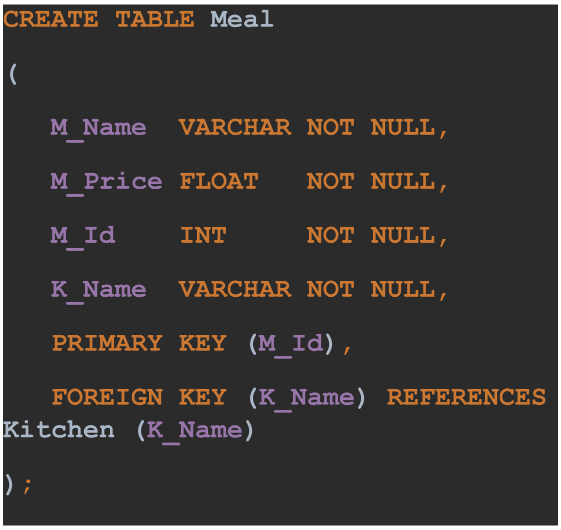
```
CREATE TABLE Kitchen
(
   K_Name VARCHAR NOT NULL,
   K_Type VARCHAR NOT NULL,
   PRIMARY KEY (K_Name)
);
```

```
CREATE TABLE Work
(
   E_Salary FLOAT   NOT NULL,
   E_Id     INT     NOT NULL,
   K_Name   VARCHAR NOT NULL,
   PRIMARY KEY (E_Id, K_Name),
   FOREIGN KEY (E_Id) REFERENCES
Employee (E_Id),
   FOREIGN KEY (K_Name) REFERENCES
Kitchen (K_Name)
);
```

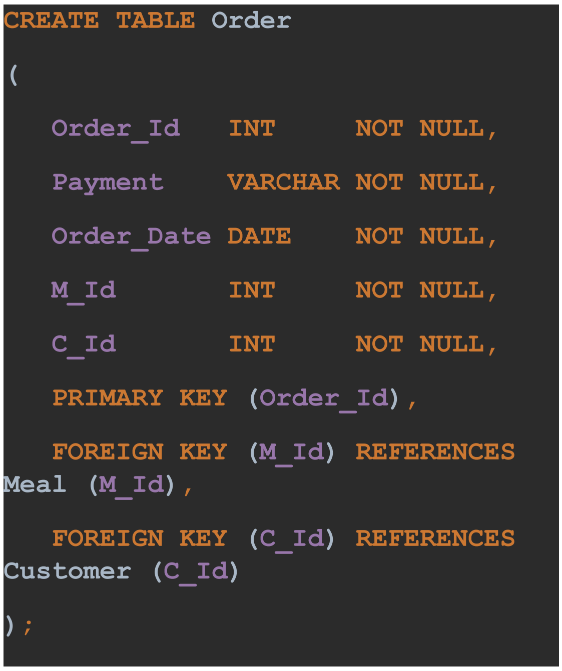
```
CREATE TABLE Ingredient
(
   I_Calories   FLOAT   NOT NULL,
   I_Casherout VARCHAR NOT NULL,
   I_Name      VARCHAR NOT NULL,
   I_Type      VARCHAR NOT NULL,
   I_Id        INT     NOT NULL,
   I_Price     FLOAT   NOT NULL,
   S_Company   VARCHAR NOT NULL,
```



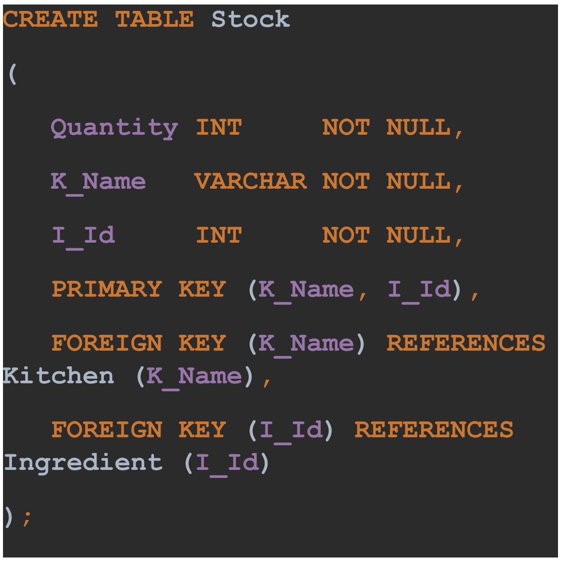
```
   PRIMARY KEY (I_Id),

   FOREIGN KEY (S_Company)
REFERENCES Supplier (S_Company)

);
```



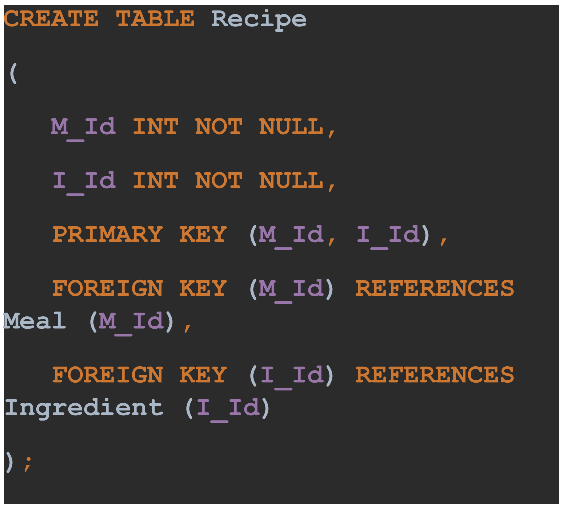
```
CREATE TABLE Meal

(

   M_Name  VARCHAR NOT NULL,

   M_Price FLOAT   NOT NULL,

   M_Id    INT     NOT NULL,

   K_Name  VARCHAR NOT NULL,

   PRIMARY KEY (M_Id),

   FOREIGN KEY (K_Name) REFERENCES
Kitchen (K_Name)

);
```



```
CREATE TABLE Order

(

   Order_Id   INT     NOT NULL,

   Payment    VARCHAR NOT NULL,

   Order_Date DATE    NOT NULL,

   M_Id       INT     NOT NULL,

   C_Id       INT     NOT NULL,

   PRIMARY KEY (Order_Id),

   FOREIGN KEY (M_Id) REFERENCES
Meal (M_Id),

   FOREIGN KEY (C_Id) REFERENCES
Customer (C_Id)

);
```

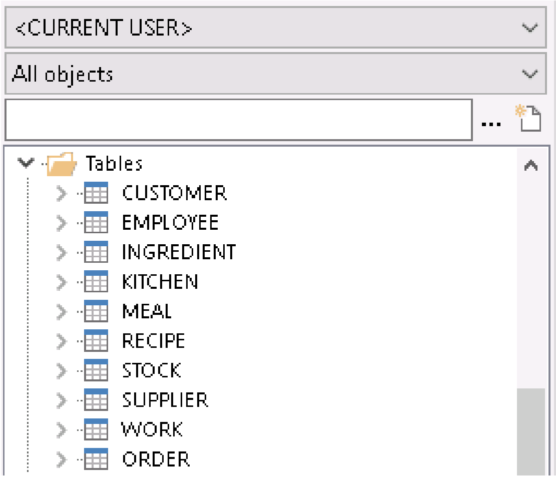


```
CREATE TABLE Stock

(

   Quantity INT     NOT NULL,

   K_Name   VARCHAR NOT NULL,

   I_Id     INT     NOT NULL,

   PRIMARY KEY (K_Name, I_Id),

   FOREIGN KEY (K_Name) REFERENCES
Kitchen (K_Name),

   FOREIGN KEY (I_Id) REFERENCES
Ingredient (I_Id)

);
```



```
CREATE TABLE Recipe

(

   M_Id INT NOT NULL,

   I_Id INT NOT NULL,

   PRIMARY KEY (M_Id, I_Id),

   FOREIGN KEY (M_Id) REFERENCES
Meal (M_Id),

   FOREIGN KEY (I_Id) REFERENCES
Ingredient (I_Id)

);
```

Results



<CURRENT USER>
All objects
...
Tables
CUSTOMER
EMPLOYEE
INGREDIENT
KITCHEN
MEAL
RECIPE
STOCK
SUPPLIER
WORK
ORDER

Basic queries

Drop

1. Drop the "Order" table:

```
DROP TABLE Order;
```

2. Drop the "Work" table:

```
DROP TABLE Work;
```

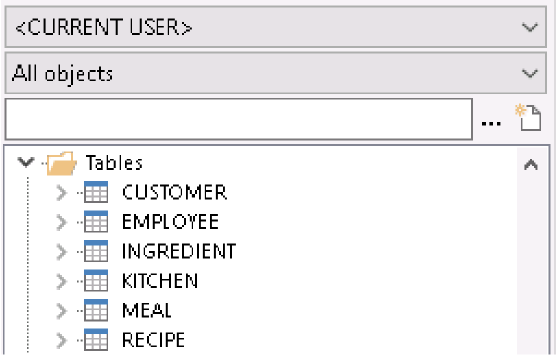
3. Drop the "Kitchen" table:

```
DROP TABLE Supplier;
```

4. Drop the "Employee" table:

```
DROP TABLE Stock;
```

Results



Insert

1. Insert a new customer:

```
INSERT INTO Customer (C_FirstName, C_LastName, C_Id)
VALUES ('John', 'Doe', 15);
```

2. Insert a new meal:

```
INSERT INTO Meal (M_Name, M_Price, M_Id, K_Name)
VALUES ('Chicken Caesar Salad', 14.5, 115, 'Juice Bar');
```

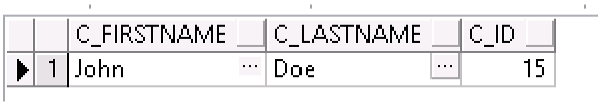
3. Insert a new order:

```
INSERT INTO Order (Order_Id, Payment, Order_Date, M_Id, C_Id)
VALUES (999, 'Cash', TO_DATE('2023-03-28', 'YYYY-MM-DD'), 115, 15);
```

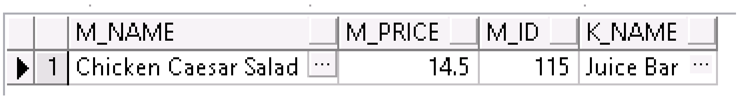
4. Insert a new kitchen:

```
INSERT INTO Kitchen (K_Name, K_Type)
VALUES ('Juice Bar', 'parve');
```

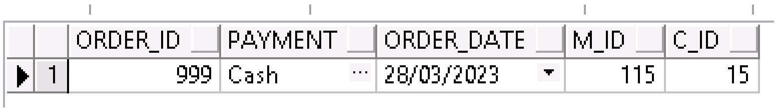
Result



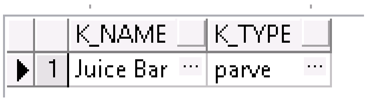
	C_FIRSTNAME	C_LASTNAME	C_ID
1	John	Doe	15



	M_NAME	M_PRICE	M_ID	K_NAME
1	Chicken Caesar Salad	14.5	115	Juice Bar



	ORDER_ID	PAYMENT	ORDER_DATE	M_ID	C_ID
1	999	Cash	28/03/2023	115	15



	K_NAME	K_TYPE
1	Juice Bar	parve

Delete

1. Delete all orders made by a customer with ID 10:

```
DELETE FROM Order
WHERE C_Id = 10;
```

2. Delete all ingredients that have not been used in any meal:

```
DELETE FROM Ingredient
WHERE I_Id NOT IN (
   SELECT I_Id
   FROM Recipe
);
```

3. Delete all meals that have not been ordered at least once:

```
DELETE FROM Meal
WHERE M_Id NOT IN (
   SELECT M_Id
   FROM Order
);
```

4. Delete all employees who do not work:

```
DELETE FROM Employee
WHERE E_Id NOT IN (
   SELECT E_Id
   FROM Work
);
```

Update

1. Update the price of all meals to increase by 10%:

```
UPDATE Meal
SET M_Price = M_Price * 1.1;
```

2. Update the phone number of all employees who are working:

```
UPDATE Employee
SET E_Telephone = '050-1876543'
WHERE E_Id IN (
   SELECT E_Id
   FROM Work
);
```

3. Update the quantity of all ingredients to increase by 50 units:

```
UPDATE Stock

SET Quantity = Quantity + 50;
```

4. Update the salary of all workers to increase by 1000:

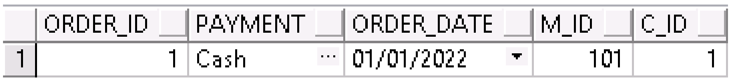
```
UPDATE Work

SET E_Salary = E_Salary + 1000;
```

Select

1. Retrieve all the orders made by customers with ID 101:

```
SELECT *

FROM Order

WHERE C_Id = 1;
```

Result:

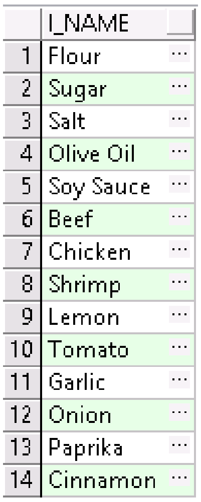


	ORDER_ID	PAYMENT	ORDER_DATE	M_ID	C_ID
1	1	Cash	01/01/2022	101	1

2. Retrieve the names of all the ingredients used in a specific meal:

```
SELECT I_Name

FROM Ingredient NATURAL JOIN Meal

WHERE M_Id = 101;
```

Result:

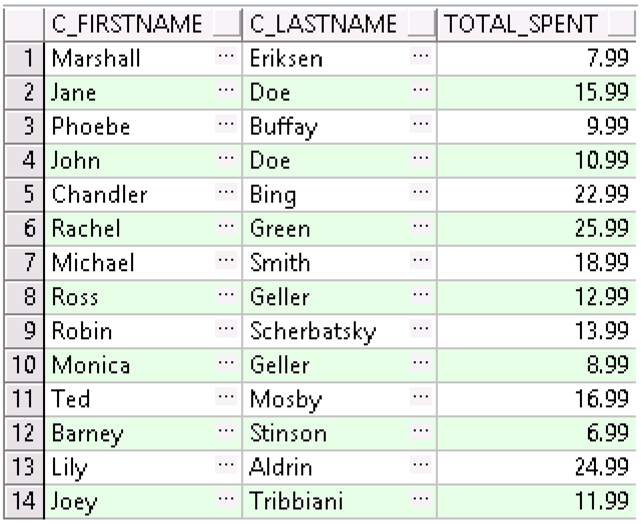


	I_NAME
1	Flour
2	Sugar
3	Salt
4	Olive Oil
5	Soy Sauce
6	Beef
7	Chicken
8	Shrimp
9	Lemon
10	Tomato
11	Garlic
12	Onion
13	Paprika
14	Cinnamon

3. Retrieve the total amount spent by each customer:

```
SELECT C_FirstName, C_LastName, SUM(M_Price) AS Total_Spent

FROM Order NATURAL JOIN Customer NATURAL JOIN Meal

GROUP BY C_FirstName, C_LastName;
```

Result:

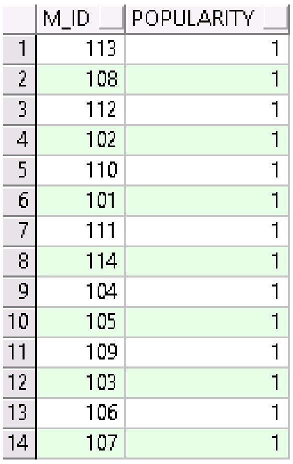


	C_FIRSTNAME	C_LASTNAME	TOTAL_SPENT
1	Marshall	Eriksen	7.99
2	Jane	Doe	15.99
3	Phoebe	Buffay	9.99
4	John	Doe	10.99
5	Chandler	Bing	22.99
6	Rachel	Green	25.99
7	Michael	Smith	18.99
8	Ross	Geller	12.99
9	Robin	Scherbatsky	13.99
10	Monica	Geller	8.99
11	Ted	Mosby	16.99
12	Barney	Stinson	6.99
13	Lily	Aldrin	24.99
14	Joey	Tribbiani	11.99

4. Retrieve the most popular meals (in descending order of popularity):

```
SELECT M_Id, COUNT(M_Id) AS Popularity

FROM Order NATURAL JOIN Meal

GROUP BY M_Id

ORDER BY Popularity DESC;
```

Result:



	M_ID	POPULARITY
1	113	1
2	108	1
3	112	1
4	102	1
5	110	1
6	101	1
7	111	1
8	114	1
9	104	1
10	105	1
11	109	1
12	103	1
13	106	1
14	107	1

Inserting values in the tables:

Employee

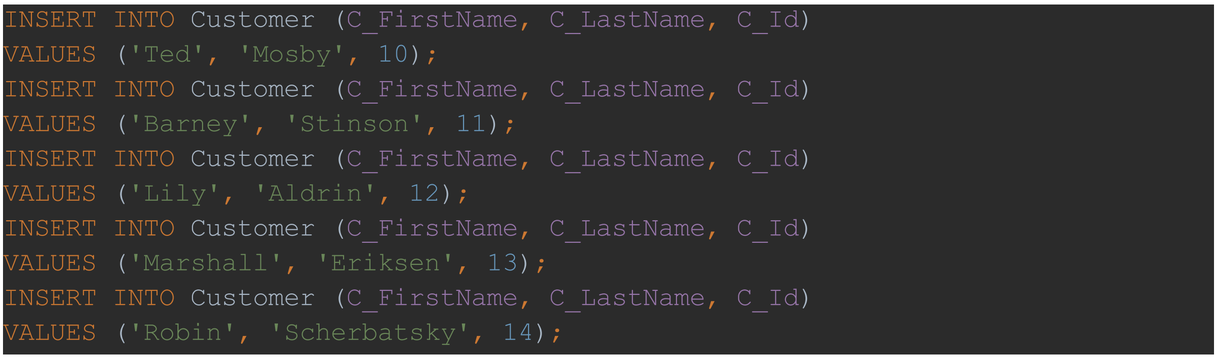
```
INSERT INTO Employee (E_Id, E_FirstName, E_LastName, E_Telephone, E_Address,
E_Age)
VALUES (1, 'John', 'Doe', '054-1234567', '123 Main St.', 30);
INSERT INTO Employee (E_Id, E_FirstName, E_LastName, E_Telephone, E_Address,
E_Age)
VALUES (2, 'Jane', 'Smith', '055-2345678', '456 Oak Ave.', 25);
INSERT INTO Employee (E_Id, E_FirstName, E_LastName, E_Telephone, E_Address,
E_Age)
VALUES (3, 'Bob', 'Johnson', '056-3456789', '789 Maple Rd.', 35);
INSERT INTO Employee (E_Id, E_FirstName, E_LastName, E_Telephone, E_Address,
E_Age)
VALUES (4, 'Alice', 'Williams', '052-4567890', '246 Elm St.', 28);
INSERT INTO Employee (E_Id, E_FirstName, E_LastName, E_Telephone, E_Address,
E_Age)
VALUES (5, 'Mike', 'Brown', '050-5678901', '135 Pine Ave.', 42);
INSERT INTO Employee (E_Id, E_FirstName, E_LastName, E_Telephone, E_Address,
E_Age)
VALUES (6, 'Karen', 'Taylor', '058-6789012', '678 Cedar Ln.', 31);
INSERT INTO Employee (E_Id, E_FirstName, E_LastName, E_Telephone, E_Address,
E_Age)
VALUES (7, 'David', 'Wilson', '051-7890123', '910 Oak Rd.', 27);
INSERT INTO Employee (E_Id, E_FirstName, E_LastName, E_Telephone, E_Address,
E_Age)
VALUES (8, 'Amy', 'Miller', '059-8901234', '345 Elm Ave.', 29);
INSERT INTO Employee (E_Id, E_FirstName, E_LastName, E_Telephone, E_Address,
E_Age)
VALUES (9, 'Chris', 'Lee', '056-9012345', '789 Maple St.', 33);
INSERT INTO Employee (E_Id, E_FirstName, E_LastName, E_Telephone, E_Address,
E_Age)
VALUES (10, 'Maria', 'Garcia', '052-0123456', '246 Oak Ave.', 26);
INSERT INTO Employee (E_Id, E_FirstName, E_LastName, E_Telephone, E_Address,
E_Age)
VALUES (11, 'Tom', 'Anderson', '057-1234567', '123 Pine St.', 38);
INSERT INTO Employee (E_Id, E_FirstName, E_LastName, E_Telephone, E_Address,
E_Age)
VALUES (12, 'Emily', 'Clark', '055-2345678', '456 Cedar Ave.', 24);
INSERT INTO Employee (E_Id, E_FirstName, E_LastName, E_Telephone, E_Address,
E_Age)
VALUES (13, 'Josh', 'Wright', '058-3456789', '789 Maple Rd.', 30);
INSERT INTO Employee (E_Id, E_FirstName, E_LastName, E_Telephone, E_Address,
E_Age)
VALUES (14, 'Samantha', 'Martin', '050-4567890', '246 Oak St.', 32);
```

Supplier

```sql
INSERT INTO Supplier (S_Company, S_Telephone, S_Address)
VALUES ('ABC Inc.', '050-1234567', '123 Main St.');
INSERT INTO Supplier (S_Company, S_Telephone, S_Address)
VALUES ('XYZ Corp.', '055-2345678', '456 Oak Ave.');
INSERT INTO Supplier (S_Company, S_Telephone, S_Address)
VALUES ('123 Co.', '056-3456789', '789 Maple Rd.');
INSERT INTO Supplier (S_Company, S_Telephone, S_Address)
VALUES ('456 Ltd.', '052-4567890', '246 Elm St.');
INSERT INTO Supplier (S_Company, S_Telephone, S_Address)
VALUES ('789 LLC', '050-5678901', '135 Pine Ave.');
INSERT INTO Supplier (S_Company, S_Telephone, S_Address)
VALUES ('DEF Inc.', '058-6789012', '678 Cedar Ln.');
INSERT INTO Supplier (S_Company, S_Telephone, S_Address)
VALUES ('GHI Corp.', '051-7890123', '910 Oak Rd.');
INSERT INTO Supplier (S_Company, S_Telephone, S_Address)
VALUES ('JKL Co.', '059-8901234', '345 Elm Ave.');
INSERT INTO Supplier (S_Company, S_Telephone, S_Address)
VALUES ('MNO Ltd.', '056-9012345', '789 Maple St.');
INSERT INTO Supplier (S_Company, S_Telephone, S_Address)
VALUES ('PQR LLC', '052-0123456', '246 Oak Ave.');
INSERT INTO Supplier (S_Company, S_Telephone, S_Address)
VALUES ('STU Inc.', '057-1234567', '123 Pine St.');
INSERT INTO Supplier (S_Company, S_Telephone, S_Address)
VALUES ('VWX Corp.', '055-2345678', '456 Cedar Ave.');
INSERT INTO Supplier (S_Company, S_Telephone, S_Address)
VALUES ('YZA Co.', '058-3456789', '789 Maple Rd.');
INSERT INTO Supplier (S_Company, S_Telephone, S_Address)
VALUES ('BCD Ltd.', '050-4567890', '246 Oak St.');
```

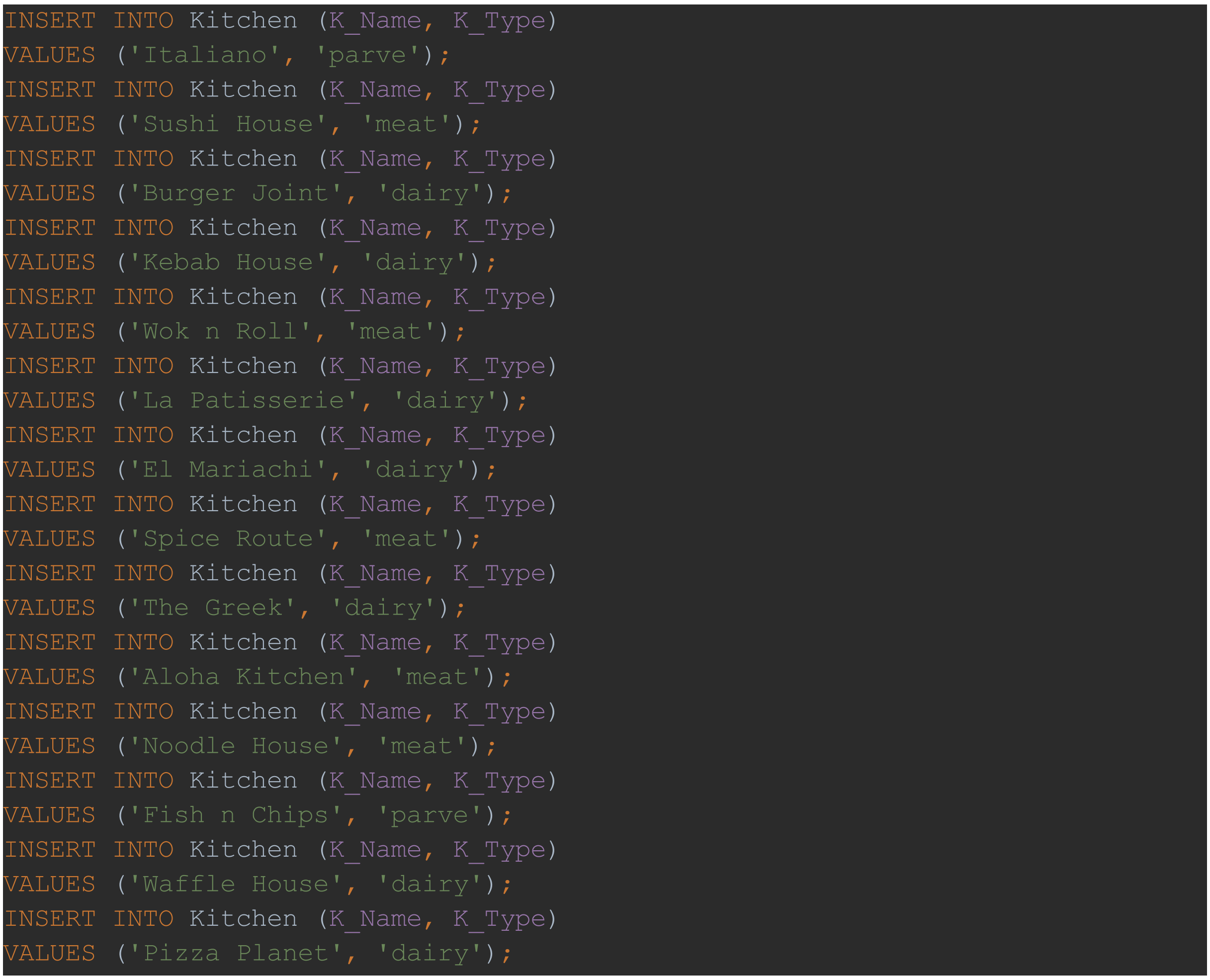
Customer

```sql
INSERT INTO Customer (C_FirstName, C_LastName, C_Id)
VALUES ('John', 'Doe', 1);
INSERT INTO Customer (C_FirstName, C_LastName, C_Id)
VALUES ('Jane', 'Doe', 2);
INSERT INTO Customer (C_FirstName, C_LastName, C_Id)
VALUES ('Michael', 'Smith', 3);
INSERT INTO Customer (C_FirstName, C_LastName, C_Id)
VALUES ('Rachel', 'Green', 4);
INSERT INTO Customer (C_FirstName, C_LastName, C_Id)
VALUES ('Ross', 'Geller', 5);
INSERT INTO Customer (C_FirstName, C_LastName, C_Id)
VALUES ('Monica', 'Geller', 6);
INSERT INTO Customer (C_FirstName, C_LastName, C_Id)
VALUES ('Chandler', 'Bing', 7);
INSERT INTO Customer (C_FirstName, C_LastName, C_Id)
VALUES ('Phoebe', 'Buffay', 8);
INSERT INTO Customer (C_FirstName, C_LastName, C_Id)
VALUES ('Joey', 'Tribbiani', 9);
```



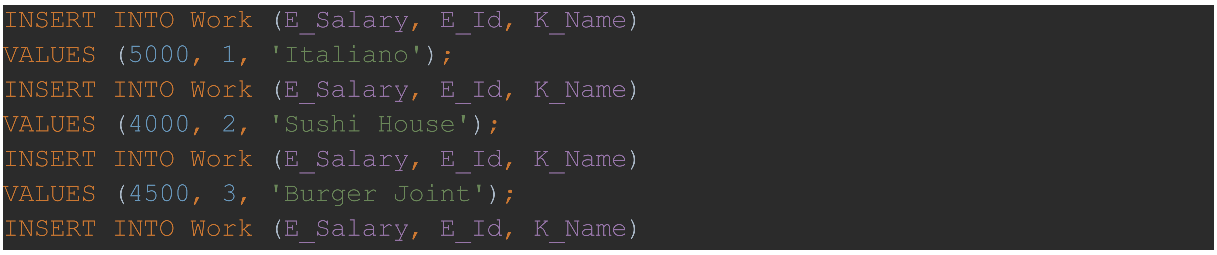
```
INSERT INTO Customer (C_FirstName, C_LastName, C_Id)
VALUES ('Ted', 'Mosby', 10);
INSERT INTO Customer (C_FirstName, C_LastName, C_Id)
VALUES ('Barney', 'Stinson', 11);
INSERT INTO Customer (C_FirstName, C_LastName, C_Id)
VALUES ('Lily', 'Aldrin', 12);
INSERT INTO Customer (C_FirstName, C_LastName, C_Id)
VALUES ('Marshall', 'Eriksen', 13);
INSERT INTO Customer (C_FirstName, C_LastName, C_Id)
VALUES ('Robin', 'Scherbatsky', 14);
```

Kitchen



```
INSERT INTO Kitchen (K_Name, K_Type)
VALUES ('Italiano', 'parve');
INSERT INTO Kitchen (K_Name, K_Type)
VALUES ('Sushi House', 'meat');
INSERT INTO Kitchen (K_Name, K_Type)
VALUES ('Burger Joint', 'dairy');
INSERT INTO Kitchen (K_Name, K_Type)
VALUES ('Kebab House', 'dairy');
INSERT INTO Kitchen (K_Name, K_Type)
VALUES ('Wok n Roll', 'meat');
INSERT INTO Kitchen (K_Name, K_Type)
VALUES ('La Patisserie', 'dairy');
INSERT INTO Kitchen (K_Name, K_Type)
VALUES ('El Mariachi', 'dairy');
INSERT INTO Kitchen (K_Name, K_Type)
VALUES ('Spice Route', 'meat');
INSERT INTO Kitchen (K_Name, K_Type)
VALUES ('The Greek', 'dairy');
INSERT INTO Kitchen (K_Name, K_Type)
VALUES ('Aloha Kitchen', 'meat');
INSERT INTO Kitchen (K_Name, K_Type)
VALUES ('Noodle House', 'meat');
INSERT INTO Kitchen (K_Name, K_Type)
VALUES ('Fish n Chips', 'parve');
INSERT INTO Kitchen (K_Name, K_Type)
VALUES ('Waffle House', 'dairy');
INSERT INTO Kitchen (K_Name, K_Type)
VALUES ('Pizza Planet', 'dairy');
```

Work



```
INSERT INTO Work (E_Salary, E_Id, K_Name)
VALUES (5000, 1, 'Italiano');
INSERT INTO Work (E_Salary, E_Id, K_Name)
VALUES (4000, 2, 'Sushi House');
INSERT INTO Work (E_Salary, E_Id, K_Name)
VALUES (4500, 3, 'Burger Joint');
INSERT INTO Work (E_Salary, E_Id, K_Name)
```

```
VALUES (3500, 4, 'Kebab House');
INSERT INTO Work (E_Salary, E_Id, K_Name)
VALUES (6000, 5, 'Wok n Roll');
INSERT INTO Work (E_Salary, E_Id, K_Name)
VALUES (3500, 6, 'La Patisserie');
INSERT INTO Work (E_Salary, E_Id, K_Name)
VALUES (4000, 7, 'El Mariachi');
INSERT INTO Work (E_Salary, E_Id, K_Name)
VALUES (5000, 8, 'Spice Route');
INSERT INTO Work (E_Salary, E_Id, K_Name)
VALUES (4500, 9, 'The Greek');
INSERT INTO Work (E_Salary, E_Id, K_Name)
VALUES (5500, 10, 'Aloha Kitchen');
INSERT INTO Work (E_Salary, E_Id, K_Name)
VALUES (3500, 11, 'Noodle House');
INSERT INTO Work (E_Salary, E_Id, K_Name)
VALUES (4000, 12, 'Fish n Chips');
INSERT INTO Work (E_Salary, E_Id, K_Name)
VALUES (3000, 13, 'Waffle House');
INSERT INTO Work (E_Salary, E_Id, K_Name)
VALUES (5500, 14, 'Pizza Planet');
```

Ingredient

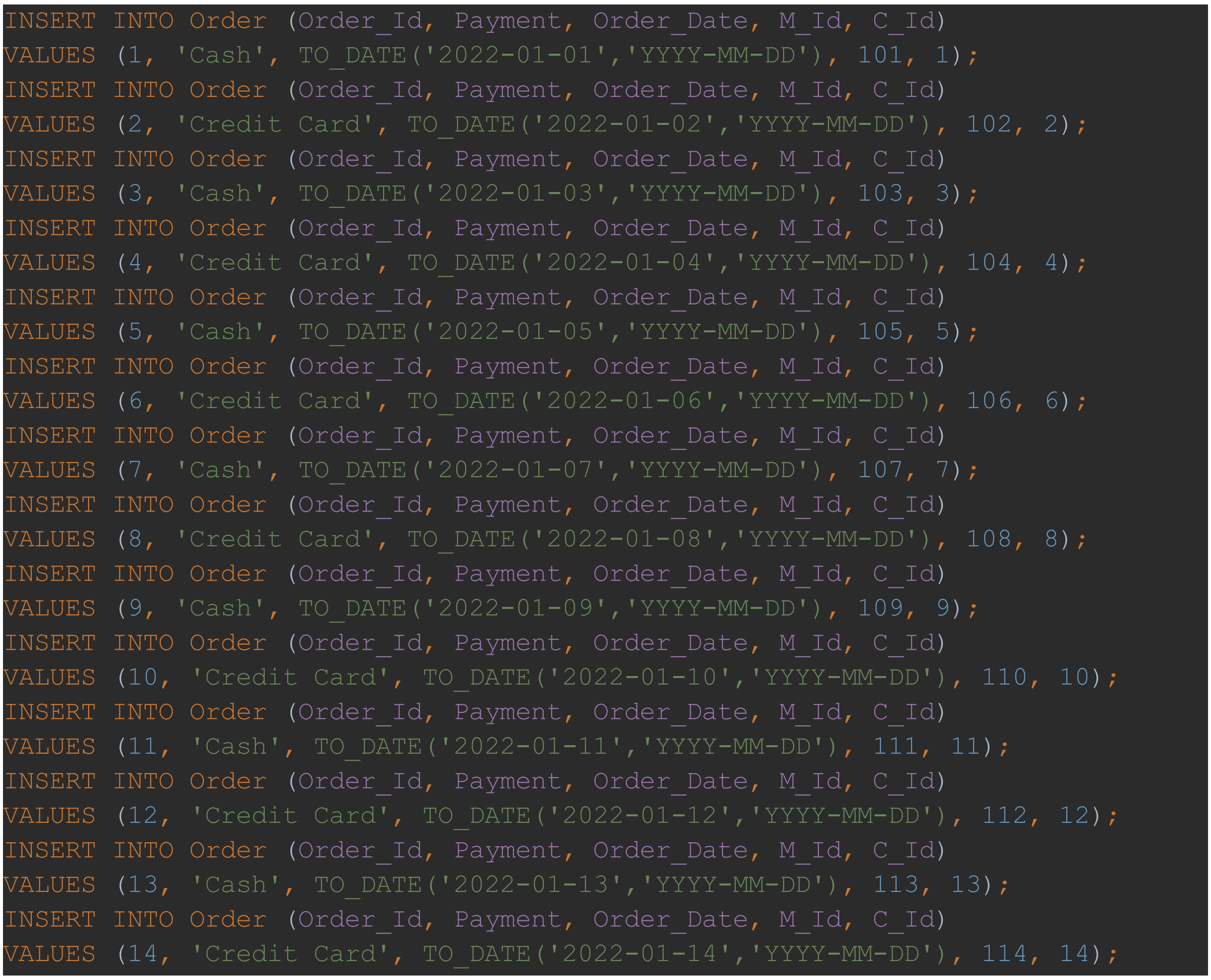
```
INSERT INTO Ingredient (I_Calories, I_Casherout, I_Name, I_Type, I_Id,
I_Price, S_Company)
VALUES (100.5, 'Rabanout', 'Flour', 'parve', 1, 2.5, 'ABC Inc.');
INSERT INTO Ingredient (I_Calories, I_Casherout, I_Name, I_Type, I_Id,
I_Price, S_Company)
VALUES (150.2, 'Badatz', 'Sugar', 'parve', 2, 4.0, 'ABC Inc.');
INSERT INTO Ingredient (I_Calories, I_Casherout, I_Name, I_Type, I_Id,
I_Price, S_Company)
VALUES (80.0, 'Rabanout', 'Salt', 'parve', 3, 1.5, 'XYZ Corp.');
INSERT INTO Ingredient (I_Calories, I_Casherout, I_Name, I_Type, I_Id,
I_Price, S_Company)
VALUES (200.0, 'LaMehadrine', 'Olive Oil', 'parve', 4, 8.0, 'JKL Co.');
INSERT INTO Ingredient (I_Calories, I_Casherout, I_Name, I_Type, I_Id,
I_Price, S_Company)
VALUES (150.0, 'Badatz', 'Soy Sauce', 'parve', 5, 4.5, 'JKL Co.');
INSERT INTO Ingredient (I_Calories, I_Casherout, I_Name, I_Type, I_Id,
I_Price, S_Company)
VALUES (250.0, 'Rabanout', 'Beef', 'meat', 6, 12.0, 'XYZ Corp.');
INSERT INTO Ingredient (I_Calories, I_Casherout, I_Name, I_Type, I_Id,
I_Price, S_Company)
VALUES (200.0, 'Rabanout', 'Chicken', 'meat', 7, 10.0, '456 Ltd.');
INSERT INTO Ingredient (I_Calories, I_Casherout, I_Name, I_Type, I_Id,
I_Price, S_Company)
VALUES (150.0, 'LaMehadrine', 'Shrimp', 'parve', 8, 15.0, '456 Ltd.');
INSERT INTO Ingredient (I_Calories, I_Casherout, I_Name, I_Type, I_Id,
I_Price, S_Company)
VALUES (100.0, 'Rabanout', 'Lemon', 'parve', 9, 3.0, 'MNO Ltd.');
```

```
INSERT INTO Ingredient (I_Calories, I_Casherout, I_Name, I_Type, I_Id,
I_Price, S_Company)
VALUES (80.0, 'LaMehadrine', 'Tomato', 'parve', 10, 2.5, 'MNO Ltd.');
INSERT INTO Ingredient (I_Calories, I_Casherout, I_Name, I_Type, I_Id,
I_Price, S_Company)
VALUES (50.0, 'Badatz', 'Garlic', 'parve', 11, 1.5, 'VWX Corp.');
INSERT INTO Ingredient (I_Calories, I_Casherout, I_Name, I_Type, I_Id,
I_Price, S_Company)
VALUES (30.0, 'Badatz', 'Onion', 'parve', 12, 1.0, 'VWX Corp.');
INSERT INTO Ingredient (I_Calories, I_Casherout, I_Name, I_Type, I_Id,
I_Price, S_Company)
VALUES (120.0, 'Rabanout', 'Paprika', 'parve', 13, 2.0, 'BCD Ltd.');
INSERT INTO Ingredient (I_Calories, I_Casherout, I_Name, I_Type, I_Id,
I_Price, S_Company)
VALUES (100.0, 'Badatz', 'Cinnamon', 'parve', 14, 3.0, 'BCD Ltd.');
```

Meal

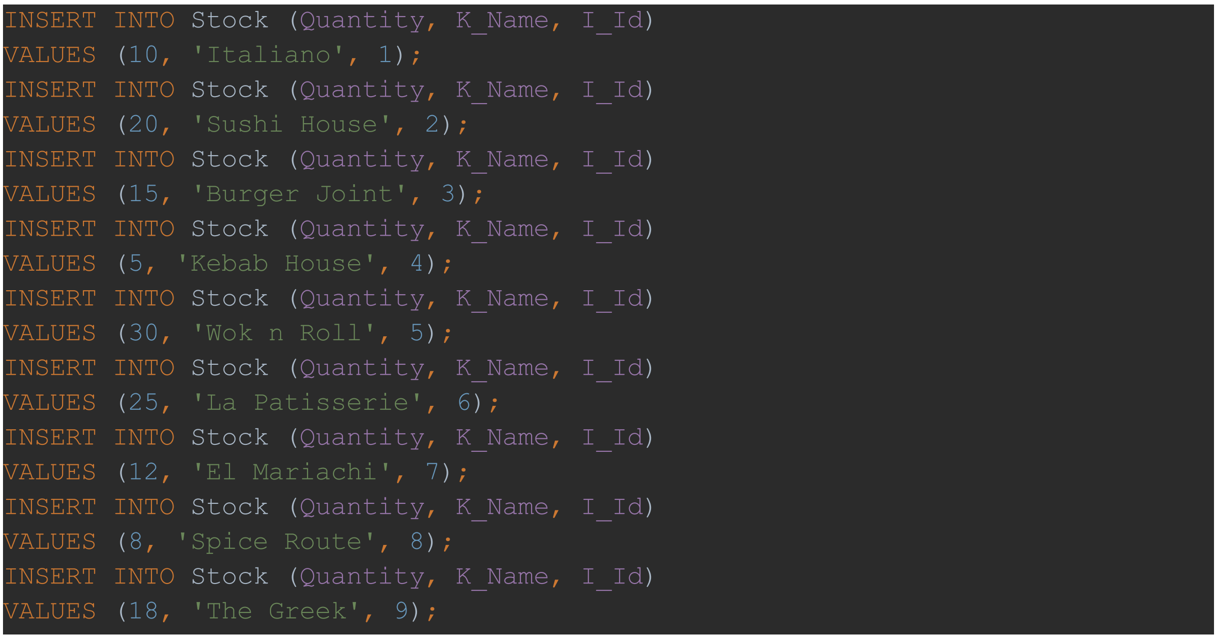
```
INSERT INTO Meal (M_Name, M_Price, M_Id, K_Name)
VALUES ('Hamburger', 10.99, 101, 'Italiano');
INSERT INTO Meal (M_Name, M_Price, M_Id, K_Name)
VALUES ('Pizza', 15.99, 102, 'Sushi House');
INSERT INTO Meal (M_Name, M_Price, M_Id, K_Name)
VALUES ('Sushi', 18.99, 103, 'Burger Joint');
INSERT INTO Meal (M_Name, M_Price, M_Id, K_Name)
VALUES ('Steak', 25.99, 104, 'Kebab House');
INSERT INTO Meal (M_Name, M_Price, M_Id, K_Name)
VALUES ('Pad Thai', 12.99, 105, 'Wok n Roll');
INSERT INTO Meal (M_Name, M_Price, M_Id, K_Name)
VALUES ('Tacos', 8.99, 106, 'La Patisserie');
INSERT INTO Meal (M_Name, M_Price, M_Id, K_Name)
VALUES ('Salmon', 22.99, 107, 'El Mariachi');
INSERT INTO Meal (M_Name, M_Price, M_Id, K_Name)
VALUES ('Fried Rice', 9.99, 108, 'Spice Route');
INSERT INTO Meal (M_Name, M_Price, M_Id, K_Name)
VALUES ('Pasta', 11.99, 109, 'The Greek');
INSERT INTO Meal (M_Name, M_Price, M_Id, K_Name)
VALUES ('Kebab', 16.99, 110, 'Aloha Kitchen');
INSERT INTO Meal (M_Name, M_Price, M_Id, K_Name)
VALUES ('Soup', 6.99, 111, 'Noodle House');
INSERT INTO Meal (M_Name, M_Price, M_Id, K_Name)
VALUES ('BBQ Ribs', 24.99, 112, 'Fish n Chips');
INSERT INTO Meal (M_Name, M_Price, M_Id, K_Name)
VALUES ('Falafel', 7.99, 113, 'Waffle House');
INSERT INTO Meal (M_Name, M_Price, M_Id, K_Name)
VALUES ('Pho', 13.99, 114, 'Pizza Planet');
```

Order



```
INSERT INTO Order (Order_Id, Payment, Order_Date, M_Id, C_Id)
VALUES (1, 'Cash', TO_DATE('2022-01-01','YYYY-MM-DD'), 101, 1);
INSERT INTO Order (Order_Id, Payment, Order_Date, M_Id, C_Id)
VALUES (2, 'Credit Card', TO_DATE('2022-01-02','YYYY-MM-DD'), 102, 2);
INSERT INTO Order (Order_Id, Payment, Order_Date, M_Id, C_Id)
VALUES (3, 'Cash', TO_DATE('2022-01-03','YYYY-MM-DD'), 103, 3);
INSERT INTO Order (Order_Id, Payment, Order_Date, M_Id, C_Id)
VALUES (4, 'Credit Card', TO_DATE('2022-01-04','YYYY-MM-DD'), 104, 4);
INSERT INTO Order (Order_Id, Payment, Order_Date, M_Id, C_Id)
VALUES (5, 'Cash', TO_DATE('2022-01-05','YYYY-MM-DD'), 105, 5);
INSERT INTO Order (Order_Id, Payment, Order_Date, M_Id, C_Id)
VALUES (6, 'Credit Card', TO_DATE('2022-01-06','YYYY-MM-DD'), 106, 6);
INSERT INTO Order (Order_Id, Payment, Order_Date, M_Id, C_Id)
VALUES (7, 'Cash', TO_DATE('2022-01-07','YYYY-MM-DD'), 107, 7);
INSERT INTO Order (Order_Id, Payment, Order_Date, M_Id, C_Id)
VALUES (8, 'Credit Card', TO_DATE('2022-01-08','YYYY-MM-DD'), 108, 8);
INSERT INTO Order (Order_Id, Payment, Order_Date, M_Id, C_Id)
VALUES (9, 'Cash', TO_DATE('2022-01-09','YYYY-MM-DD'), 109, 9);
INSERT INTO Order (Order_Id, Payment, Order_Date, M_Id, C_Id)
VALUES (10, 'Credit Card', TO_DATE('2022-01-10','YYYY-MM-DD'), 110, 10);
INSERT INTO Order (Order_Id, Payment, Order_Date, M_Id, C_Id)
VALUES (11, 'Cash', TO_DATE('2022-01-11','YYYY-MM-DD'), 111, 11);
INSERT INTO Order (Order_Id, Payment, Order_Date, M_Id, C_Id)
VALUES (12, 'Credit Card', TO_DATE('2022-01-12','YYYY-MM-DD'), 112, 12);
INSERT INTO Order (Order_Id, Payment, Order_Date, M_Id, C_Id)
VALUES (13, 'Cash', TO_DATE('2022-01-13','YYYY-MM-DD'), 113, 13);
INSERT INTO Order (Order_Id, Payment, Order_Date, M_Id, C_Id)
VALUES (14, 'Credit Card', TO_DATE('2022-01-14','YYYY-MM-DD'), 114, 14);
```

Stock



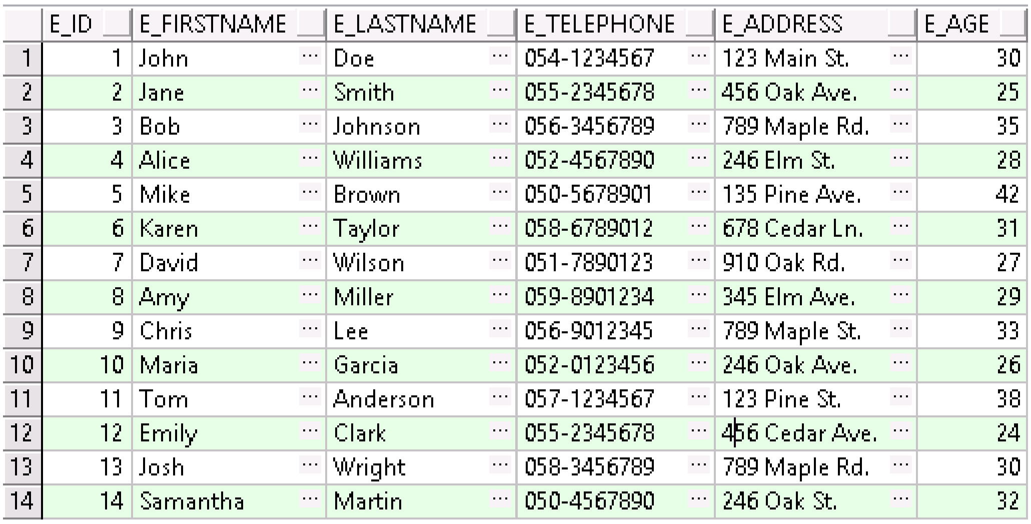
```
INSERT INTO Stock (Quantity, K_Name, I_Id)
VALUES (10, 'Italiano', 1);
INSERT INTO Stock (Quantity, K_Name, I_Id)
VALUES (20, 'Sushi House', 2);
INSERT INTO Stock (Quantity, K_Name, I_Id)
VALUES (15, 'Burger Joint', 3);
INSERT INTO Stock (Quantity, K_Name, I_Id)
VALUES (5, 'Kebab House', 4);
INSERT INTO Stock (Quantity, K_Name, I_Id)
VALUES (30, 'Wok n Roll', 5);
INSERT INTO Stock (Quantity, K_Name, I_Id)
VALUES (25, 'La Patisserie', 6);
INSERT INTO Stock (Quantity, K_Name, I_Id)
VALUES (12, 'El Mariachi', 7);
INSERT INTO Stock (Quantity, K_Name, I_Id)
VALUES (8, 'Spice Route', 8);
INSERT INTO Stock (Quantity, K_Name, I_Id)
VALUES (18, 'The Greek', 9);
```

```
INSERT INTO Stock (Quantity, K_Name, I_Id)
VALUES (7, 'Aloha Kitchen', 10);
INSERT INTO Stock (Quantity, K_Name, I_Id)
VALUES (13, 'Noodle House', 11);
INSERT INTO Stock (Quantity, K_Name, I_Id)
VALUES (22, 'Fish n Chips', 12);
INSERT INTO Stock (Quantity, K_Name, I_Id)
VALUES (9, 'Waffle House', 13);
INSERT INTO Stock (Quantity, K_Name, I_Id)
VALUES (16, 'Pizza Planet', 14);
```

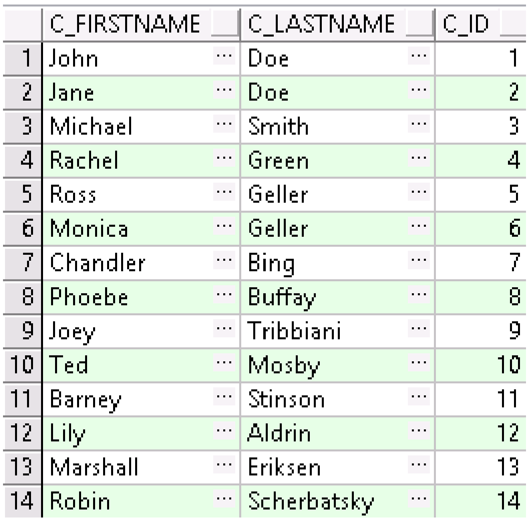
Recipe

```
INSERT INTO Recipe (M_Id, I_Id)
VALUES (101, 1);
INSERT INTO Recipe (M_Id, I_Id)
VALUES (101, 2);
INSERT INTO Recipe (M_Id, I_Id)
VALUES (101, 3);
INSERT INTO Recipe (M_Id, I_Id)
VALUES (102, 4);
INSERT INTO Recipe (M_Id, I_Id)
VALUES (102, 5);
INSERT INTO Recipe (M_Id, I_Id)
VALUES (102, 6);
INSERT INTO Recipe (M_Id, I_Id)
VALUES (103, 7);
INSERT INTO Recipe (M_Id, I_Id)
VALUES (103, 8);
INSERT INTO Recipe (M_Id, I_Id)
VALUES (103, 9);
INSERT INTO Recipe (M_Id, I_Id)
VALUES (104, 10);
INSERT INTO Recipe (M_Id, I_Id)
VALUES (104, 11);
INSERT INTO Recipe (M_Id, I_Id)
VALUES (104, 12);
INSERT INTO Recipe (M_Id, I_Id)
VALUES (105, 13);
INSERT INTO Recipe (M_Id, I_Id)
VALUES (105, 14);
```

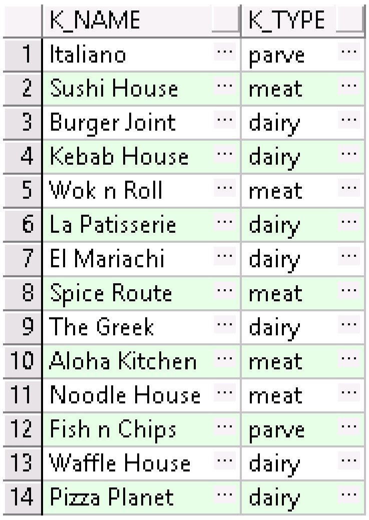
Results



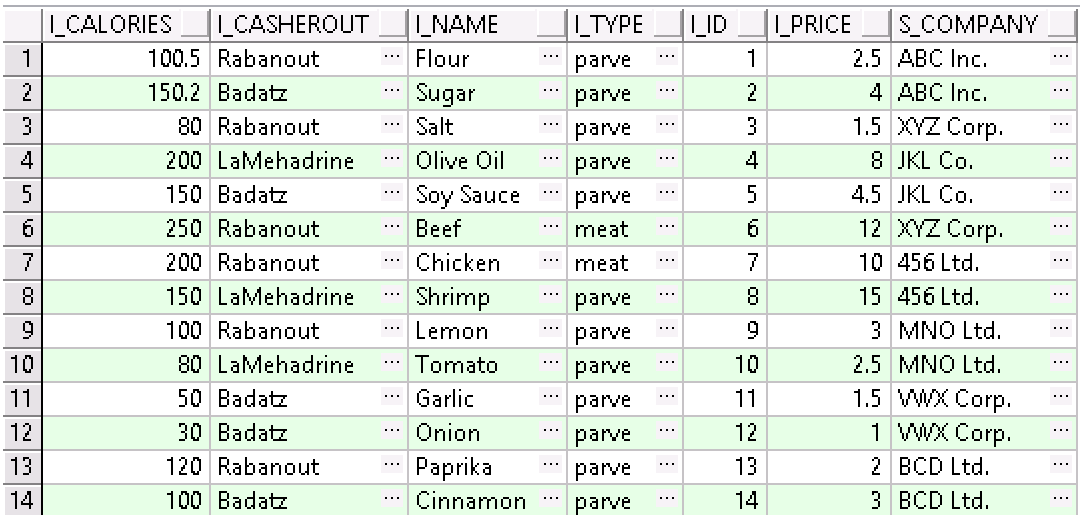
	E_ID	E_FIRSTNAME	E_LASTNAME	E_TELEPHONE	E_ADDRESS	E_AGE
1	1	John	Doe	054-1234567	123 Main St.	30
2	2	Jane	Smith	055-2345678	456 Oak Ave.	25
3	3	Bob	Johnson	056-3456789	789 Maple Rd.	35
4	4	Alice	Williams	052-4567890	246 Elm St.	28
5	5	Mike	Brown	050-5678901	135 Pine Ave.	42
6	6	Karen	Taylor	058-6789012	678 Cedar Ln.	31
7	7	David	Wilson	051-7890123	910 Oak Rd.	27
8	8	Amy	Miller	059-8901234	345 Elm Ave.	29
9	9	Chris	Lee	056-9012345	789 Maple St.	33
10	10	Maria	Garcia	052-0123456	246 Oak Ave.	26
11	11	Tom	Anderson	057-1234567	123 Pine St.	38
12	12	Emily	Clark	055-2345678	456 Cedar Ave.	24
13	13	Josh	Wright	058-3456789	789 Maple Rd.	30
14	14	Samantha	Martin	050-4567890	246 Oak St.	32



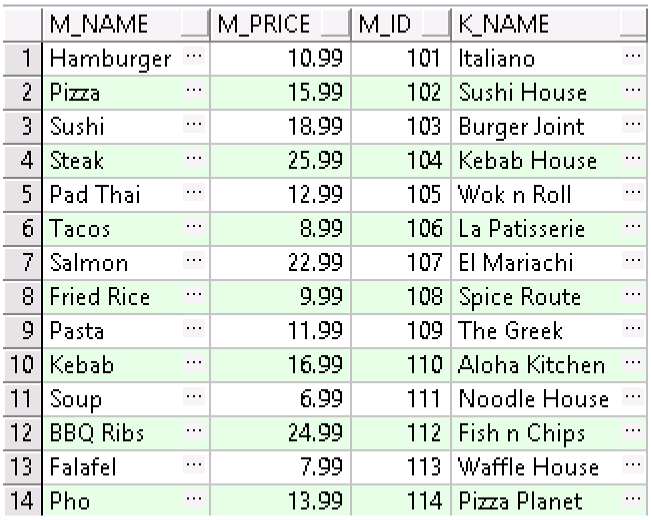
	C_FIRSTNAME	C_LASTNAME	C_ID
1	John	Doe	1
2	Jane	Doe	2
3	Michael	Smith	3
4	Rachel	Green	4
5	Ross	Geller	5
6	Monica	Geller	6
7	Chandler	Bing	7
8	Phoebe	Buffay	8
9	Joey	Tribbiani	9
10	Ted	Mosby	10
11	Barney	Stinson	11
12	Lily	Aldrin	12
13	Marshall	Eriksen	13
14	Robin	Scherbatsky	14



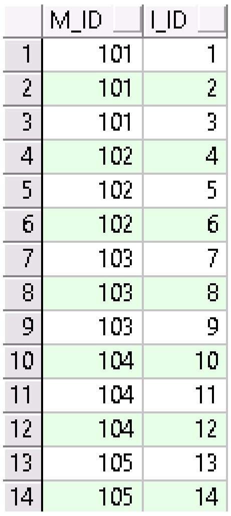
	K_NAME	K_TYPE
1	Italiano	parve
2	Sushi House	meat
3	Burger Joint	dairy
4	Kebab House	dairy
5	Wok n Roll	meat
6	La Patisserie	dairy
7	El Mariachi	dairy
8	Spice Route	meat
9	The Greek	dairy
10	Aloha Kitchen	meat
11	Noodle House	meat
12	Fish n Chips	parve
13	Waffle House	dairy
14	Pizza Planet	dairy



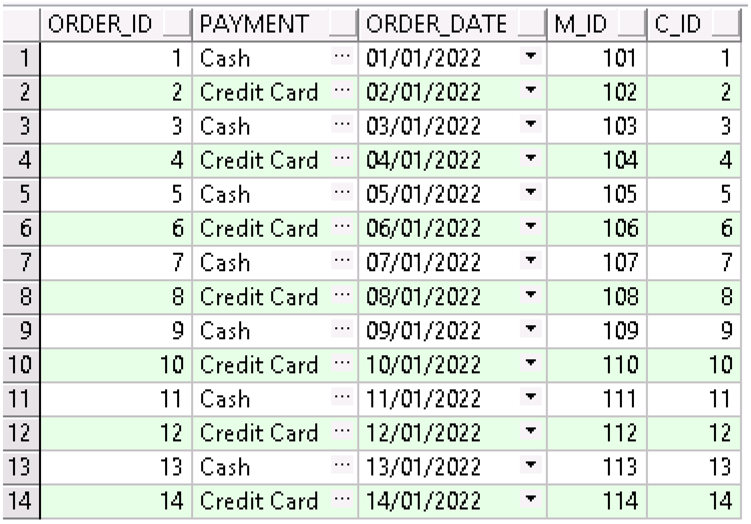
	I_CALORIES	I_CASHEROUT	I_NAME	I_TYPE	I_ID	I_PRICE	S_COMPANY
1	100.5	Rabanout	Flour	parve	1	2.5	ABC Inc.
2	150.2	Badatz	Sugar	parve	2	4	ABC Inc.
3	80	Rabanout	Salt	parve	3	1.5	XYZ Corp.
4	200	LaMehadrine	Olive Oil	parve	4	8	JKL Co.
5	150	Badatz	Soy Sauce	parve	5	4.5	JKL Co.
6	250	Rabanout	Beef	meat	6	12	XYZ Corp.
7	200	Rabanout	Chicken	meat	7	10	456 Ltd.
8	150	LaMehadrine	Shrimp	parve	8	15	456 Ltd.
9	100	Rabanout	Lemon	parve	9	3	MNO Ltd.
10	80	LaMehadrine	Tomato	parve	10	2.5	MNO Ltd.
11	50	Badatz	Garlic	parve	11	1.5	VWX Corp.
12	30	Badatz	Onion	parve	12	1	VWX Corp.
13	120	Rabanout	Paprika	parve	13	2	BCD Ltd.
14	100	Badatz	Cinnamon	parve	14	3	BCD Ltd.



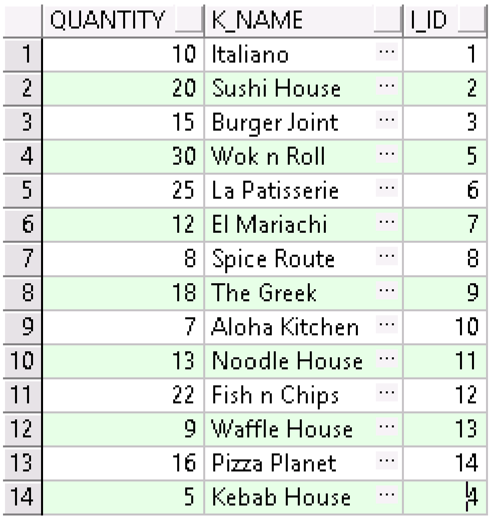
	M_NAME	M_PRICE	M_ID	K_NAME
1	Hamburger	10.99	101	Italiano
2	Pizza	15.99	102	Sushi House
3	Sushi	18.99	103	Burger Joint
4	Steak	25.99	104	Kebab House
5	Pad Thai	12.99	105	Wok n Roll
6	Tacos	8.99	106	La Patisserie
7	Salmon	22.99	107	El Mariachi
8	Fried Rice	9.99	108	Spice Route
9	Pasta	11.99	109	The Greek
10	Kebab	16.99	110	Aloha Kitchen
11	Soup	6.99	111	Noodle House
12	BBQ Ribs	24.99	112	Fish n Chips
13	Falafel	7.99	113	Waffle House
14	Pho	13.99	114	Pizza Planet



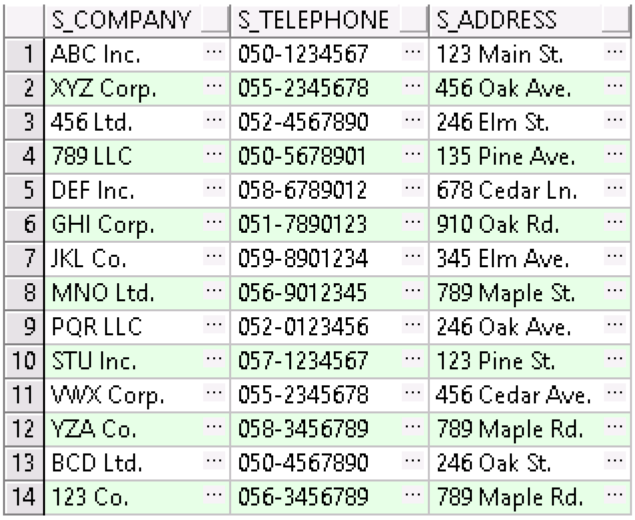
	M_ID	I_ID
1	101	1
2	101	2
3	101	3
4	102	4
5	102	5
6	102	6
7	103	7
8	103	8
9	103	9
10	104	10
11	104	11
12	104	12
13	105	13
14	105	14



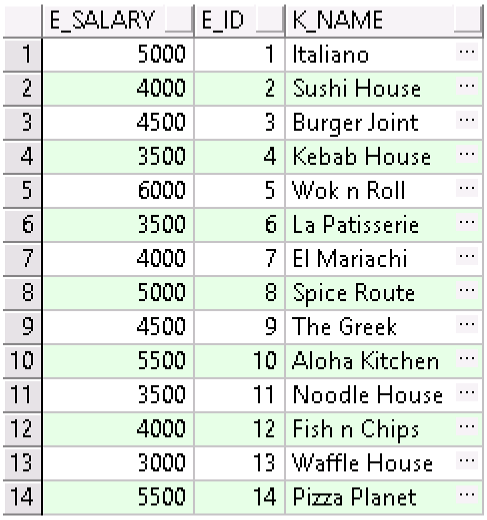
	ORDER_ID	PAYMENT	ORDER_DATE	M_ID	C_ID
1	1	Cash	01/01/2022	101	1
2	2	Credit Card	02/01/2022	102	2
3	3	Cash	03/01/2022	103	3
4	4	Credit Card	04/01/2022	104	4
5	5	Cash	05/01/2022	105	5
6	6	Credit Card	06/01/2022	106	6
7	7	Cash	07/01/2022	107	7
8	8	Credit Card	08/01/2022	108	8
9	9	Cash	09/01/2022	109	9
10	10	Credit Card	10/01/2022	110	10
11	11	Cash	11/01/2022	111	11
12	12	Credit Card	12/01/2022	112	12
13	13	Cash	13/01/2022	113	13
14	14	Credit Card	14/01/2022	114	14



	QUANTITY	K_NAME	I_ID
1	10	Italiano	1
2	20	Sushi House	2
3	15	Burger Joint	3
4	30	Wok n Roll	5
5	25	La Patisserie	6
6	12	El Mariachi	7
7	8	Spice Route	8
8	18	The Greek	9
9	7	Aloha Kitchen	10
10	13	Noodle House	11
11	22	Fish n Chips	12
12	9	Waffle House	13
13	16	Pizza Planet	14
14	5	Kebab House	4



	S_COMPANY	S_TELEPHONE	S_ADDRESS
1	ABC Inc.	050-1234567	123 Main St.
2	XYZ Corp.	055-2345678	456 Oak Ave.
3	456 Ltd.	052-4567890	246 Elm St.
4	789 LLC	050-5678901	135 Pine Ave.
5	DEF Inc.	058-6789012	678 Cedar Ln.
6	GHI Corp.	051-7890123	910 Oak Rd.
7	JKL Co.	059-8901234	345 Elm Ave.
8	MNO Ltd.	056-9012345	789 Maple St.
9	PQR LLC	052-0123456	246 Oak Ave.
10	STU Inc.	057-1234567	123 Pine St.
11	VWX Corp.	055-2345678	456 Cedar Ave.
12	YZA Co.	058-3456789	789 Maple Rd.
13	BCD Ltd.	050-4567890	246 Oak St.
14	123 Co.	056-3456789	789 Maple Rd.

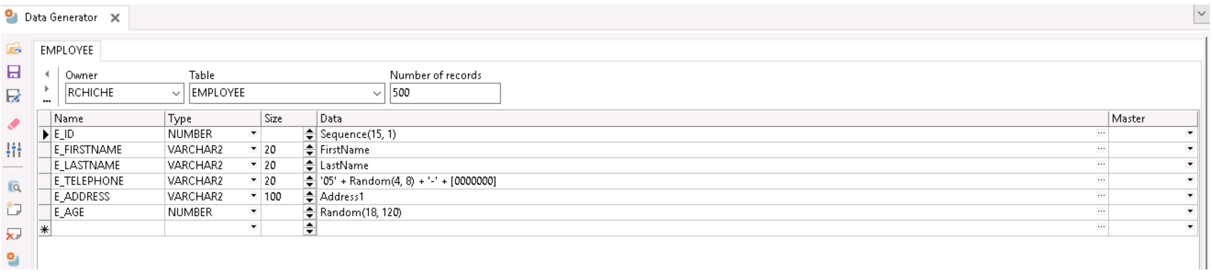


	E_SALARY	E_ID	K_NAME
1	5000	1	Italiano
2	4000	2	Sushi House
3	4500	3	Burger Joint
4	3500	4	Kebab House
5	6000	5	Wok n Roll
6	3500	6	La Patisserie
7	4000	7	El Mariachi
8	5000	8	Spice Route
9	4500	9	The Greek
10	5500	10	Aloha Kitchen
11	3500	11	Noodle House
12	4000	12	Fish n Chips
13	3000	13	Waffle House
14	5500	14	Pizza Planet

Inserting data using Data-Generator

Inserting data in the Employee table, using the data generator of PLSQL.

The configuration of the data:



Owner	Table	Number of records
RCHICHE	EMPLOYEE	500

Name	Type	Size	Data	Master
E_ID	NUMBER		Sequence(15, 1)	
E_FIRSTNAME	VARCHAR2	20	FirstName	
E_LASTNAME	VARCHAR2	20	LastName	
E_TELEPHONE	VARCHAR2	20	'05' + Random(4, 8) + '-' + [0000000]	
E_ADDRESS	VARCHAR2	100	Address1	
E_AGE	NUMBER		Random(18, 120)	

Explanation:

E_ID: Sequence(15,1) -> creating numbers from 15 incrementing by 1.

E_FIRSTNAME: FirstName -> uses first names.

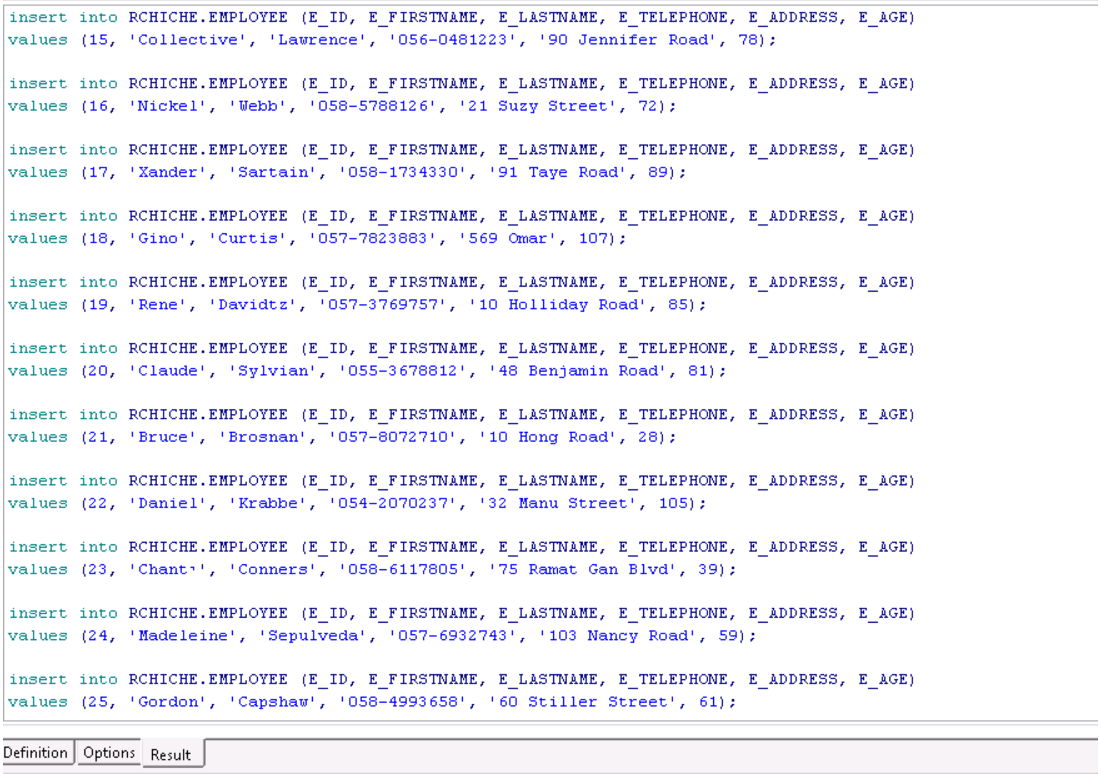
E_LASTNAME: LastName -> uses last names.

E_TELEPHONE: ‘05’ + Random(4,8) + ‘-’ + [0000000] -> uses numbers between 054-0000000 to 058-9999999.

E_ADDRESS: Address1 -> uses addresses.

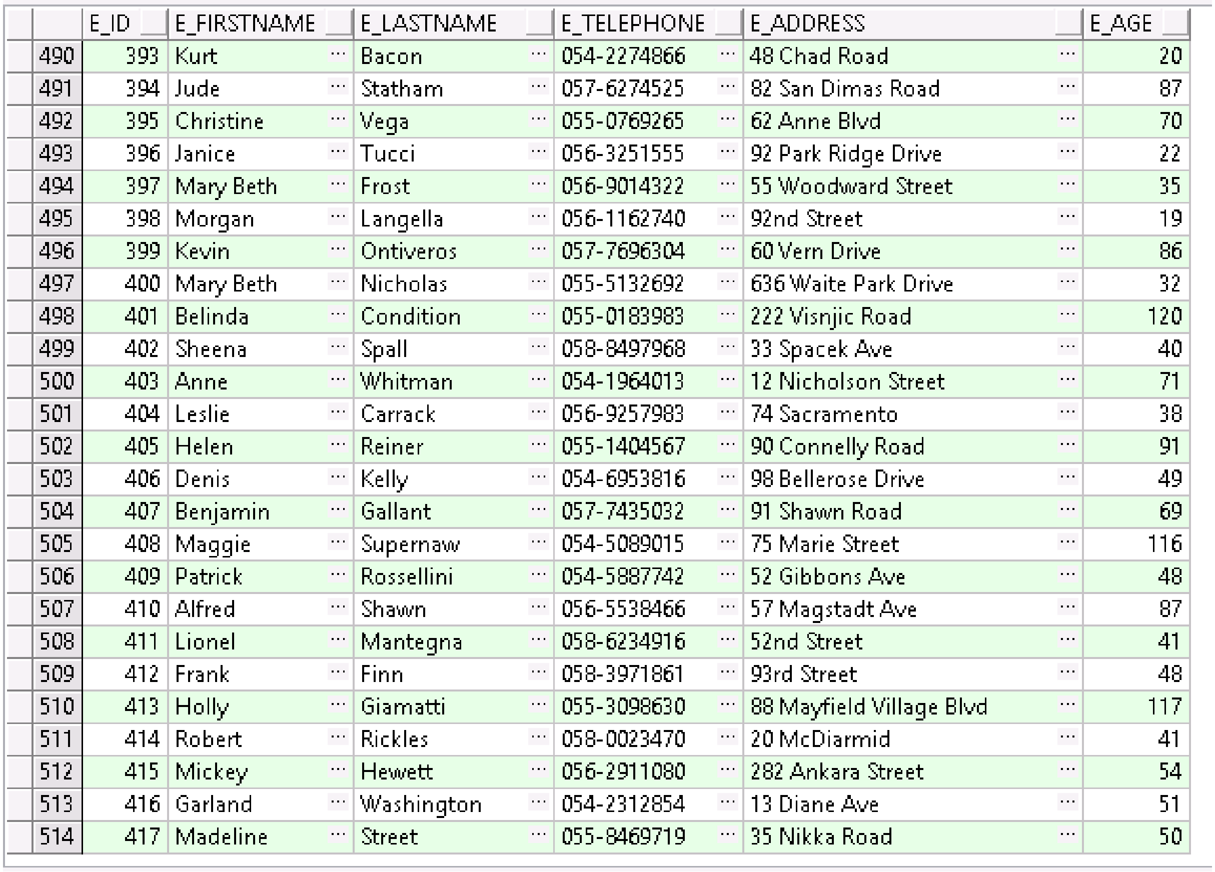
E_AGE: Random(18,120) -> creating numbers between 18 to 120.

Result:



```
insert into RCHICHE.EMPLOYEE (E_ID, E_FIRSTNAME, E_LASTNAME, E_TELEPHONE, E_ADDRESS, E_AGE)
values (15, 'Collective', 'Lawrence', '056-0481223', '90 Jennifer Road', 78);

insert into RCHICHE.EMPLOYEE (E_ID, E_FIRSTNAME, E_LASTNAME, E_TELEPHONE, E_ADDRESS, E_AGE)
values (16, 'Nickel', 'Webb', '058-5788126', '21 Suzy Street', 72);

insert into RCHICHE.EMPLOYEE (E_ID, E_FIRSTNAME, E_LASTNAME, E_TELEPHONE, E_ADDRESS, E_AGE)
values (17, 'Xander', 'Sartain', '058-1734330', '91 Taye Road', 89);

insert into RCHICHE.EMPLOYEE (E_ID, E_FIRSTNAME, E_LASTNAME, E_TELEPHONE, E_ADDRESS, E_AGE)
values (18, 'Gino', 'Curtis', '057-7823883', '569 Omar', 107);

insert into RCHICHE.EMPLOYEE (E_ID, E_FIRSTNAME, E_LASTNAME, E_TELEPHONE, E_ADDRESS, E_AGE)
values (19, 'Rene', 'Davidtz', '057-3769757', '10 Holliday Road', 85);

insert into RCHICHE.EMPLOYEE (E_ID, E_FIRSTNAME, E_LASTNAME, E_TELEPHONE, E_ADDRESS, E_AGE)
values (20, 'Claude', 'Sylvian', '055-3678812', '48 Benjamin Road', 81);

insert into RCHICHE.EMPLOYEE (E_ID, E_FIRSTNAME, E_LASTNAME, E_TELEPHONE, E_ADDRESS, E_AGE)
values (21, 'Bruce', 'Brosnan', '057-8072710', '10 Hong Road', 28);

insert into RCHICHE.EMPLOYEE (E_ID, E_FIRSTNAME, E_LASTNAME, E_TELEPHONE, E_ADDRESS, E_AGE)
values (22, 'Daniel', 'Krabbe', '054-2070237', '32 Manu Street', 105);

insert into RCHICHE.EMPLOYEE (E_ID, E_FIRSTNAME, E_LASTNAME, E_TELEPHONE, E_ADDRESS, E_AGE)
values (23, 'Chant'', 'Conners', '058-6117805', '75 Ramat Gan Blvd', 39);

insert into RCHICHE.EMPLOYEE (E_ID, E_FIRSTNAME, E_LASTNAME, E_TELEPHONE, E_ADDRESS, E_AGE)
values (24, 'Madeleine', 'Sepulveda', '057-6932743', '103 Nancy Road', 59);

insert into RCHICHE.EMPLOYEE (E_ID, E_FIRSTNAME, E_LASTNAME, E_TELEPHONE, E_ADDRESS, E_AGE)
values (25, 'Gordon', 'Capshaw', '058-4993658', '60 Stiller Street', 61);
```

The data in the table:

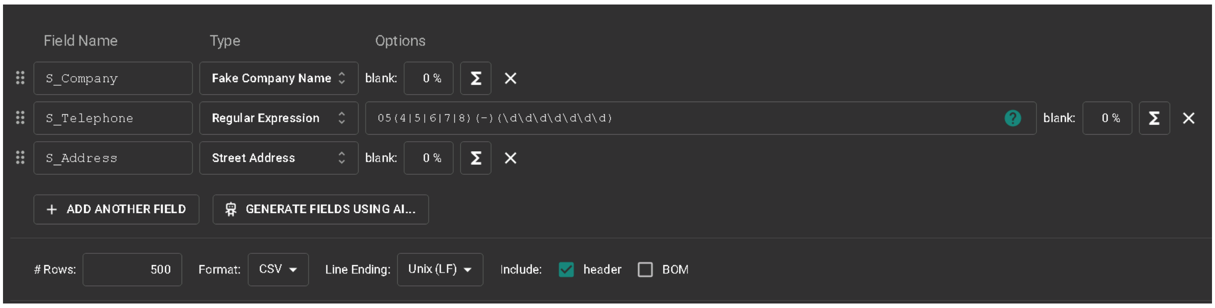


	E_ID	E_FIRSTNAME	E_LASTNAME	E_TELEPHONE	E_ADDRESS	E_AGE
490	393	Kurt	Bacon	054-2274866	48 Chad Road	20
491	394	Jude	Statham	057-6274525	82 San Dimas Road	87
492	395	Christine	Vega	055-0769265	62 Anne Blvd	70
493	396	Janice	Tucci	056-3251555	92 Park Ridge Drive	22
494	397	Mary Beth	Frost	056-9014322	55 Woodward Street	35
495	398	Morgan	Langella	056-1162740	92nd Street	19
496	399	Kevin	Ontiveros	057-7696304	60 Vern Drive	86
497	400	Mary Beth	Nicholas	055-5132692	636 Waite Park Drive	32
498	401	Belinda	Condition	055-0183983	222 Visnjic Road	120
499	402	Sheena	Spall	058-8497968	33 Spacek Ave	40
500	403	Anne	Whitman	054-1964013	12 Nicholson Street	71
501	404	Leslie	Carrack	056-9257983	74 Sacramento	38
502	405	Helen	Reiner	055-1404567	90 Connelly Road	91
503	406	Denis	Kelly	054-6953816	98 Bellerose Drive	49
504	407	Benjamin	Gallant	057-7435032	91 Shawn Road	69
505	408	Maggie	Supernaw	054-5089015	75 Marie Street	116
506	409	Patrick	Rossellini	054-5887742	52 Gibbons Ave	48
507	410	Alfred	Shawn	056-5538466	57 Magstadt Ave	87
508	411	Lionel	Mantegna	058-6234916	52nd Street	41
509	412	Frank	Finn	058-3971861	93rd Street	48
510	413	Holly	Giamatti	055-3098630	88 Mayfield Village Blvd	117
511	414	Robert	Rickles	058-0023470	20 McDiarmid	41
512	415	Mickey	Hewett	056-2911080	282 Ankara Street	54
513	416	Garland	Washington	054-2312854	13 Diane Ave	51
514	417	Madeline	Street	055-8469719	35 Nikka Road	50

Inserting data using mockaroo

Inserting data in the Supplier table, using a CSV from mockaroo.

The configuration of the data:



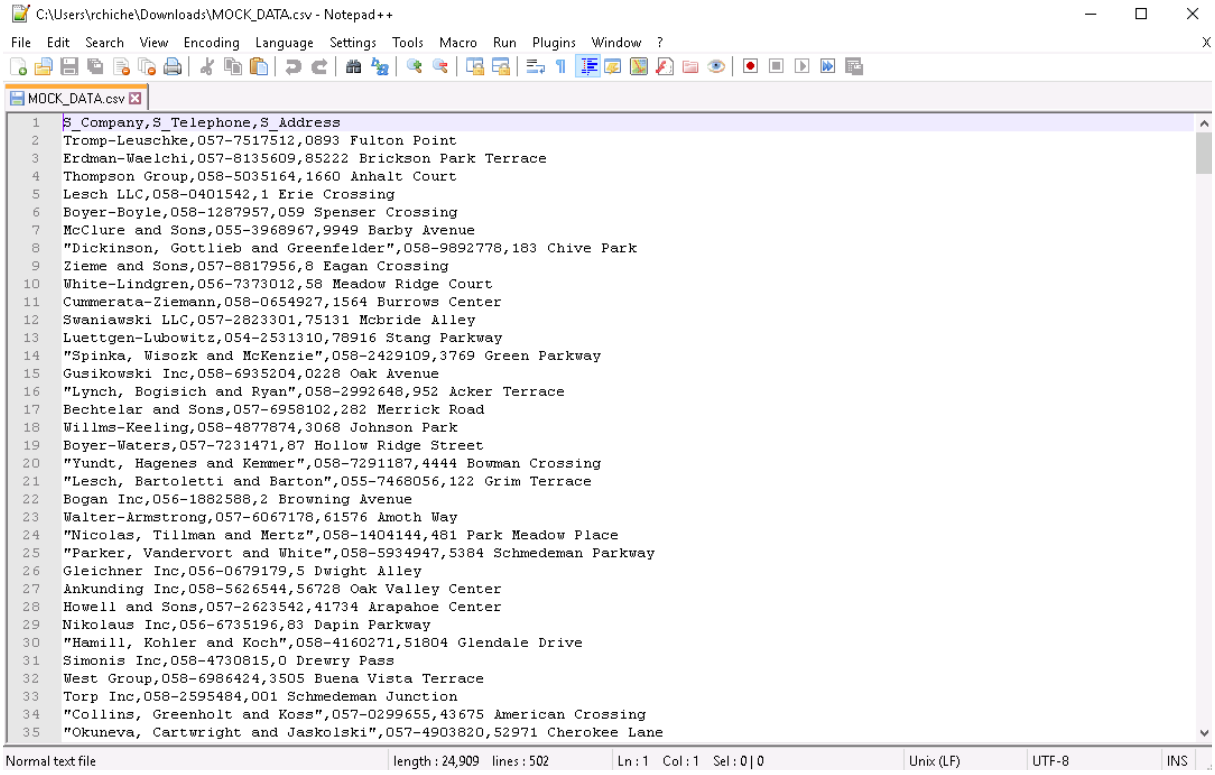
Field Name	Type	Options
S_Company	Fake Company Name	blank: 0 %
S_Telephone	Regular Expression	05(4|5|6|7|8)(-)(\d\d\d\d\d\d\d) blank: 0 %
S_Address	Street Address	blank: 0 %

Rows: 500 Format: CSV Line Ending: Unix (LF) Include: ☑ header □ BOM

Explanation:

S_COMPANY: Fake Company Name -> a fake company name.
S_TELEPHONE: Regular Expression 05(4|5|6|7|8)(-)(\d\d\d\d\d\d\d) -> uses numbers between 054-0000000 to 058-9999999.
S_ADDRESS: Street Address -> uses addresses.

The csv output file:



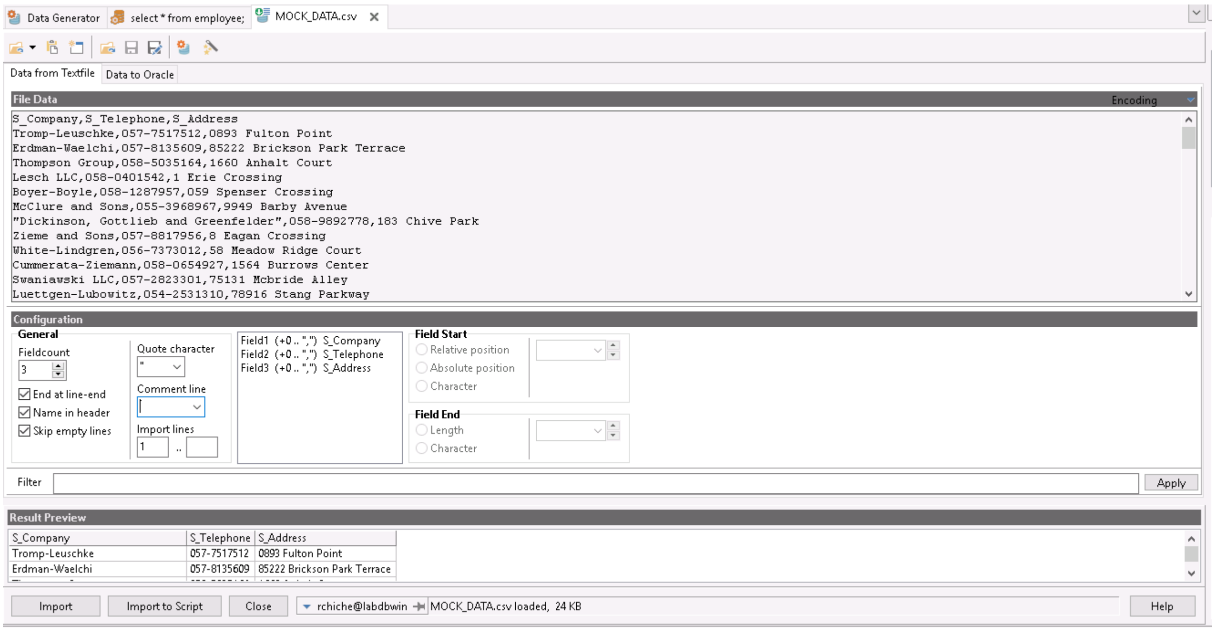
C:\Users\rchiche\Downloads\MOCK_DATA.csv - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
MOCK_DATA.csv

```
S_Company,S_Telephone,S_Address
Tromp-Leuschke,057-7517512,0893 Fulton Point
Erdman-Waelchi,057-8135609,85222 Brickson Park Terrace
Thompson Group,058-5035164,1660 Anhalt Court
Lesch LLC,058-0401542,1 Erie Crossing
Boyer-Boyle,058-1287957,059 Spenser Crossing
McClure and Sons,055-3968967,9949 Barby Avenue
"Dickinson, Gottlieb and Greenfelder",058-9892778,183 Chive Park
Zieme and Sons,057-8817956,8 Eagan Crossing
White-Lindgren,056-7373012,58 Meadow Ridge Court
Cummerata-Ziemann,058-0654927,1564 Burrows Center
Swaniawski LLC,057-2823301,75131 Mcbride Alley
Luettgen-Lubowitz,054-2531310,78916 Stang Parkway
"Spinka, Wisozk and McKenzie",058-2429109,3769 Green Parkway
Gusikowski Inc,058-6935204,0228 Oak Avenue
"Lynch, Bogisich and Ryan",058-2992648,952 Acker Terrace
Bechtelar and Sons,057-6958102,282 Merrick Road
Willms-Keeling,058-4877874,3068 Johnson Park
Boyer-Waters,057-7231471,87 Hollow Ridge Street
"Yundt, Hagenes and Kemmer",058-7291187,4444 Bowman Crossing
"Lesch, Bartoletti and Barton",055-7468056,122 Grim Terrace
Bogan Inc,056-1882588,2 Browning Avenue
Walter-Armstrong,057-6067178,61576 Amoth Way
"Nicolas, Tillman and Mertz",058-1404144,481 Park Meadow Place
"Parker, Vandervort and White",058-5934947,5384 Schmedeman Parkway
Gleichner Inc,056-0679179,5 Dwight Alley
Ankunding Inc,058-5626544,56728 Oak Valley Center
Howell and Sons,057-2623542,41734 Arapahoe Center
Nikolaus Inc,056-6735196,83 Dapin Parkway
"Hamill, Kohler and Koch",058-4160271,51804 Glendale Drive
Simonis Inc,058-4730815,0 Drewry Pass
West Group,058-6986424,3505 Buena Vista Terrace
Torp Inc,058-2595484,001 Schmedeman Junction
"Collins, Greenholt and Koss",057-0299655,43675 American Crossing
"Okuneva, Cartwright and Jaskolski",057-4903820,52971 Cherokee Lane
```

Normal text file   length : 24,909   lines : 502   Ln : 1   Col : 1   Sel : 0 | 0   Unix (LF)   UTF-8   INS

Inserting the CSV file into PLSQL:



```
S_Company,S_Telephone,S_Address
Tromp-Leuschke,057-7517512,0893 Fulton Point
Erdman-Waelchi,057-8135609,85222 Brickson Park Terrace
Thompson Group,058-5035164,1660 Anhalt Court
Lesch LLC,058-0401542,1 Erie Crossing
Boyer-Boyle,058-1287957,059 Spenser Crossing
McClure and Sons,055-3968967,9949 Barby Avenue
"Dickinson, Gottlieb and Greenfelder",058-9892778,183 Chive Park
Zieme and Sons,057-8817956,8 Eagan Crossing
White-Lindgren,056-7373012,58 Meadow Ridge Court
Cummerata-Ziemann,058-0654927,1564 Burrows Center
Swaniawski LLC,057-2823301,75131 Mcbride Alley
Luettgen-Lubowitz,054-2531310,78916 Stang Parkway
```

General

Fieldcount: 3

☑ End at line-end
☑ Name in header
☑ Skip empty lines

Quote character: "

Comment line:

Import lines: 1 ..

```
Field1  (+0 .. ",")  S_Company
Field2  (+0 .. ",")  S_Telephone
Field3  (+0 .. ",")  S_Address
```

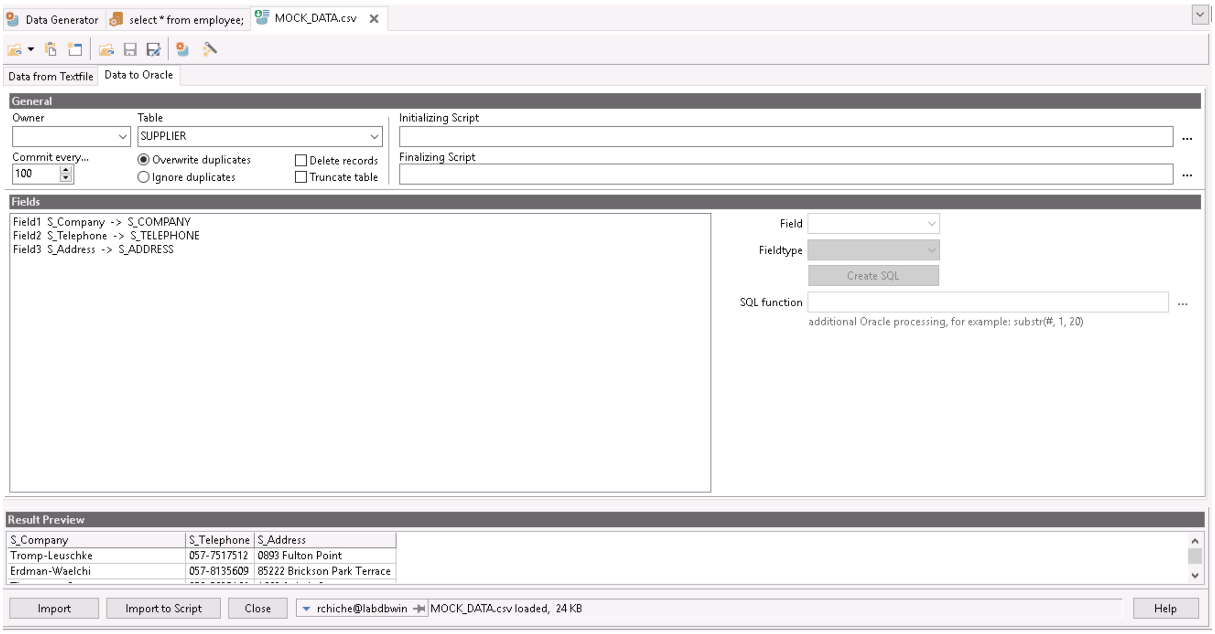
Field Start
○ Relative position
○ Absolute position
○ Character

Field End
○ Length
○ Character

Filter:

S_Company	S_Telephone	S_Address
Tromp-Leuschke	057-7517512	0893 Fulton Point
Erdman-Waelchi	057-8135609	85222 Brickson Park Terrace

rchiche@labdbwin MOCK_DATA.csv loaded, 24 KB



Owner:

Table: SUPPLIER

Commit every...: 100

● Overwrite duplicates
○ Ignore duplicates

□ Delete records
□ Truncate table

Initializing Script:

Finalizing Script:

```
Field1  S_Company  ->  S_COMPANY
Field2  S_Telephone  ->  S_TELEPHONE
Field3  S_Address  ->  S_ADDRESS
```

Field:

Fieldtype:

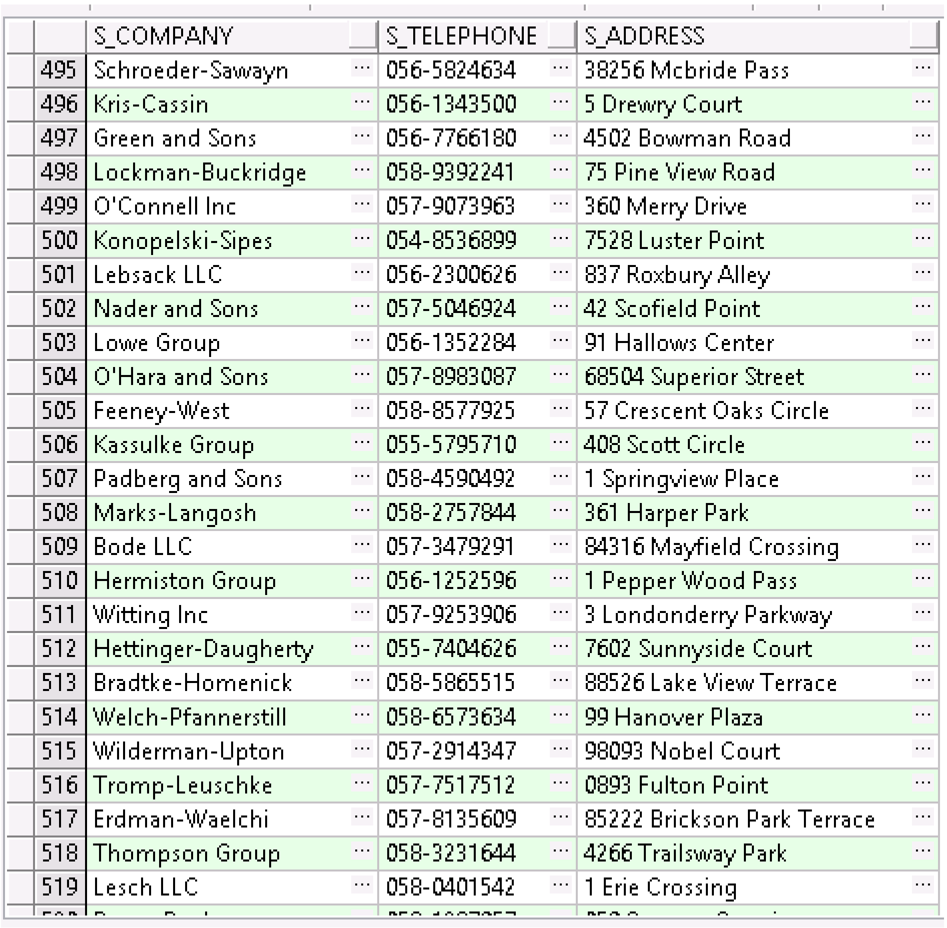
SQL function:

additional Oracle processing, for example: substr(#, 1, 20)

S_Company	S_Telephone	S_Address
Tromp-Leuschke	057-7517512	0893 Fulton Point
Erdman-Waelchi	057-8135609	85222 Brickson Park Terrace

rchiche@labdbwin MOCK_DATA.csv loaded, 24 KB

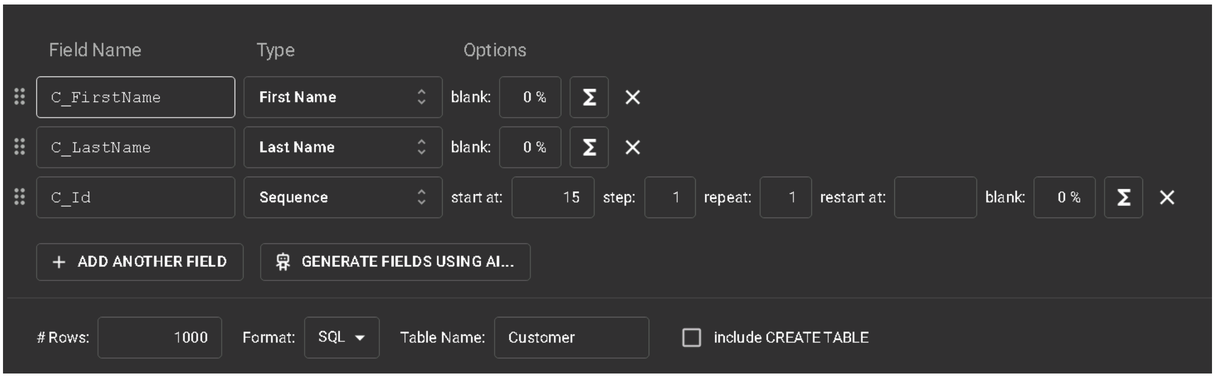
The data in the table:



	S_COMPANY	S_TELEPHONE	S_ADDRESS
495	Schroeder-Sawayn	056-5824634	38256 Mcbride Pass
496	Kris-Cassin	056-1343500	5 Drewry Court
497	Green and Sons	056-7766180	4502 Bowman Road
498	Lockman-Buckridge	058-9392241	75 Pine View Road
499	O'Connell Inc	057-9073963	360 Merry Drive
500	Konopelski-Sipes	054-8536899	7528 Luster Point
501	Lebsack LLC	056-2300626	837 Roxbury Alley
502	Nader and Sons	057-5046924	42 Scofield Point
503	Lowe Group	056-1352284	91 Hallows Center
504	O'Hara and Sons	057-8983087	68504 Superior Street
505	Feeney-West	058-8577925	57 Crescent Oaks Circle
506	Kassulke Group	055-5795710	408 Scott Circle
507	Padberg and Sons	058-4590492	1 Springview Place
508	Marks-Langosh	058-2757844	361 Harper Park
509	Bode LLC	057-3479291	84316 Mayfield Crossing
510	Hermiston Group	056-1252596	1 Pepper Wood Pass
511	Witting Inc	057-9253906	3 Londonderry Parkway
512	Hettinger-Daugherty	055-7404626	7602 Sunnyside Court
513	Bradtke-Homenick	058-5865515	88526 Lake View Terrace
514	Welch-Pfannerstill	058-6573634	99 Hanover Plaza
515	Wilderman-Upton	057-2914347	98093 Nobel Court
516	Tromp-Leuschke	057-7517512	0893 Fulton Point
517	Erdman-Waelchi	057-8135609	85222 Brickson Park Terrace
518	Thompson Group	058-3231644	4266 Trailsway Park
519	Lesch LLC	058-0401542	1 Erie Crossing

Inserting data in the Customer table, using SQL commands from mockaroo.

The configuration of the data:



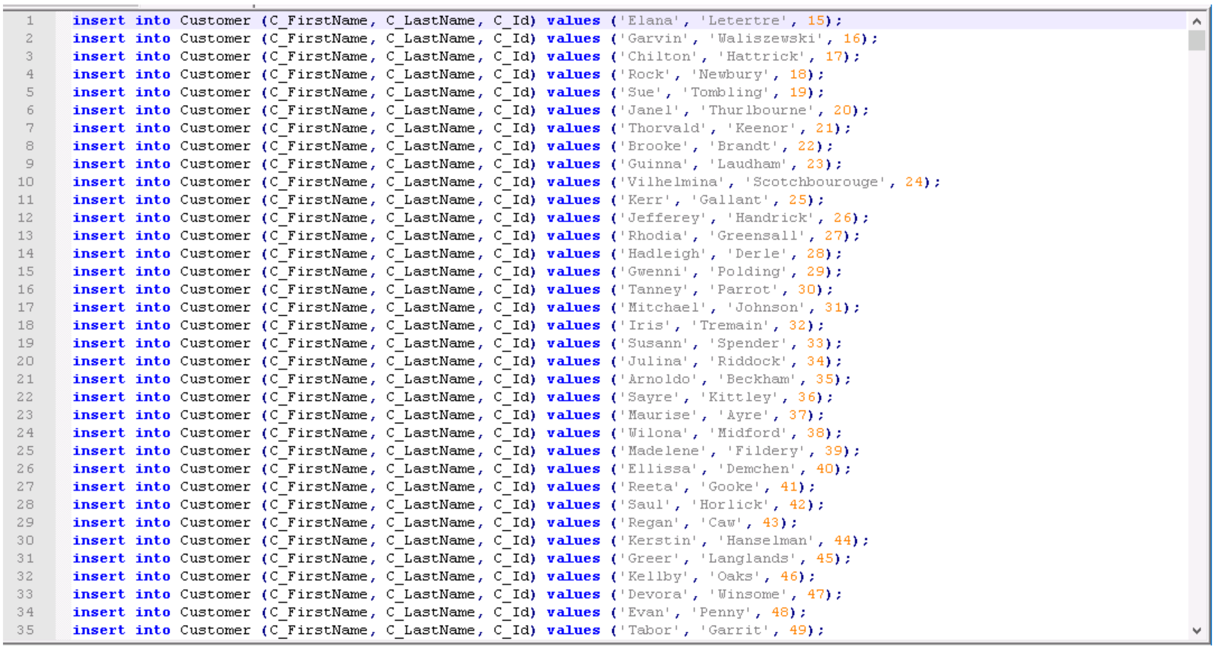
Field Name	Type	Options
C_FirstName	First Name	blank: 0 %
C_LastName	Last Name	blank: 0 %
C_Id	Sequence	start at: 15 step: 1 repeat: 1 restart at: blank: 0 %

Rows: 1000 Format: SQL Table Name: Customer □ include CREATE TABLE

Explanation:

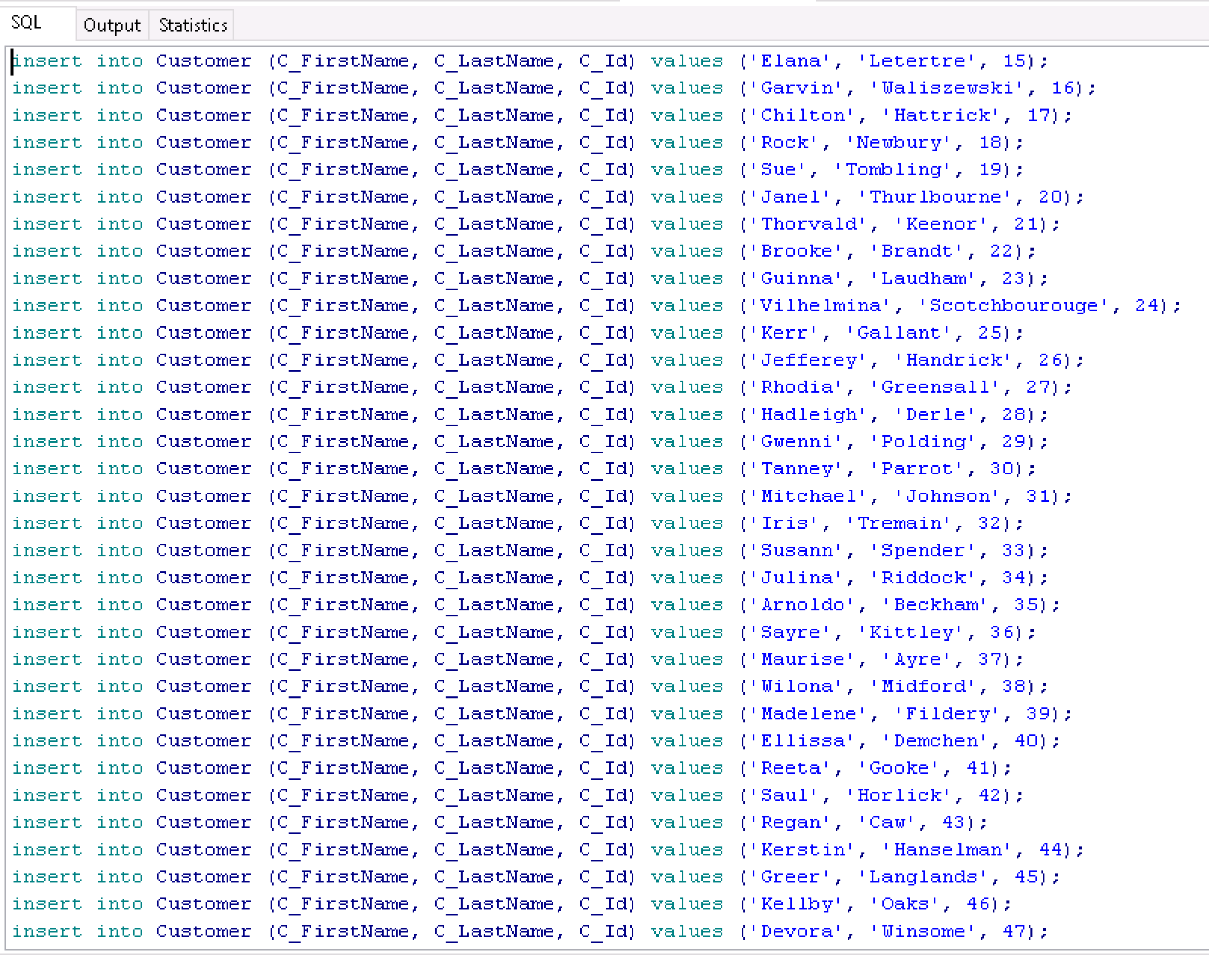
C_FIRSTNAME: First Name -> uses first names.
C_LASTNAME: Last Name -> uses last names.
S_ID: Sequence ->creating numbers from 15 that are incrementing by 1.

The output SQL file:



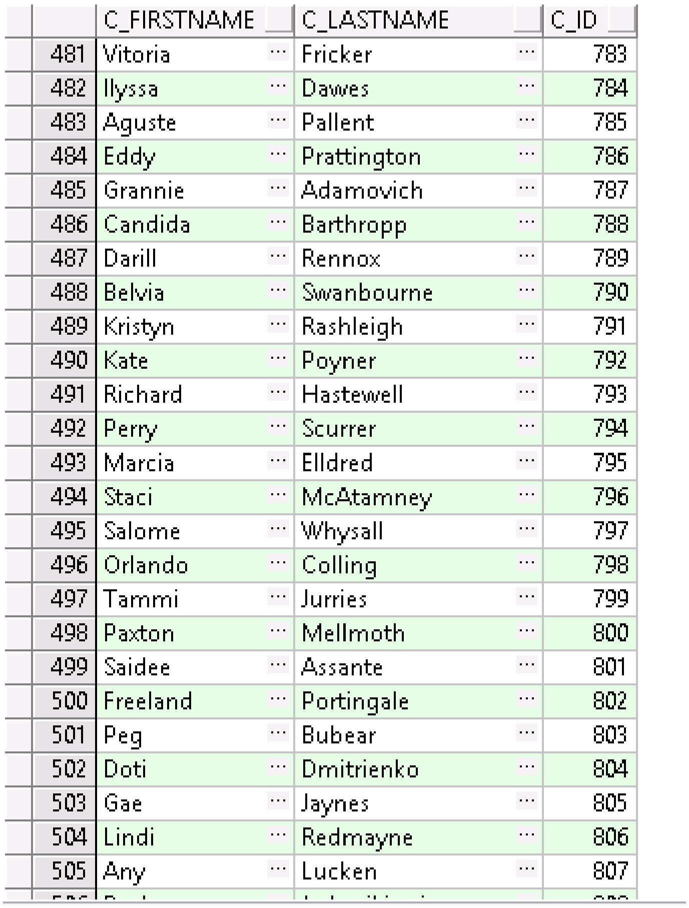
```sql
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Elana', 'Letertre', 15);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Garvin', 'Waliszewski', 16);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Chilton', 'Hattrick', 17);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Rock', 'Newbury', 18);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Sue', 'Tombling', 19);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Janel', 'Thurlbourne', 20);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Thorvald', 'Keenor', 21);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Brooke', 'Brandt', 22);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Guinna', 'Laudham', 23);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Vilhelmina', 'Scotchbourouge', 24);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Kerr', 'Gallant', 25);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Jefferey', 'Handrick', 26);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Rhodia', 'Greensall', 27);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Hadleigh', 'Derle', 28);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Gwenni', 'Polding', 29);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Tanney', 'Parrot', 30);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Mitchael', 'Johnson', 31);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Iris', 'Tremain', 32);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Susann', 'Spender', 33);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Julina', 'Riddock', 34);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Arnoldo', 'Beckham', 35);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Sayre', 'Kittley', 36);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Maurise', 'Ayre', 37);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Wilona', 'Midford', 38);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Madelene', 'Fildery', 39);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Ellissa', 'Demchen', 40);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Reeta', 'Gooke', 41);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Saul', 'Horlick', 42);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Regan', 'Caw', 43);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Kerstin', 'Hanselman', 44);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Greer', 'Langlands', 45);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Kellby', 'Oaks', 46);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Devora', 'Winsome', 47);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Evan', 'Penny', 48);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Tabor', 'Garrit', 49);
```

The SQL commands into PLSQL:



```
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Elana', 'Letertre', 15);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Garvin', 'Waliszewski', 16);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Chilton', 'Hattrick', 17);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Rock', 'Newbury', 18);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Sue', 'Tombling', 19);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Janel', 'Thurlbourne', 20);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Thorvald', 'Keenor', 21);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Brooke', 'Brandt', 22);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Guinna', 'Laudham', 23);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Vilhelmina', 'Scotchbourouge', 24);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Kerr', 'Gallant', 25);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Jefferey', 'Handrick', 26);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Rhodia', 'Greensall', 27);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Hadleigh', 'Derle', 28);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Gwenni', 'Polding', 29);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Tanney', 'Parrot', 30);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Mitchael', 'Johnson', 31);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Iris', 'Tremain', 32);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Susann', 'Spender', 33);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Julina', 'Riddock', 34);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Arnoldo', 'Beckham', 35);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Sayre', 'Kittley', 36);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Maurise', 'Ayre', 37);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Wilona', 'Midford', 38);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Madelene', 'Fildery', 39);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Ellissa', 'Demchen', 40);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Reeta', 'Gooke', 41);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Saul', 'Horlick', 42);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Regan', 'Caw', 43);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Kerstin', 'Hanselman', 44);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Greer', 'Langlands', 45);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Kellby', 'Oaks', 46);
insert into Customer (C_FirstName, C_LastName, C_Id) values ('Devora', 'Winsome', 47);
```

The data in the table:



	C_FIRSTNAME	C_LASTNAME	C_ID
481	Vitoria	Fricker	783
482	Ilyssa	Dawes	784
483	Aguste	Pallent	785
484	Eddy	Prattington	786
485	Grannie	Adamovich	787
486	Candida	Barthropp	788
487	Darill	Rennox	789
488	Belvia	Swanbourne	790
489	Kristyn	Rashleigh	791
490	Kate	Poyner	792
491	Richard	Hastewell	793
492	Perry	Scurrer	794
493	Marcia	Elldred	795
494	Staci	McAtamney	796
495	Salome	Whysall	797
496	Orlando	Colling	798
497	Tammi	Jurries	799
498	Paxton	Mellmoth	800
499	Saidee	Assante	801
500	Freeland	Portingale	802
501	Peg	Bubear	803
502	Doti	Dmitrienko	804
503	Gae	Jaynes	805
504	Lindi	Redmayne	806
505	Any	Lucken	807

Eight queries

1-Query

This query returns a list of employees with the highest salary in their respective kitchen, but only if the kitchen has more than one customer who has ordered a meal from them.

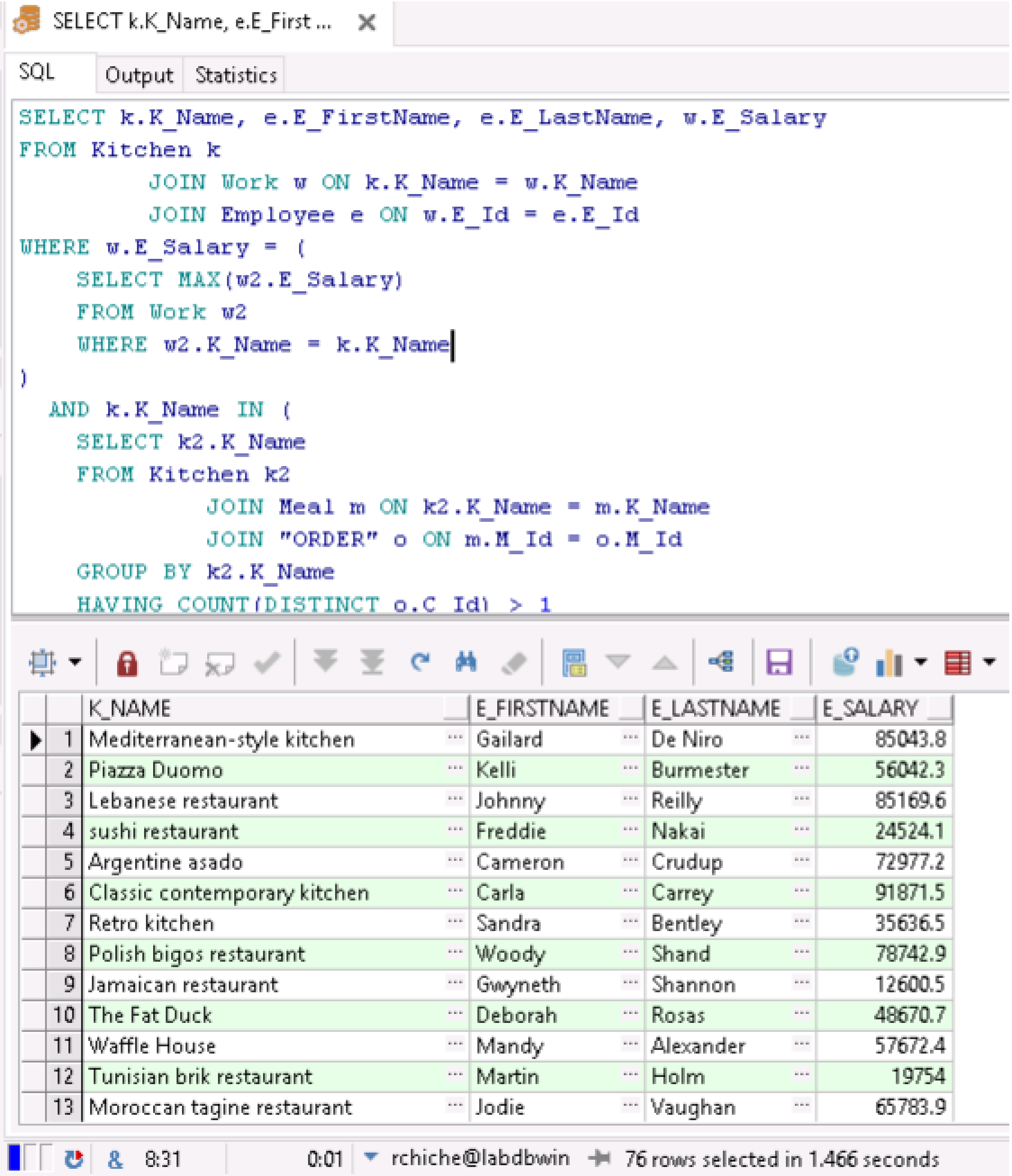
SQL Code

```
SELECT k.K_Name, e.E_FirstName, e.E_LastName, w.E_Salary
FROM Kitchen k
        JOIN Work w ON k.K_Name = w.K_Name
        JOIN Employee e ON w.E_Id = e.E_Id
WHERE w.E_Salary = (
   SELECT MAX(w2.E_Salary)
   FROM Work w2
   WHERE w2.K_Name = k.K_Name
)
 AND k.K_Name IN (
   SELECT k2.K_Name
   FROM Kitchen k2
            JOIN Meal m ON k2.K_Name = m.K_Name
            JOIN "ORDER" o ON m.M_Id = o.M_Id
   GROUP BY k2.K_Name
   HAVING COUNT(DISTINCT o.C_Id) > 1
);
```

Motivation

The motivation for the query is to find the employee with the highest salary in each kitchen, but only for kitchens that have more than 1 customer who has ordered a meal from them. This information can be useful for analyzing the performance of the kitchen and the employees working in it, and for identifying potential areas of improvement.

Result



```
SELECT k.K_Name, e.E_FirstName, e.E_LastName, w.E_Salary
FROM Kitchen k
         JOIN Work w ON k.K_Name = w.K_Name
         JOIN Employee e ON w.E_Id = e.E_Id
WHERE w.E_Salary = (
    SELECT MAX(w2.E_Salary)
    FROM Work w2
    WHERE w2.K_Name = k.K_Name
)
  AND k.K_Name IN (
    SELECT k2.K_Name
    FROM Kitchen k2
             JOIN Meal m ON k2.K_Name = m.K_Name
             JOIN "ORDER" o ON m.M_Id = o.M_Id
    GROUP BY k2.K_Name
    HAVING COUNT(DISTINCT o.C_Id) > 1
```

	K_NAME	E_FIRSTNAME	E_LASTNAME	E_SALARY
1	Mediterranean-style kitchen	Gailard	De Niro	85043.8
2	Piazza Duomo	Kelli	Burmester	56042.3
3	Lebanese restaurant	Johnny	Reilly	85169.6
4	sushi restaurant	Freddie	Nakai	24524.1
5	Argentine asado	Cameron	Crudup	72977.2
6	Classic contemporary kitchen	Carla	Carrey	91871.5
7	Retro kitchen	Sandra	Bentley	35636.5
8	Polish bigos restaurant	Woody	Shand	78742.9
9	Jamaican restaurant	Gwyneth	Shannon	12600.5
10	The Fat Duck	Deborah	Rosas	48670.7
11	Waffle House	Mandy	Alexander	57672.4
12	Tunisian brik restaurant	Martin	Holm	19754
13	Moroccan tagine restaurant	Jodie	Vaughan	65783.9

8:31 0:01 rchiche@labdbwin 76 rows selected in 1.466 seconds

2-Query

Find the names of the kitchens that have at least one employee over the age of retirement, 67, and find the sum of all their salaries.

SQL Code

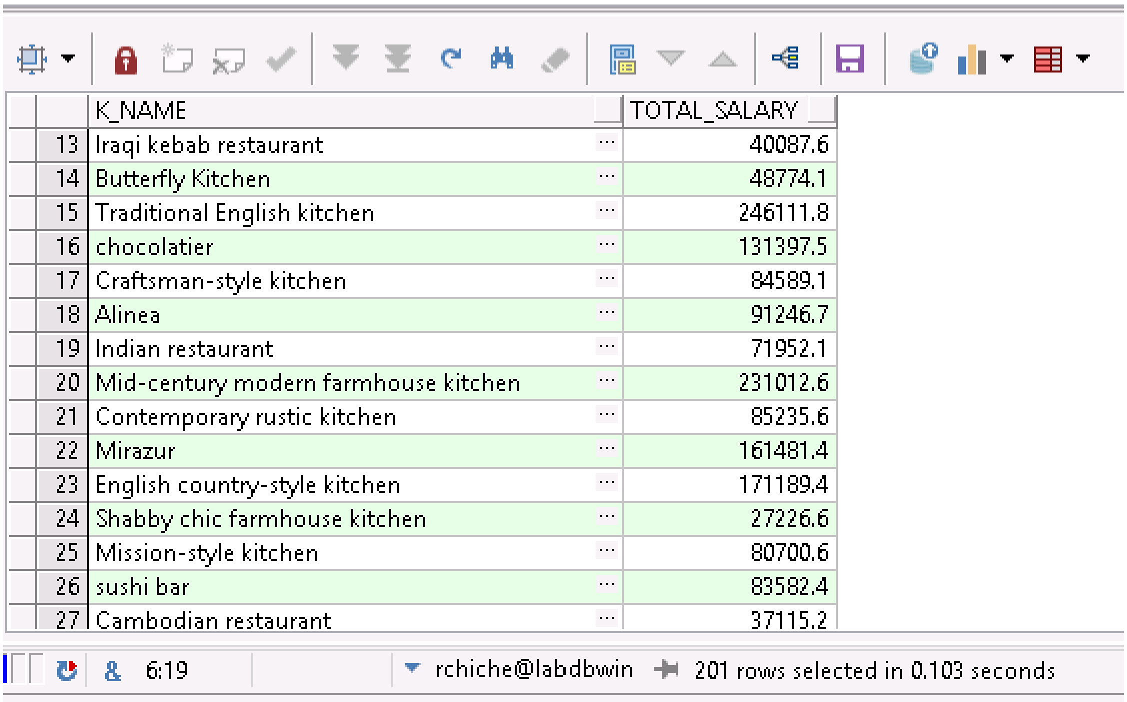
```
SELECT k.K_Name, SUM(w.E_Salary) AS Total_Salary
FROM Kitchen k
        JOIN Work w ON k.K_Name = w.K_Name
        JOIN Employee e ON w.E_Id = e.E_Id
WHERE e.E_Age > 67
GROUP BY k.K_Name;
```

Motivation

The motivation behind the query is to identify kitchens that have at least one employee who has exceeded the retirement age of 67 and to calculate the total salary of all employees working in those kitchens. This information can be useful for various reasons, such as identifying kitchens with older staff and potential labor issues, analyzing the cost of labor for different kitchens, and identifying kitchens that may be at risk of losing experienced staff members due to retirement.

Result

```
SELECT k.K_Name, SUM(w.E_Salary) AS Total_Salary
FROM Kitchen k
JOIN Work w ON k.K_Name = w.K_Name
JOIN Employee e ON w.E_Id = e.E_Id
WHERE e.E_Age > 67
GROUP BY k.K_Name;
```



	K_NAME	TOTAL_SALARY
13	Iraqi kebab restaurant	40087.6
14	Butterfly Kitchen	48774.1
15	Traditional English kitchen	246111.8
16	chocolatier	131397.5
17	Craftsman-style kitchen	84589.1
18	Alinea	91246.7
19	Indian restaurant	71952.1
20	Mid-century modern farmhouse kitchen	231012.6
21	Contemporary rustic kitchen	85235.6
22	Mirazur	161481.4
23	English country-style kitchen	171189.4
24	Shabby chic farmhouse kitchen	27226.6
25	Mission-style kitchen	80700.6
26	sushi bar	83582.4
27	Cambodian restaurant	37115.2

6:19 rchiche@labdbwin 201 rows selected in 0.103 seconds

3-Query

Query that retrieves the most expensive meal for each kitchen.

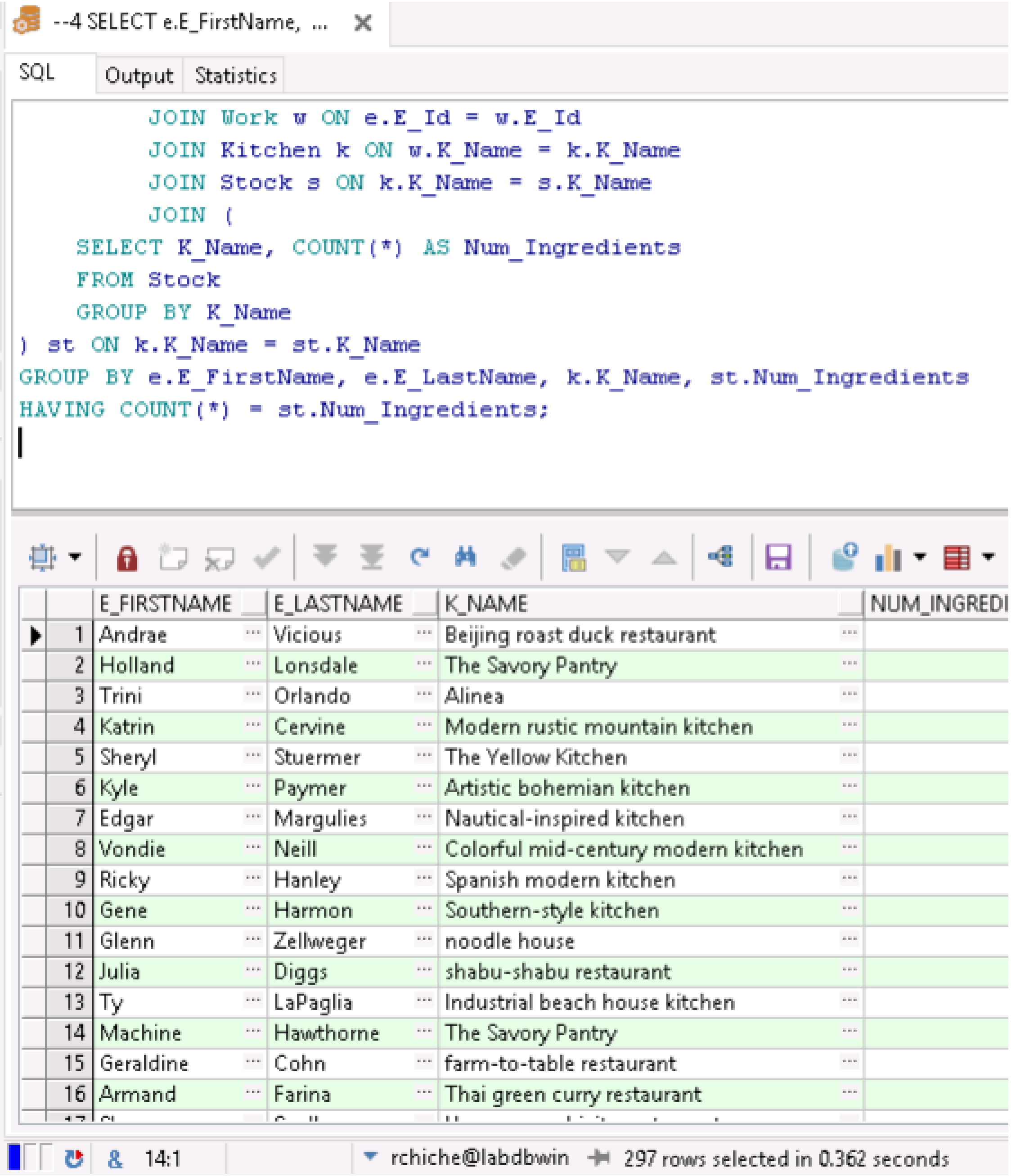
SQL Code

```
SELECT k.K_Name, m.M_Name, MAX(m.M_Price) AS Max_Price
FROM Kitchen k
        JOIN Meal m ON k.K_Name = m.K_Name
        JOIN "ORDER" o ON m.M_Id = o.M_Id
GROUP BY k.K_Name, m.M_Name
HAVING COUNT(*) >= ALL (
   SELECT COUNT(*)
   FROM "ORDER" o2
            JOIN Meal m2 ON o2.M_Id = m2.M_Id
   WHERE m2.K_Name = k.K_Name
   GROUP BY m2.M_Id
)
```

Motivation

The motivation of the query is to determine the most ordered meal in each kitchen along with its price. Knowing which meal is the most popular can help kitchen managers optimize their menu and ensure that they have enough ingredients on hand to meet demand. Additionally, knowing the price of the most ordered meal can help managers set their pricing strategy and determine their profit margins.

Result



```
        JOIN Work w ON e.E_Id = w.E_Id
        JOIN Kitchen k ON w.K_Name = k.K_Name
        JOIN Stock s ON k.K_Name = s.K_Name
        JOIN (
    SELECT K_Name, COUNT(*) AS Num_Ingredients
    FROM Stock
    GROUP BY K_Name
) st ON k.K_Name = st.K_Name
GROUP BY e.E_FirstName, e.E_LastName, k.K_Name, st.Num_Ingredients
HAVING COUNT(*) = st.Num_Ingredients;
```

	E_FIRSTNAME	E_LASTNAME	K_NAME	NUM_INGREDI
1	Andrae	Vicious	Beijing roast duck restaurant	
2	Holland	Lonsdale	The Savory Pantry	
3	Trini	Orlando	Alinea	
4	Katrin	Cervine	Modern rustic mountain kitchen	
5	Sheryl	Stuermer	The Yellow Kitchen	
6	Kyle	Paymer	Artistic bohemian kitchen	
7	Edgar	Margulies	Nautical-inspired kitchen	
8	Vondie	Neill	Colorful mid-century modern kitchen	
9	Ricky	Hanley	Spanish modern kitchen	
10	Gene	Harmon	Southern-style kitchen	
11	Glenn	Zellweger	noodle house	
12	Julia	Diggs	shabu-shabu restaurant	
13	Ty	LaPaglia	Industrial beach house kitchen	
14	Machine	Hawthorne	The Savory Pantry	
15	Geraldine	Cohn	farm-to-table restaurant	
16	Armand	Farina	Thai green curry restaurant	

14:1   rchiche@labdbwin   297 rows selected in 0.362 seconds

4-Query

find employees who work in a kitchen where all the ingredients in the stock are currently available.

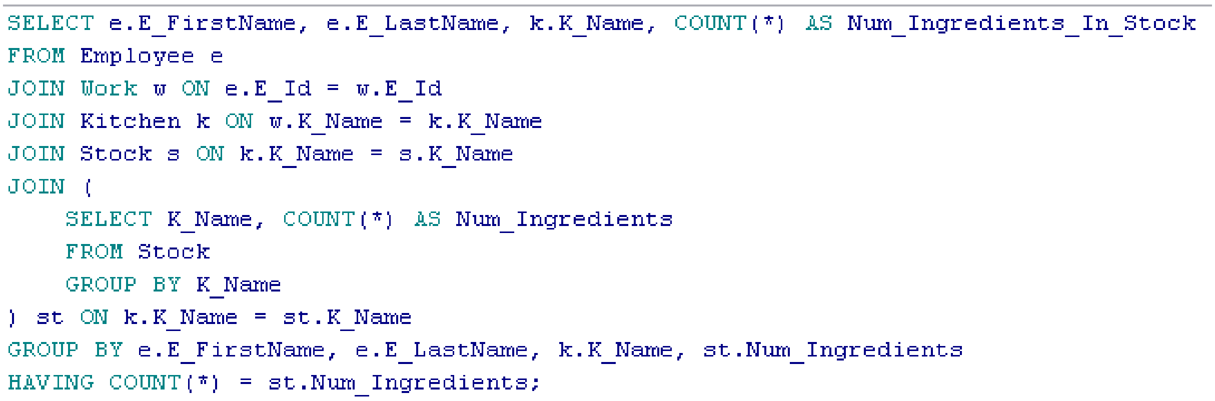
SQL Code

```sql
SELECT e.E_FirstName, e.E_LastName, k.K_Name, COUNT(*) AS
Num_Ingredients_In_Stock
FROM Employee e
        JOIN Work w ON e.E_Id = w.E_Id
        JOIN Kitchen k ON w.K_Name = k.K_Name
        JOIN Stock s ON k.K_Name = s.K_Name
        JOIN (
   SELECT K_Name, COUNT(*) AS Num_Ingredients
   FROM Stock
   GROUP BY K_Name
) st ON k.K_Name = st.K_Name
GROUP BY e.E_FirstName, e.E_LastName, k.K_Name, st.Num_Ingredients
HAVING COUNT(*) = st.Num_Ingredients;
```

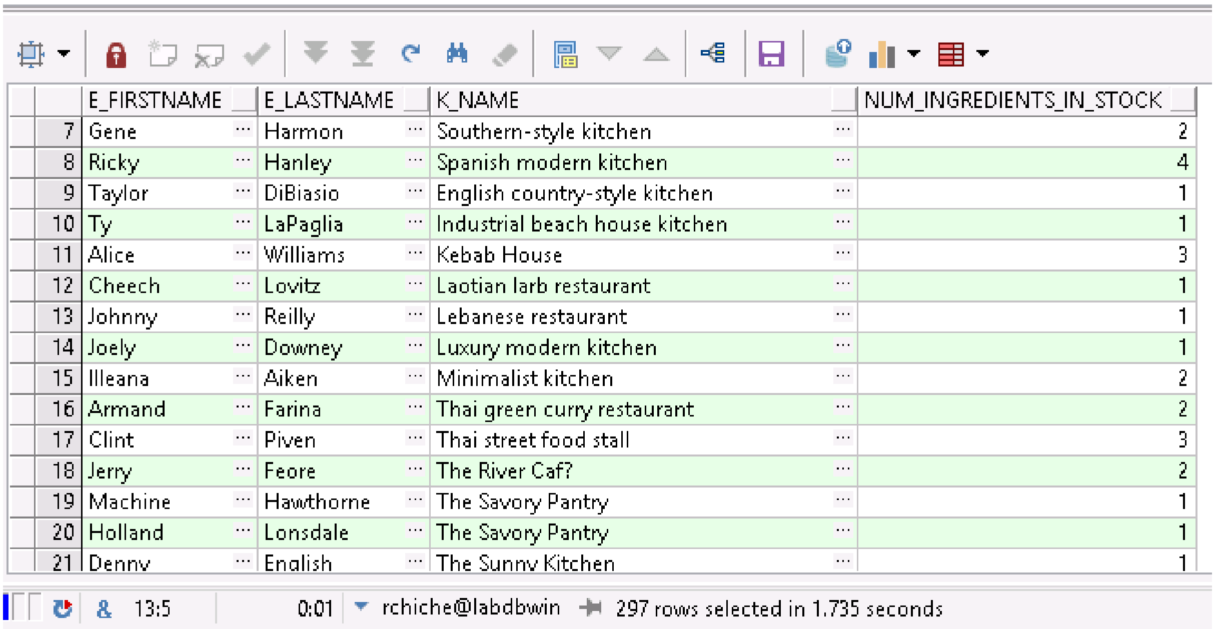
Motivation

The motivation of the query is to find all the employees who have access to the complete stock of ingredients in their kitchen, i.e., they have all the ingredients available to prepare any meal. This can be useful in identifying employees who have the necessary resources to create new or complex dishes.

Result



```
SELECT e.E_FirstName, e.E_LastName, k.K_Name, COUNT(*) AS Num_Ingredients_In_Stock
FROM Employee e
JOIN Work w ON e.E_Id = w.E_Id
JOIN Kitchen k ON w.K_Name = k.K_Name
JOIN Stock s ON k.K_Name = s.K_Name
JOIN (
    SELECT K_Name, COUNT(*) AS Num_Ingredients
    FROM Stock
    GROUP BY K_Name
) st ON k.K_Name = st.K_Name
GROUP BY e.E_FirstName, e.E_LastName, k.K_Name, st.Num_Ingredients
HAVING COUNT(*) = st.Num_Ingredients;
```



	E_FIRSTNAME	E_LASTNAME	K_NAME	NUM_INGREDIENTS_IN_STOCK
7	Gene	Harmon	Southern-style kitchen	2
8	Ricky	Hanley	Spanish modern kitchen	4
9	Taylor	DiBiasio	English country-style kitchen	1
10	Ty	LaPaglia	Industrial beach house kitchen	1
11	Alice	Williams	Kebab House	3
12	Cheech	Lovitz	Laotian larb restaurant	1
13	Johnny	Reilly	Lebanese restaurant	1
14	Joely	Downey	Luxury modern kitchen	1
15	Illeana	Aiken	Minimalist kitchen	2
16	Armand	Farina	Thai green curry restaurant	2
17	Clint	Piven	Thai street food stall	3
18	Jerry	Feore	The River Caf?	2
19	Machine	Hawthorne	The Savory Pantry	1
20	Holland	Lonsdale	The Savory Pantry	1
21	Denny	English	The Sunny Kitchen	1

13:5 0:01 rchiche@labdbwin 297 rows selected in 1.735 seconds

5-Query

Finding the kitchens where the total calorie count of all meals served is above the average calorie count of all meals.

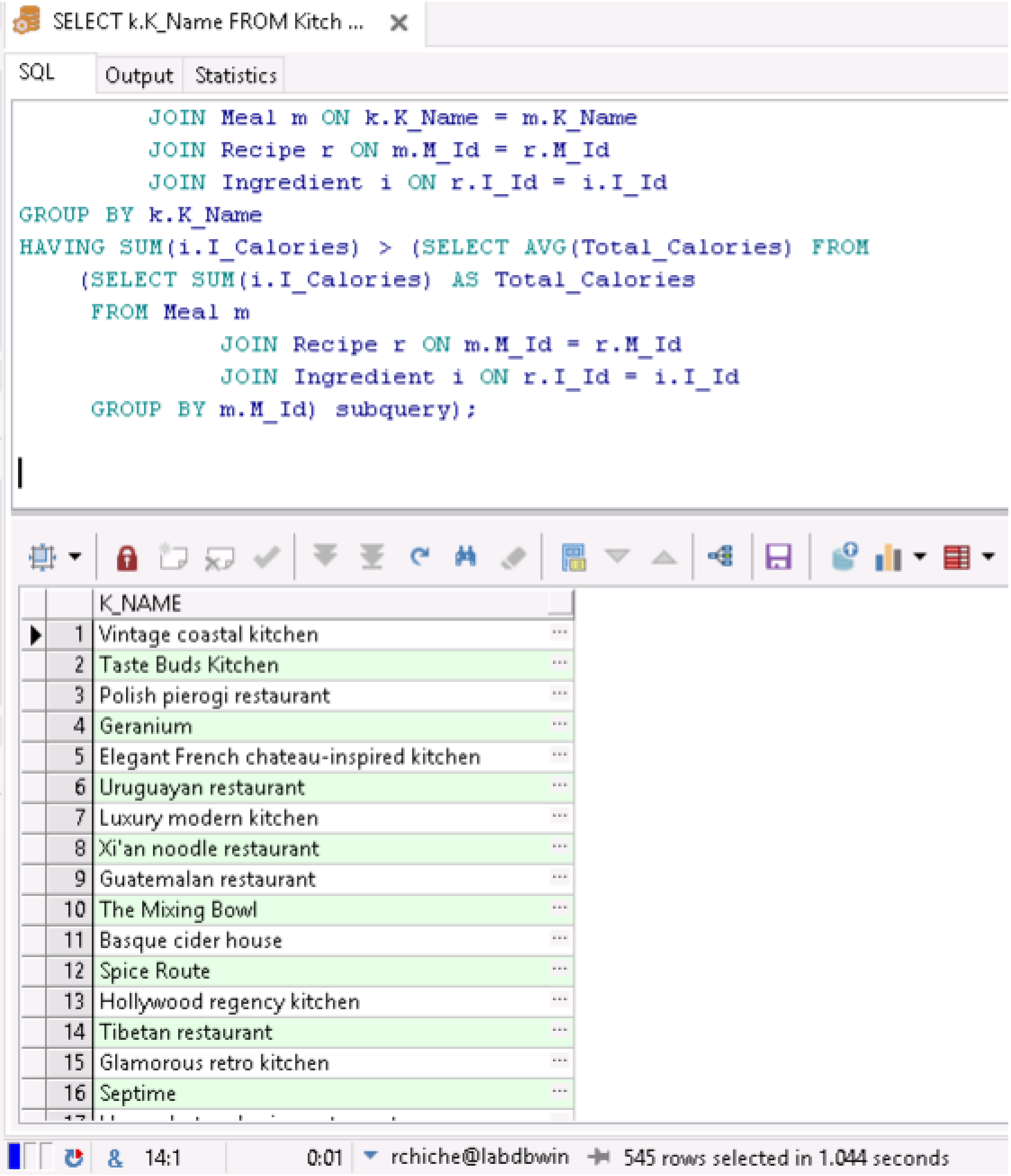
SQL Code

```sql
SELECT k.K_Name
FROM Kitchen k
        JOIN Meal m ON k.K_Name = m.K_Name
        JOIN Recipe r ON m.M_Id = r.M_Id
        JOIN Ingredient i ON r.I_Id = i.I_Id
GROUP BY k.K_Name
HAVING SUM(i.I_Calories) > (SELECT AVG(Total_Calories) FROM
   (SELECT SUM(i.I_Calories) AS Total_Calories
    FROM Meal m
             JOIN Recipe r ON m.M_Id = r.M_Id
             JOIN Ingredient i ON r.I_Id = i.I_Id
    GROUP BY m.M_Id) subquery);
```

Motivation

The motivation of this query is to find the kitchens that prepare meals with a total calorie count that is higher than the average calorie count of all meals. This can be useful for identifying kitchens that may need to adjust their menus to offer healthier options or for analyzing trends in the types of meals being served in different kitchens.

Result



```
         JOIN Meal m ON k.K_Name = m.K_Name
         JOIN Recipe r ON m.M_Id = r.M_Id
         JOIN Ingredient i ON r.I_Id = i.I_Id
GROUP BY k.K_Name
HAVING SUM(i.I_Calories) > (SELECT AVG(Total_Calories) FROM
    (SELECT SUM(i.I_Calories) AS Total_Calories
     FROM Meal m
              JOIN Recipe r ON m.M_Id = r.M_Id
              JOIN Ingredient i ON r.I_Id = i.I_Id
     GROUP BY m.M_Id) subquery);
```

	K_NAME
1	Vintage coastal kitchen
2	Taste Buds Kitchen
3	Polish pierogi restaurant
4	Geranium
5	Elegant French chateau-inspired kitchen
6	Uruguayan restaurant
7	Luxury modern kitchen
8	Xi'an noodle restaurant
9	Guatemalan restaurant
10	The Mixing Bowl
11	Basque cider house
12	Spice Route
13	Hollywood regency kitchen
14	Tibetan restaurant
15	Glamorous retro kitchen
16	Septime

14:1 0:01 rchiche@labdbwin 545 rows selected in 1.044 seconds

6-Query

Finding the name and total sales of the employee who sold the most expensive meal in each kitchen.

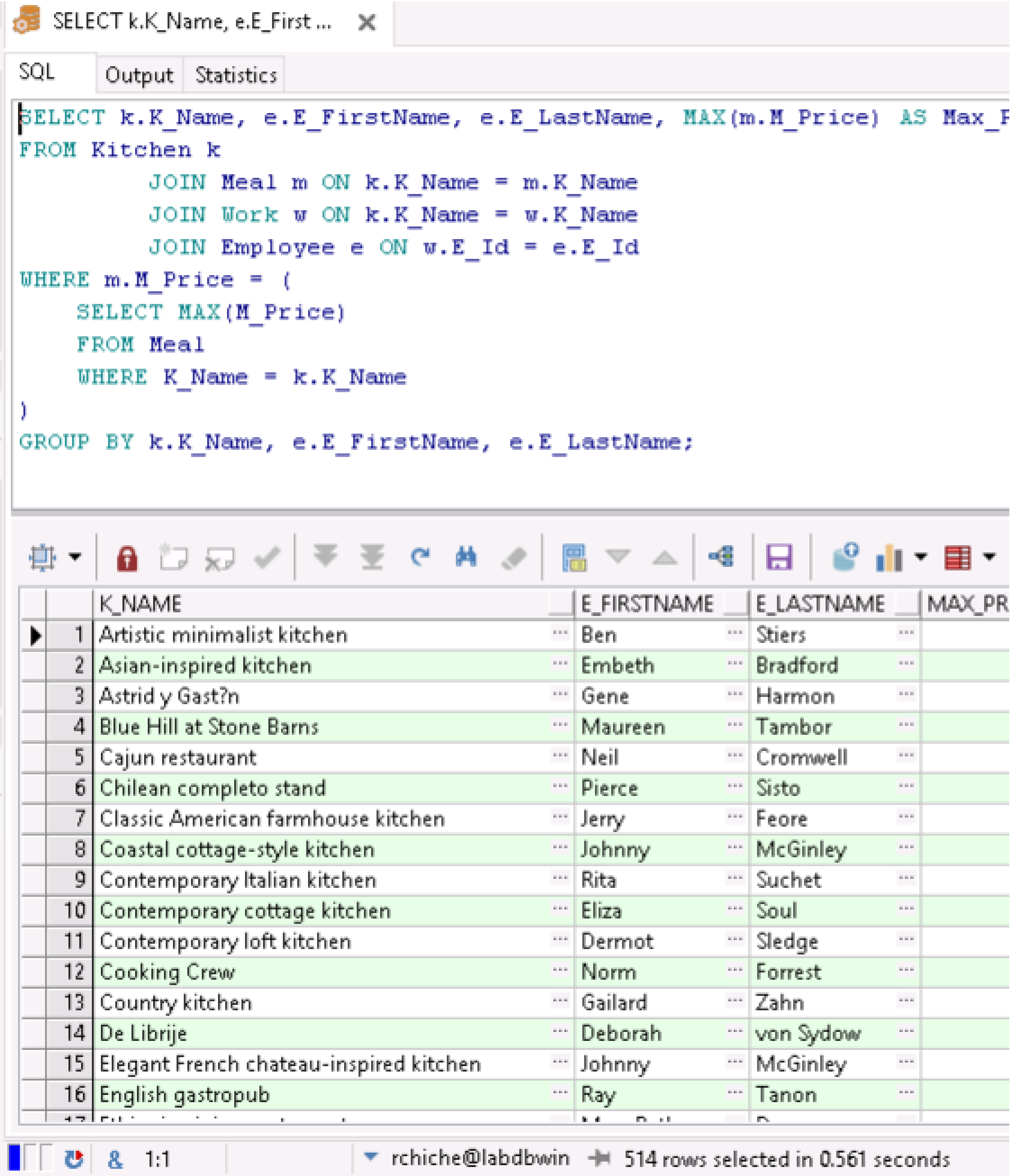
SQL Code

```
SELECT k.K_Name, e.E_FirstName, e.E_LastName, MAX(m.M_Price) AS Max_Price
FROM Kitchen k
        JOIN Meal m ON k.K_Name = m.K_Name
        JOIN Work w ON k.K_Name = w.K_Name
        JOIN Employee e ON w.E_Id = e.E_Id
WHERE m.M_Price = (
   SELECT MAX(M_Price)
   FROM Meal
   WHERE K_Name = k.K_Name
)
GROUP BY k.K_Name, e.E_FirstName, e.E_LastName;
```

Motivation

The motivation of this query is to find the employee who sold the most expensive meal in each kitchen, along with the name of the kitchen and the total sales. This information can be used to identify the top-performing employees and kitchens, as well as to optimize sales strategies and reward high-performing employees.

Result



```
SELECT k.K_Name, e.E_FirstName, e.E_LastName, MAX(m.M_Price) AS Max_P
FROM Kitchen k
         JOIN Meal m ON k.K_Name = m.K_Name
         JOIN Work w ON k.K_Name = w.K_Name
         JOIN Employee e ON w.E_Id = e.E_Id
WHERE m.M_Price = (
    SELECT MAX(M_Price)
    FROM Meal
    WHERE K_Name = k.K_Name
)
GROUP BY k.K_Name, e.E_FirstName, e.E_LastName;
```

	K_NAME	E_FIRSTNAME	E_LASTNAME	MAX_PR
1	Artistic minimalist kitchen	Ben	Stiers	
2	Asian-inspired kitchen	Embeth	Bradford	
3	Astrid y Gast?n	Gene	Harmon	
4	Blue Hill at Stone Barns	Maureen	Tambor	
5	Cajun restaurant	Neil	Cromwell	
6	Chilean completo stand	Pierce	Sisto	
7	Classic American farmhouse kitchen	Jerry	Feore	
8	Coastal cottage-style kitchen	Johnny	McGinley	
9	Contemporary Italian kitchen	Rita	Suchet	
10	Contemporary cottage kitchen	Eliza	Soul	
11	Contemporary loft kitchen	Dermot	Sledge	
12	Cooking Crew	Norm	Forrest	
13	Country kitchen	Gailard	Zahn	
14	De Librije	Deborah	von Sydow	
15	Elegant French chateau-inspired kitchen	Johnny	McGinley	
16	English gastropub	Ray	Tanon	

rchiche@labdbwin 514 rows selected in 0.561 seconds

7-Query

finding the names of the kitchens that prepare at least one meal that contains at least 1/2 of the available ingredients in the stock.

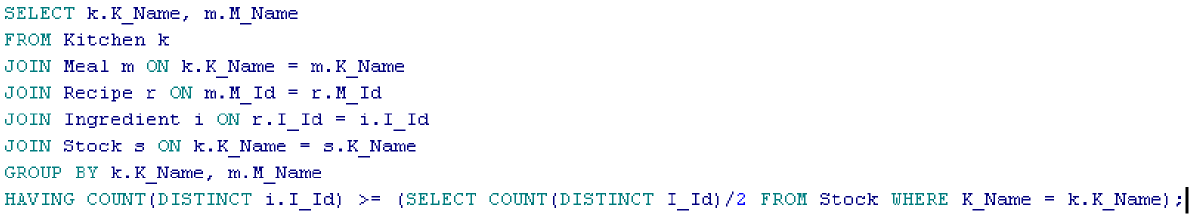
SQL Code

```sql
SELECT k.K_Name, m.M_Name

FROM Kitchen k

        JOIN Meal m ON k.K_Name = m.K_Name

        JOIN Recipe r ON m.M_Id = r.M_Id

        JOIN Ingredient i ON r.I_Id = i.I_Id

        JOIN Stock s ON k.K_Name = s.K_Name

GROUP BY k.K_Name, m.M_Name

HAVING COUNT(DISTINCT i.I_Id) >= (SELECT COUNT(DISTINCT I_Id) / 2 FROM Stock
WHERE K_Name = k.K_Name);
```

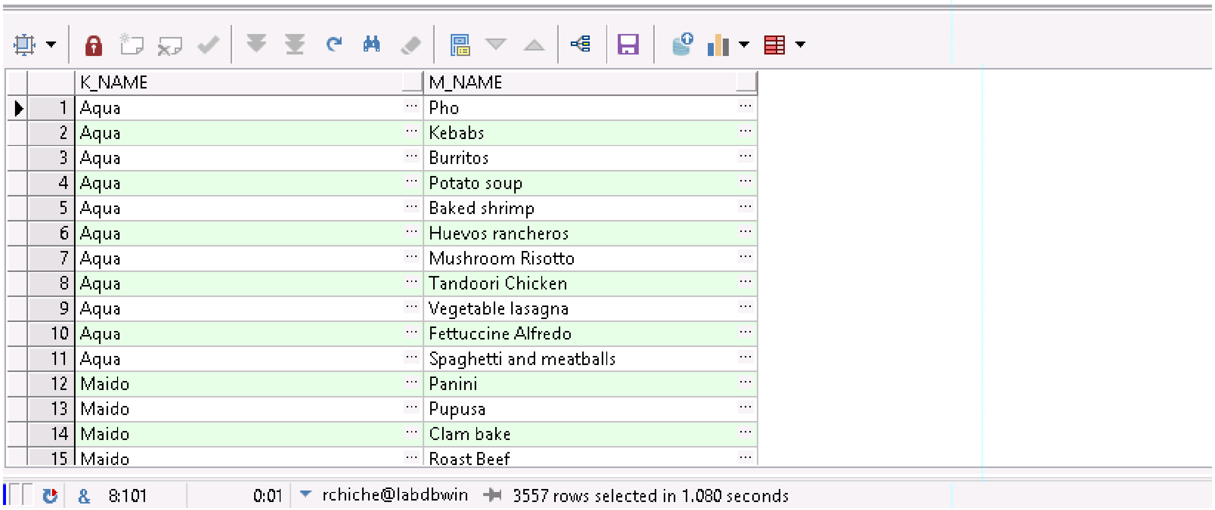
Motivation

The motivation of this query is to find the kitchens that are well-stocked and can prepare a variety of meals without running out of ingredients. By identifying the kitchens that can prepare meals with at least 1/2 of the available ingredients in stock, we can determine which kitchens are most efficient and effective in their ingredient management, and potentially identify areas for improvement in kitchens that do not meet this criteria. The addition of the meal name in the select allows for easier identification of which meals meet this criteria and can aid in making decisions about menu planning and ingredient purchasing.

Result



```
SELECT k.K_Name, m.M_Name
FROM Kitchen k
JOIN Meal m ON k.K_Name = m.K_Name
JOIN Recipe r ON m.M_Id = r.M_Id
JOIN Ingredient i ON r.I_Id = i.I_Id
JOIN Stock s ON k.K_Name = s.K_Name
GROUP BY k.K_Name, m.M_Name
HAVING COUNT(DISTINCT i.I_Id) >= (SELECT COUNT(DISTINCT I_Id)/2 FROM Stock WHERE K_Name = k.K_Name);
```



	K_NAME	M_NAME
1	Aqua	Pho
2	Aqua	Kebabs
3	Aqua	Burritos
4	Aqua	Potato soup
5	Aqua	Baked shrimp
6	Aqua	Huevos rancheros
7	Aqua	Mushroom Risotto
8	Aqua	Tandoori Chicken
9	Aqua	Vegetable lasagna
10	Aqua	Fettuccine Alfredo
11	Aqua	Spaghetti and meatballs
12	Maido	Panini
13	Maido	Pupusa
14	Maido	Clam bake
15	Maido	Roast Beef

8:101 0:01 rchiche@labdbwin 3557 rows selected in 1.080 seconds

8-Query

Finding the name of each kitchen, the total number of meals served, and the total revenue generated by each kitchen between two dates.

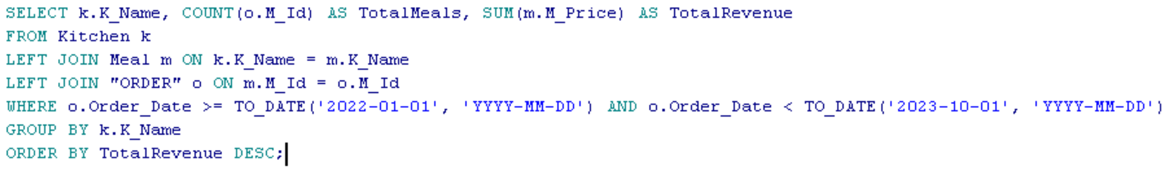
SQL Code

```
SELECT k.K_Name, COUNT(o.M_Id) AS TotalMeals, SUM(m.M_Price) AS
TotalRevenue

FROM Kitchen k

        LEFT JOIN Meal m ON k.K_Name = m.K_Name

        LEFT JOIN ”ORDER” o ON m.M_Id = o.M_Id

WHERE o.Order_Date >= TO_DATE('2022-01-01','YYYY-MM-DD') AND o.Order_Date <
TO_DATE('2023-10-01','YYYY-MM-DD')

GROUP BY k.K_Name

ORDER BY TotalRevenue DESC;
```

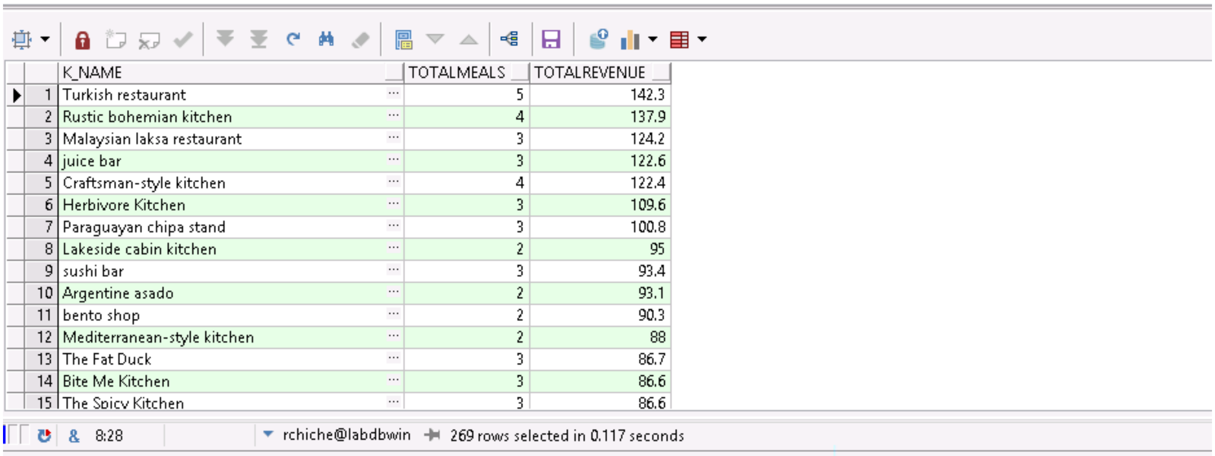
Motivation

The motivation behind the query is for analyzing the performance of different kitchens in terms of the number of meals sold and the total revenue generated over a specific period of time. By using this query, restaurant managers or owners can identify the most popular kitchens and meals, and make informed decisions about menu offerings, pricing strategies, and resource allocation. It can also help in identifying the areas where the restaurant is performing well and where it needs improvement.

Result



```
SELECT k.K_Name, COUNT(o.M_Id) AS TotalMeals, SUM(m.M_Price) AS TotalRevenue
FROM Kitchen k
LEFT JOIN Meal m ON k.K_Name = m.K_Name
LEFT JOIN "ORDER" o ON m.M_Id = o.M_Id
WHERE o.Order_Date >= TO_DATE('2022-01-01', 'YYYY-MM-DD') AND o.Order_Date < TO_DATE('2023-10-01', 'YYYY-MM-DD')
GROUP BY k.K_Name
ORDER BY TotalRevenue DESC;
```



	K_NAME	TOTALMEALS	TOTALREVENUE
1	Turkish restaurant	5	142.3
2	Rustic bohemian kitchen	4	137.9
3	Malaysian laksa restaurant	3	124.2
4	juice bar	3	122.6
5	Craftsman-style kitchen	4	122.4
6	Herbivore Kitchen	3	109.6
7	Paraguayan chipa stand	3	100.8
8	Lakeside cabin kitchen	2	95
9	sushi bar	3	93.4
10	Argentine asado	2	93.1
11	bento shop	2	90.3
12	Mediterranean-style kitchen	2	88
13	The Fat Duck	3	86.7
14	Bite Me Kitchen	3	86.6
15	The Spicy Kitchen	3	86.6

8:28 rchiche@labdbwin 269 rows selected in 0.117 seconds

Indexes

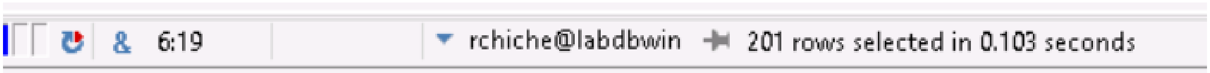
Speed improvement

The index of Query-2:

The index:

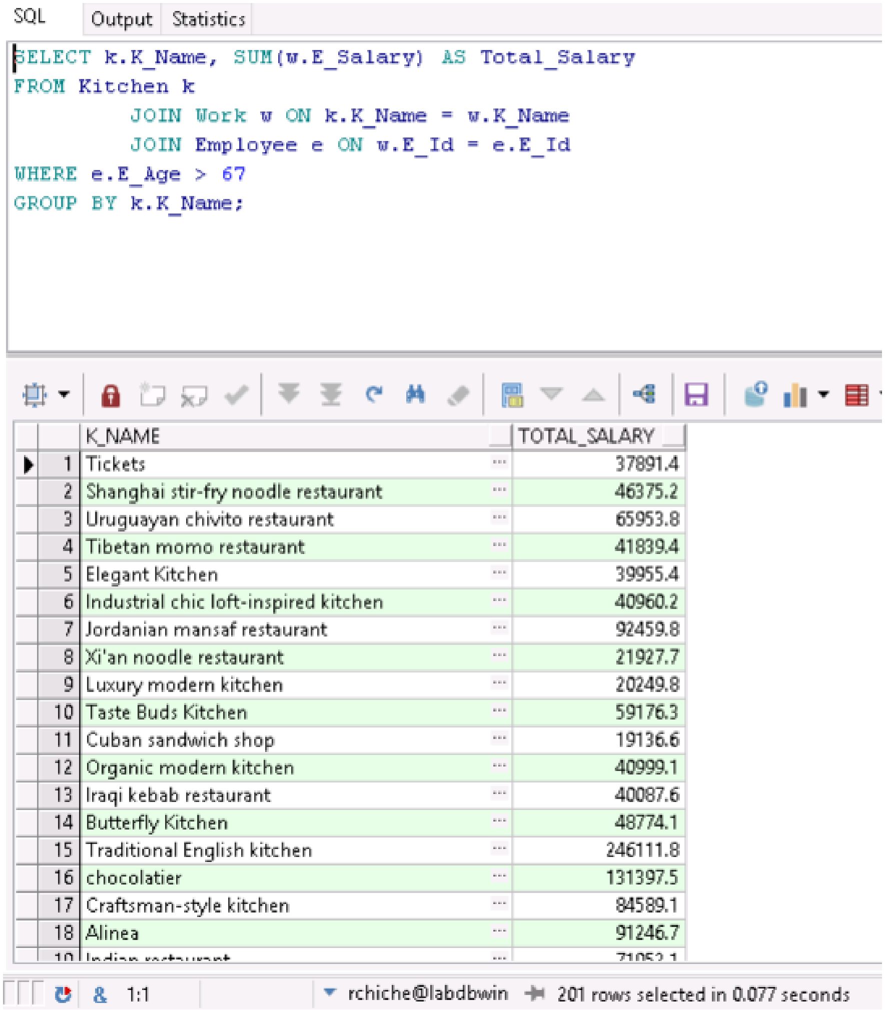
```
CREATE INDEX idx_employee_age ON Employee (E_Age);
```

Time before: 0.103 sec



6:19 rchiche@labdbwin 201 rows selected in 0.103 seconds

Time after: 0.077 sec



```
SELECT k.K_Name, SUM(w.E_Salary) AS Total_Salary
FROM Kitchen k
         JOIN Work w ON k.K_Name = w.K_Name
         JOIN Employee e ON w.E_Id = e.E_Id
WHERE e.E_Age > 67
GROUP BY k.K_Name;
```

	K_NAME	TOTAL_SALARY
1	Tickets	37891.4
2	Shanghai stir-fry noodle restaurant	46375.2
3	Uruguayan chivito restaurant	65953.8
4	Tibetan momo restaurant	41839.4
5	Elegant Kitchen	39955.4
6	Industrial chic loft-inspired kitchen	40960.2
7	Jordanian mansaf restaurant	92459.8
8	Xi'an noodle restaurant	21927.7
9	Luxury modern kitchen	20249.8
10	Taste Buds Kitchen	59176.3
11	Cuban sandwich shop	19136.6
12	Organic modern kitchen	40999.1
13	Iraqi kebab restaurant	40087.6
14	Butterfly Kitchen	48774.1
15	Traditional English kitchen	246111.8
16	chocolatier	131397.5
17	Craftsman-style kitchen	84589.1
18	Alinea	91246.7
19	Indian restaurant	71052.1

1:1 rchiche@labdbwin 201 rows selected in 0.077 seconds

Motivation and why this index improved the speed:

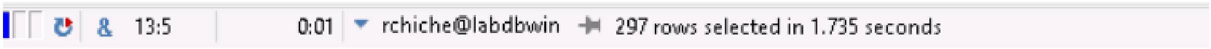
It allows the database engine to quickly locate the rows that satisfy the condition e.E_Age > 67 in the WHERE clause. Without an index on E_Age, the database engine would have to scan the entire Employee table and compare the value of E_Age for each row with the value of 67. With the index, the database engine can use an index seek operation to locate the rows that satisfy the WHERE condition efficiently, and then join the matching rows with the Work and Kitchen tables to compute the result set. This can significantly reduce the query's execution time, especially if the Employee table is large and the condition on E_Age is selective.

The index of Query-4:

The index:

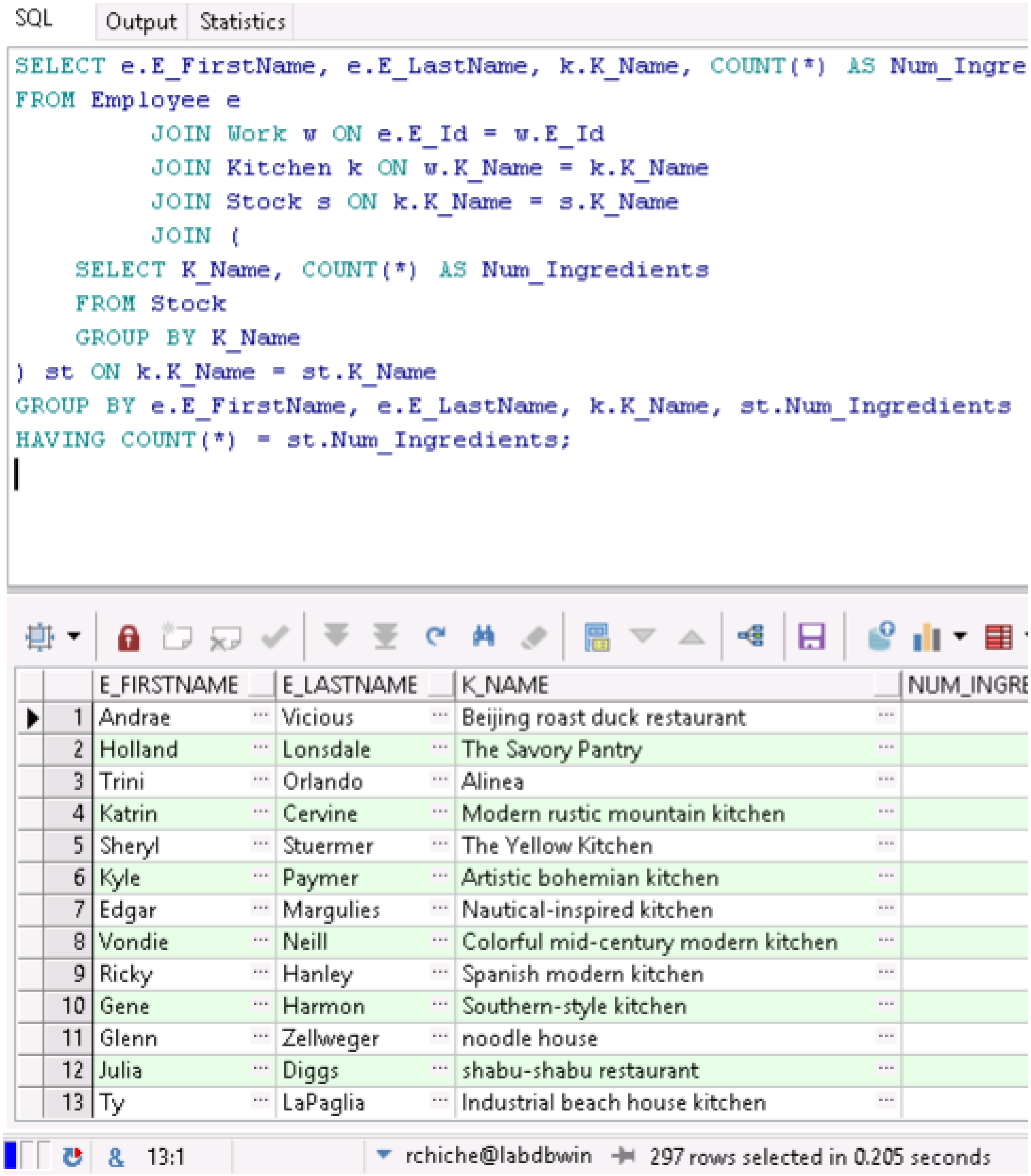
```
CREATE INDEX stock_kname_idx ON Stock (K_Name);
```

Time before: 1.735 sec



13:5 0:01 rchiche@labdbwin 297 rows selected in 1.735 seconds

Time after: 0.205 sec



```
SELECT e.E_FirstName, e.E_LastName, k.K_Name, COUNT(*) AS Num_Ingre
FROM Employee e
         JOIN Work w ON e.E_Id = w.E_Id
         JOIN Kitchen k ON w.K_Name = k.K_Name
         JOIN Stock s ON k.K_Name = s.K_Name
         JOIN (
    SELECT K_Name, COUNT(*) AS Num_Ingredients
    FROM Stock
    GROUP BY K_Name
) st ON k.K_Name = st.K_Name
GROUP BY e.E_FirstName, e.E_LastName, k.K_Name, st.Num_Ingredients
HAVING COUNT(*) = st.Num_Ingredients;
```

	E_FIRSTNAME	E_LASTNAME	K_NAME	NUM_INGRE
1	Andrae	Vicious	Beijing roast duck restaurant	
2	Holland	Lonsdale	The Savory Pantry	
3	Trini	Orlando	Alinea	
4	Katrin	Cervine	Modern rustic mountain kitchen	
5	Sheryl	Stuermer	The Yellow Kitchen	
6	Kyle	Paymer	Artistic bohemian kitchen	
7	Edgar	Margulies	Nautical-inspired kitchen	
8	Vondie	Neill	Colorful mid-century modern kitchen	
9	Ricky	Hanley	Spanish modern kitchen	
10	Gene	Harmon	Southern-style kitchen	
11	Glenn	Zellweger	noodle house	
12	Julia	Diggs	shabu-shabu restaurant	
13	Ty	LaPaglia	Industrial beach house kitchen	

13:1 rchiche@labdbwin 297 rows selected in 0.205 seconds

Motivation and why this index improved the speed:

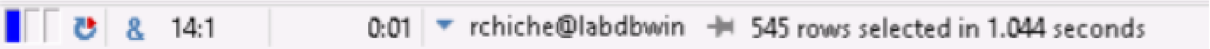
It allows faster joins between the Kitchen and Stock tables. Without the index, the database would have to perform a full table scan on the Stock table to find the matching rows, which could be slow for larger tables. By creating an index on the K_Name column, the database can quickly locate the relevant rows in the Stock table and perform the join operation more efficiently. This can significantly improve the overall performance of the query.

The index of Query-5:

The index:

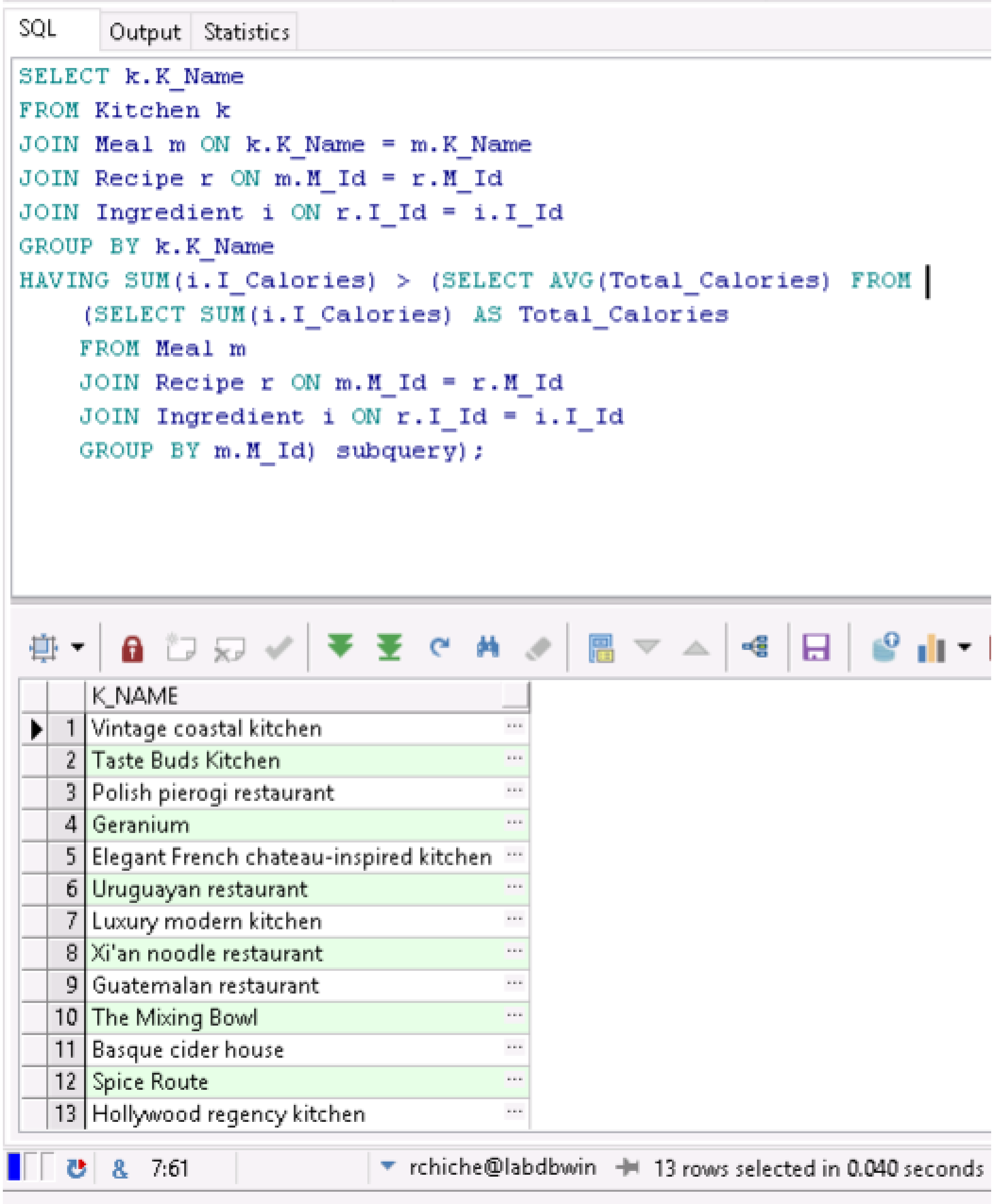
```
CREATE INDEX idx_work_salary ON Work (E_Salary);
```

Time before: 1.044 sec



```
14:1    0:01    rchiche@labdbwin    545 rows selected in 1.044 seconds
```

Time after: 0.04 sec



```
SELECT k.K_Name
FROM Kitchen k
JOIN Meal m ON k.K_Name = m.K_Name
JOIN Recipe r ON m.M_Id = r.M_Id
JOIN Ingredient i ON r.I_Id = i.I_Id
GROUP BY k.K_Name
HAVING SUM(i.I_Calories) > (SELECT AVG(Total_Calories) FROM
    (SELECT SUM(i.I_Calories) AS Total_Calories
    FROM Meal m
    JOIN Recipe r ON m.M_Id = r.M_Id
    JOIN Ingredient i ON r.I_Id = i.I_Id
    GROUP BY m.M_Id) subquery);
```

	K_NAME
1	Vintage coastal kitchen
2	Taste Buds Kitchen
3	Polish pierogi restaurant
4	Geranium
5	Elegant French chateau-inspired kitchen
6	Uruguayan restaurant
7	Luxury modern kitchen
8	Xi'an noodle restaurant
9	Guatemalan restaurant
10	The Mixing Bowl
11	Basque cider house
12	Spice Route
13	Hollywood regency kitchen

```
7:61    rchiche@labdbwin    13 rows selected in 0.040 seconds
```

Motivation and why this index improved the speed:

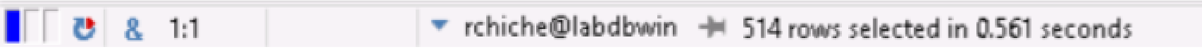
By creating an index on the E_Salary column, the database engine can more efficiently filter and retrieve the relevant rows for the join operations, which can reduce the query's overall execution time. Additionally, if the table is very large and the index is clustered, it can help the database engine retrieve the data in a more organized way and reduce the need for expensive disk I/O operations.

The index of Query-6:

The index:

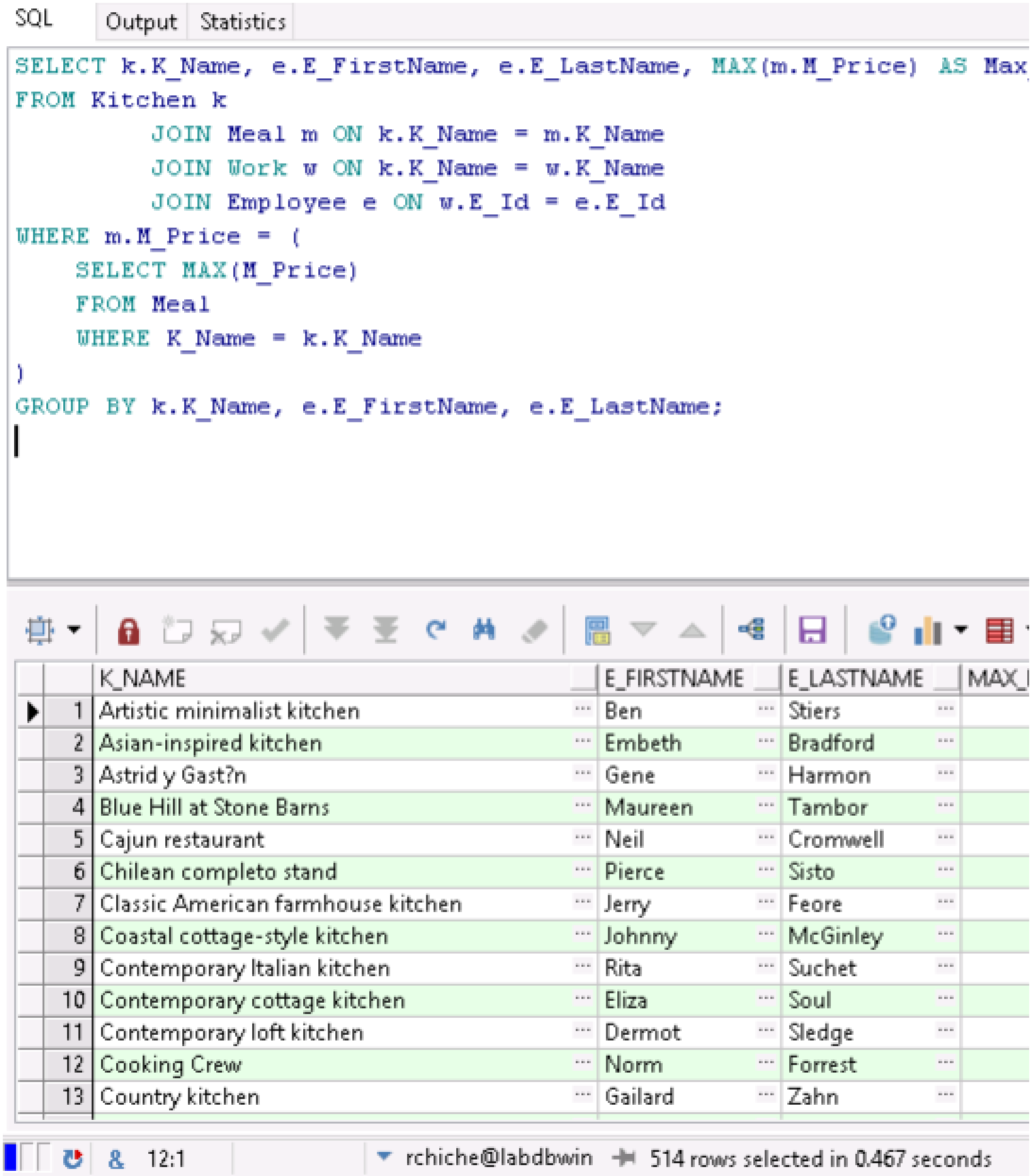
```
CREATE INDEX meal_kname_price_idx ON Meal (K_Name, M_Price);
```

Time before: 0.561 sec



1:1 rchiche@labdbwin 514 rows selected in 0.561 seconds

Time after: 0.467 sec



```
SELECT k.K_Name, e.E_FirstName, e.E_LastName, MAX(m.M_Price) AS Max
FROM Kitchen k
         JOIN Meal m ON k.K_Name = m.K_Name
         JOIN Work w ON k.K_Name = w.K_Name
         JOIN Employee e ON w.E_Id = e.E_Id
WHERE m.M_Price = (
    SELECT MAX(M_Price)
    FROM Meal
    WHERE K_Name = k.K_Name
)
GROUP BY k.K_Name, e.E_FirstName, e.E_LastName;
```

	K_NAME	E_FIRSTNAME	E_LASTNAME	MAX_
1	Artistic minimalist kitchen	Ben	Stiers	
2	Asian-inspired kitchen	Embeth	Bradford	
3	Astrid y Gast?n	Gene	Harmon	
4	Blue Hill at Stone Barns	Maureen	Tambor	
5	Cajun restaurant	Neil	Cromwell	
6	Chilean completo stand	Pierce	Sisto	
7	Classic American farmhouse kitchen	Jerry	Feore	
8	Coastal cottage-style kitchen	Johnny	McGinley	
9	Contemporary Italian kitchen	Rita	Suchet	
10	Contemporary cottage kitchen	Eliza	Soul	
11	Contemporary loft kitchen	Dermot	Sledge	
12	Cooking Crew	Norm	Forrest	
13	Country kitchen	Gailard	Zahn	

12:1 rchiche@labdbwin 514 rows selected in 0.467 seconds

Motivation and why this index improved the speed:

By creating an index on K_Name and M_Price columns, the database can quickly locate the rows that satisfy both the join and the filter conditions. This can reduce the number of rows that need to be scanned and improve the query's overall performance.

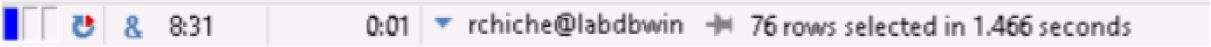
Speed degradation/No improvement

The index of Query-1:

The index:

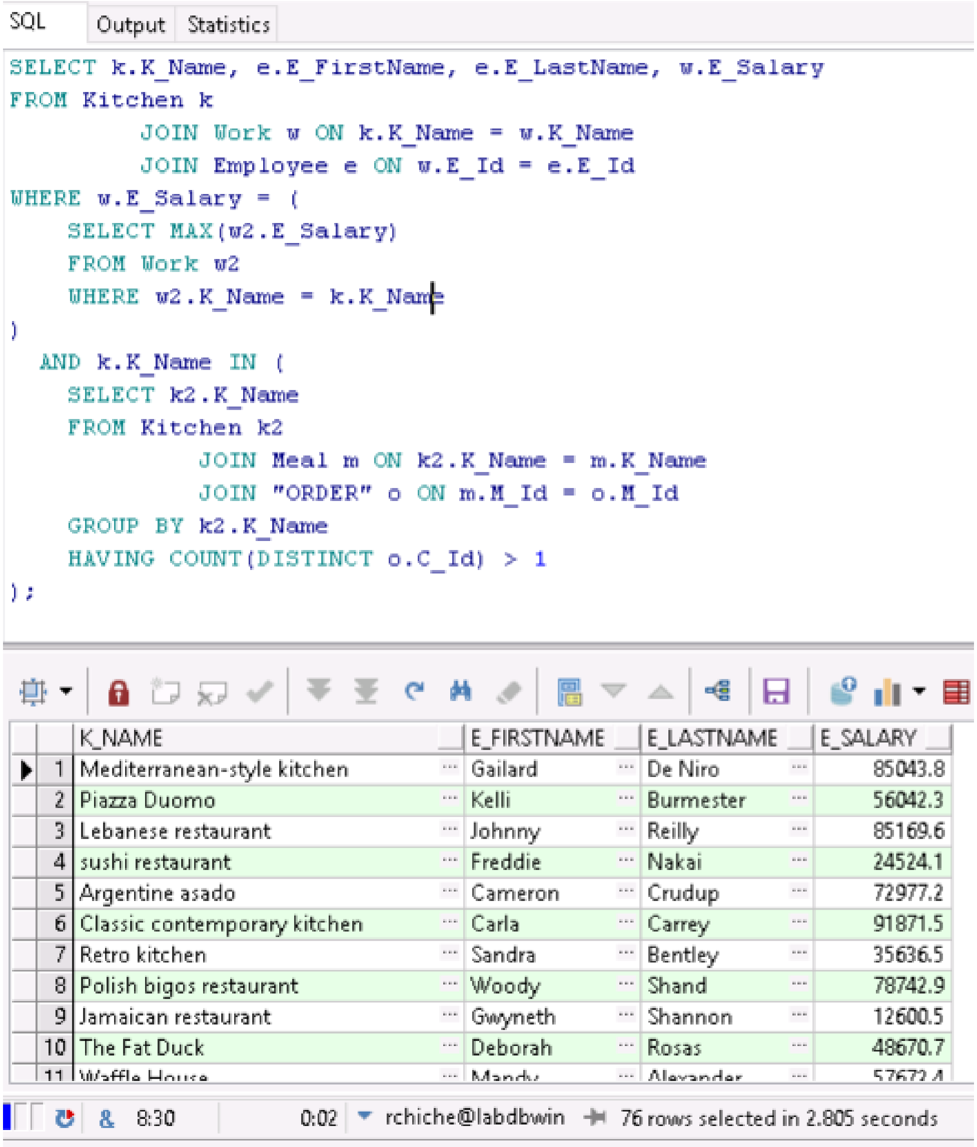
```
CREATE INDEX idx_k_type on Kitchen(k_Type);
```

Time before: 1.466 sec



```
8:31    0:01    rchiche@labdbwin    76 rows selected in 1.466 seconds
```

Time after: 2.805 sec



```
SELECT k.K_Name, e.E_FirstName, e.E_LastName, w.E_Salary
FROM Kitchen k
         JOIN Work w ON k.K_Name = w.K_Name
         JOIN Employee e ON w.E_Id = e.E_Id
WHERE w.E_Salary = (
    SELECT MAX(w2.E_Salary)
    FROM Work w2
    WHERE w2.K_Name = k.K_Name
)
  AND k.K_Name IN (
    SELECT k2.K_Name
    FROM Kitchen k2
             JOIN Meal m ON k2.K_Name = m.K_Name
             JOIN "ORDER" o ON m.M_Id = o.M_Id
    GROUP BY k2.K_Name
    HAVING COUNT(DISTINCT o.C_Id) > 1
);
```

	K_NAME	E_FIRSTNAME	E_LASTNAME	E_SALARY
1	Mediterranean-style kitchen	Gailard	De Niro	85043.8
2	Piazza Duomo	Kelli	Burmester	56042.3
3	Lebanese restaurant	Johnny	Reilly	85169.6
4	sushi restaurant	Freddie	Nakai	24524.1
5	Argentine asado	Cameron	Crudup	72977.2
6	Classic contemporary kitchen	Carla	Carrey	91871.5
7	Retro kitchen	Sandra	Bentley	35636.5
8	Polish bigos restaurant	Woody	Shand	78742.9
9	Jamaican restaurant	Gwyneth	Shannon	12600.5
10	The Fat Duck	Deborah	Rosas	48670.7
11	Waffle House	Mandy	Alexander	57672.4

```
8:30    0:02    rchiche@labdbwin    76 rows selected in 2.805 seconds
```

Motivation and why this index didn’t improve the speed:

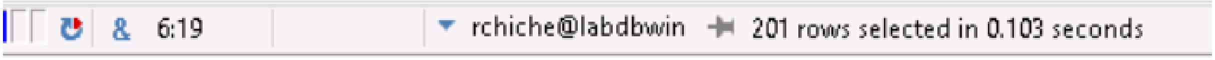
The index on k_Type is not helpful for this query because the query does not include any conditions on the k_Type column. Therefore, the database engine does not need to use the index to filter or sort the results. In general, indexes are most effective when they are used to filter or sort the data based on the columns included in the index.

The index of Query-2:

The index:

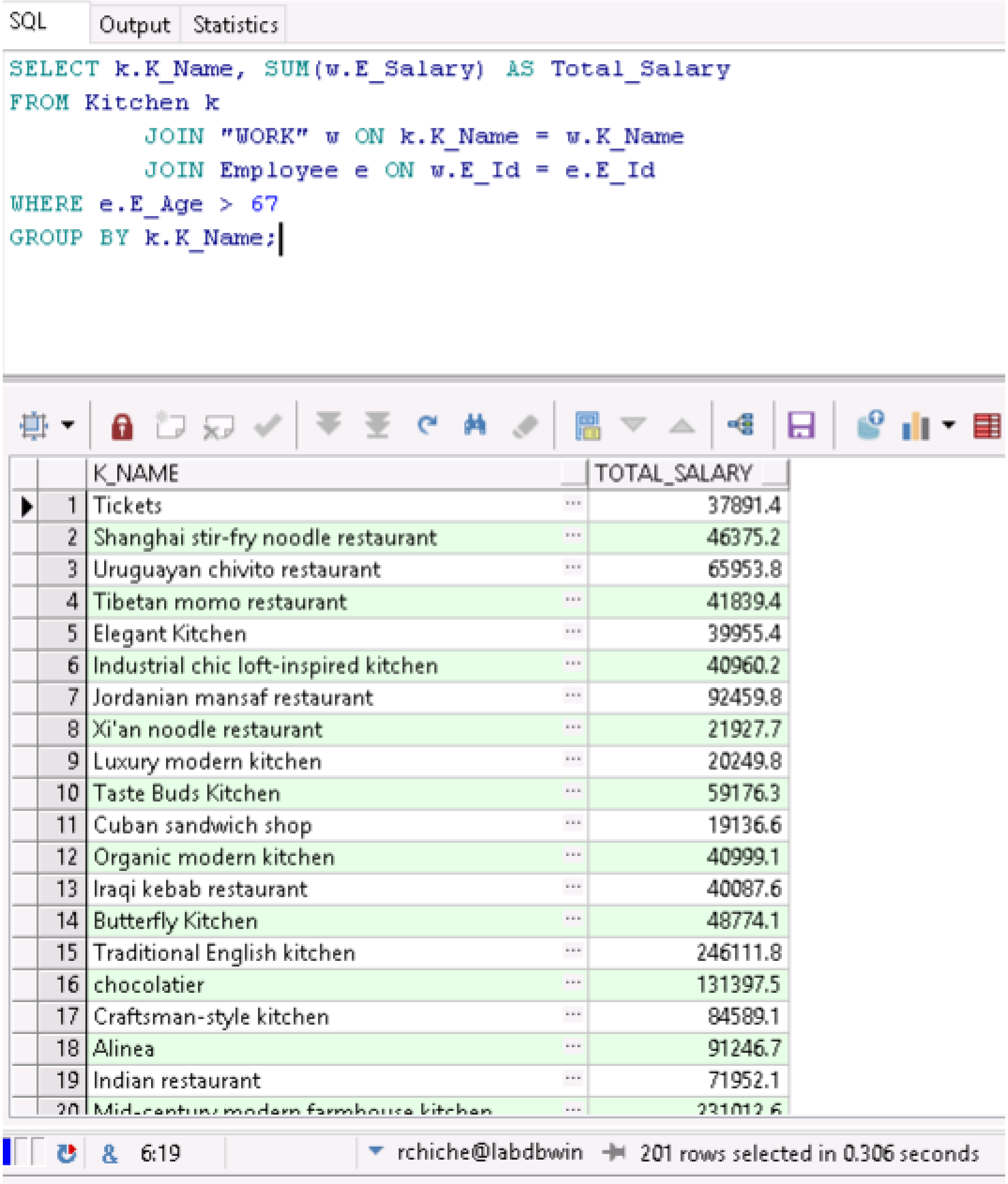
```
CREATE INDEX idx_e_pn ON Employee (E_telephone);
```

Time before: 0.103 sec



6:19 rchiche@labdbwin 201 rows selected in 0.103 seconds

Time after: 0.306 sec



```
SELECT k.K_Name, SUM(w.E_Salary) AS Total_Salary
FROM Kitchen k
         JOIN "WORK" w ON k.K_Name = w.K_Name
         JOIN Employee e ON w.E_Id = e.E_Id
WHERE e.E_Age > 67
GROUP BY k.K_Name;
```

	K_NAME	TOTAL_SALARY
1	Tickets	37891.4
2	Shanghai stir-fry noodle restaurant	46375.2
3	Uruguayan chivito restaurant	65953.8
4	Tibetan momo restaurant	41839.4
5	Elegant Kitchen	39955.4
6	Industrial chic loft-inspired kitchen	40960.2
7	Jordanian mansaf restaurant	92459.8
8	Xi'an noodle restaurant	21927.7
9	Luxury modern kitchen	20249.8
10	Taste Buds Kitchen	59176.3
11	Cuban sandwich shop	19136.6
12	Organic modern kitchen	40999.1
13	Iraqi kebab restaurant	40087.6
14	Butterfly Kitchen	48774.1
15	Traditional English kitchen	246111.8
16	chocolatier	131397.5
17	Craftsman-style kitchen	84589.1
18	Alinea	91246.7
19	Indian restaurant	71952.1
20	Mid-century modern farmhouse kitchen	231012.6

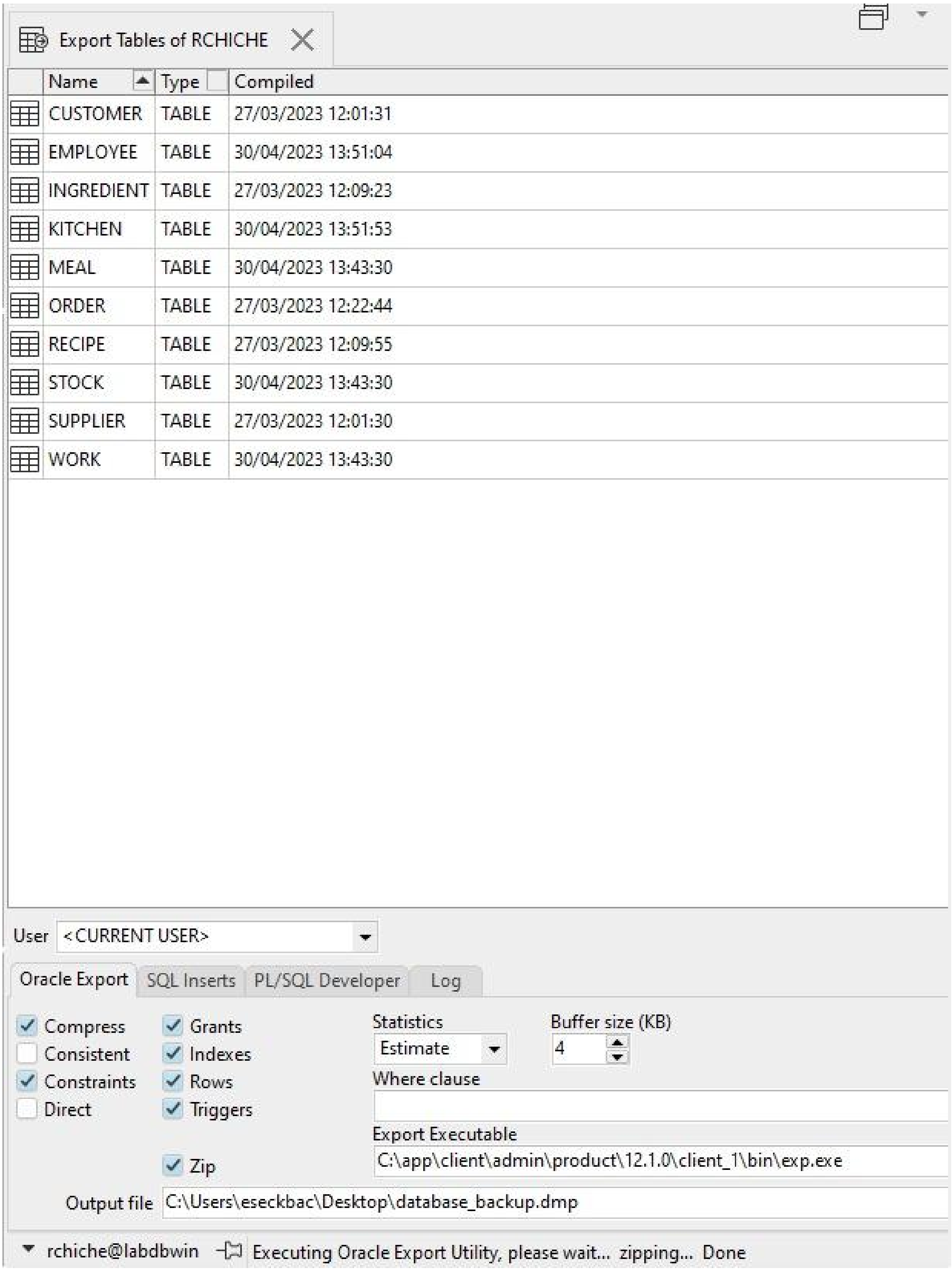
6:19 rchiche@labdbwin 201 rows selected in 0.306 seconds

Motivation and why this index didn’t improve the speed:

The index idx_e_pn on the Employee table for the column E_telephone is not helpful for this query because the E_telephone column is not used in the query at all. The query only involves joining Kitchen, Work, and Employee tables and filtering based on the E_Age column. Therefore, an index on E_telephone will not provide any performance benefits for this query.

Database backup

We are making the backup on these tables:



Export Tables of RCHICHE

Name	Type	Compiled
CUSTOMER	TABLE	27/03/2023 12:01:31
EMPLOYEE	TABLE	30/04/2023 13:51:04
INGREDIENT	TABLE	27/03/2023 12:09:23
KITCHEN	TABLE	30/04/2023 13:51:53
MEAL	TABLE	30/04/2023 13:43:30
ORDER	TABLE	27/03/2023 12:22:44
RECIPE	TABLE	27/03/2023 12:09:55
STOCK	TABLE	30/04/2023 13:43:30
SUPPLIER	TABLE	27/03/2023 12:01:30
WORK	TABLE	30/04/2023 13:43:30

User <CURRENT USER>

☑ Compress
☐ Consistent
☑ Constraints
☐ Direct
☑ Grants
☑ Indexes
☑ Rows
☑ Triggers
☑ Zip

Statistics: Estimate

Buffer size (KB): 4

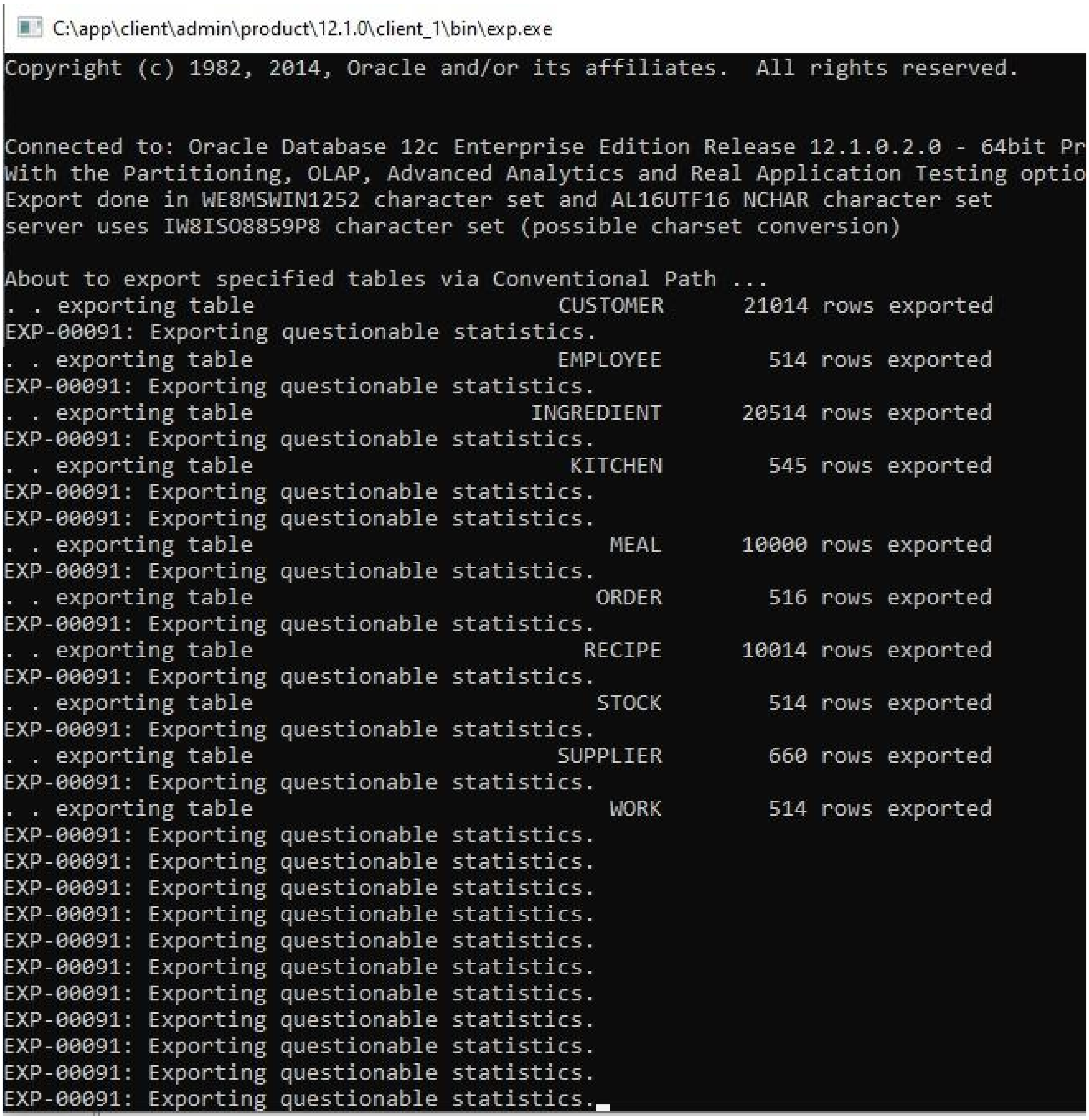
Where clause:

Export Executable: C:\app\client\admin\product\12.1.0\client_1\bin\exp.exe

Output file: C:\Users\eseckbac\Desktop\database_backup.dmp

rchiche@labdbwin   Executing Oracle Export Utility, please wait... zipping... Done

The process of making the backup:



C:\app\client\admin\product\12.1.0\client_1\bin\exp.exe

```
Copyright (c) 1982, 2014, Oracle and/or its affiliates.  All rights reserved.


Connected to: Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Pr
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing optio
Export done in WE8MSWIN1252 character set and AL16UTF16 NCHAR character set
server uses IW8ISO8859P8 character set (possible charset conversion)

About to export specified tables via Conventional Path ...
. . exporting table                       CUSTOMER      21014 rows exported
EXP-00091: Exporting questionable statistics.
. . exporting table                       EMPLOYEE        514 rows exported
EXP-00091: Exporting questionable statistics.
. . exporting table                     INGREDIENT      20514 rows exported
EXP-00091: Exporting questionable statistics.
. . exporting table                        KITCHEN        545 rows exported
EXP-00091: Exporting questionable statistics.
EXP-00091: Exporting questionable statistics.
. . exporting table                           MEAL      10000 rows exported
EXP-00091: Exporting questionable statistics.
. . exporting table                          ORDER        516 rows exported
EXP-00091: Exporting questionable statistics.
. . exporting table                         RECIPE      10014 rows exported
EXP-00091: Exporting questionable statistics.
. . exporting table                          STOCK        514 rows exported
EXP-00091: Exporting questionable statistics.
. . exporting table                       SUPPLIER        660 rows exported
EXP-00091: Exporting questionable statistics.
. . exporting table                           WORK        514 rows exported
EXP-00091: Exporting questionable statistics.
EXP-00091: Exporting questionable statistics.
EXP-00091: Exporting questionable statistics.
EXP-00091: Exporting questionable statistics.
EXP-00091: Exporting questionable statistics.
EXP-00091: Exporting questionable statistics.
EXP-00091: Exporting questionable statistics.
EXP-00091: Exporting questionable statistics.
EXP-00091: Exporting questionable statistics.
EXP-00091: Exporting questionable statistics.
EXP-00091: Exporting questionable statistics.
```

Github

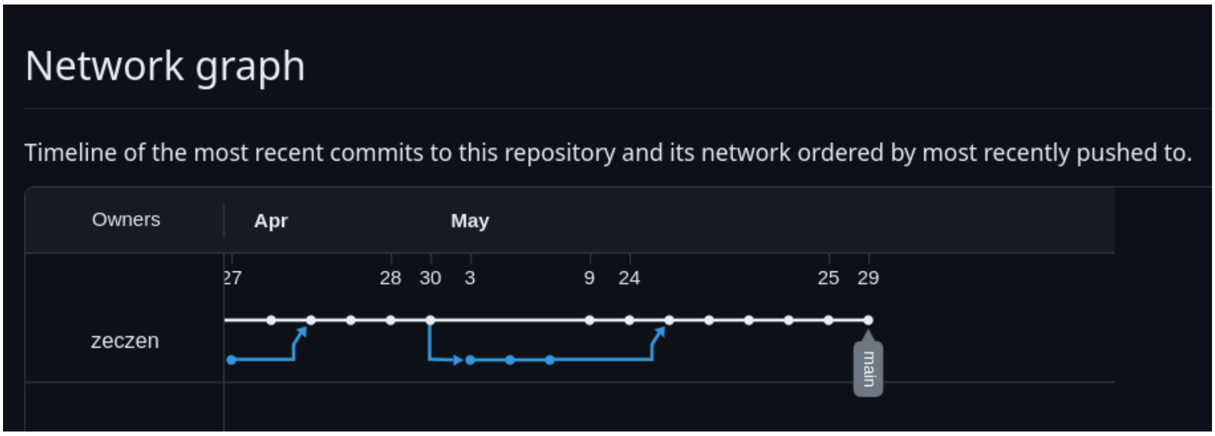
Using Git to keep track of queries and data in the database is important for several reasons.

Firstly, it allows for version control, enabling teams to track and manage changes made to the database over time, ensuring transparency and accountability.

Secondly, it provides the ability to revert to previous versions of queries or data, allowing for easy rollbacks in case of errors or issues.

Thirdly, Git facilitates collaboration among team members by enabling them to work concurrently on different branches and merge changes seamlessly.

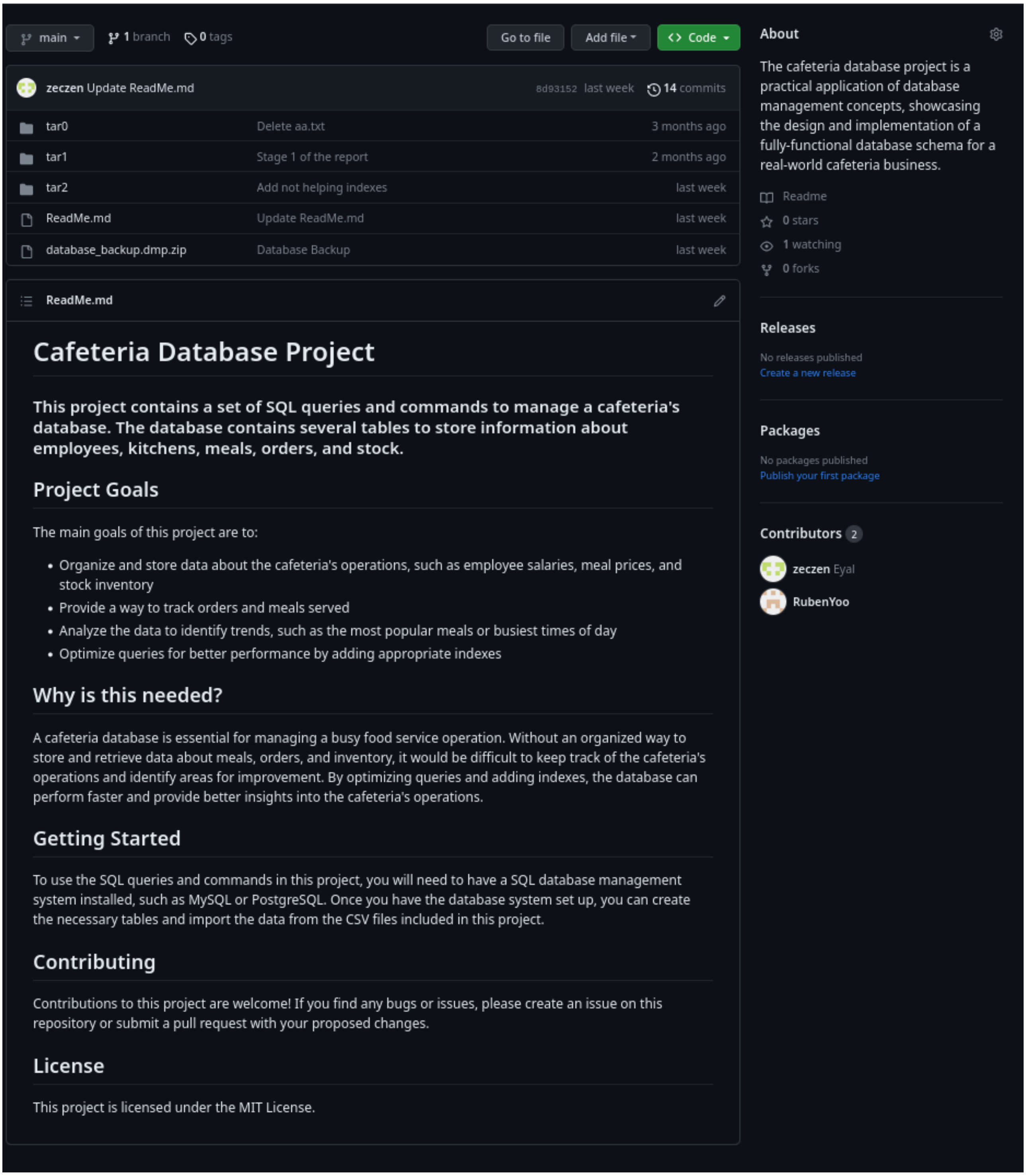
Lastly, it serves as a documentation tool, capturing the history of queries and data modifications, making it easier to understand the evolution of the database and troubleshoot any problems that may arise.



Network graph

Timeline of the most recent commits to this repository and its network ordered by most recently pushed to.

We extensively utilized GitHub throughout our project to achieve our goals, so we also included the backup file in our GitHub repository.



zeczen Update ReadMe.md 8d93152 last week 14 commits

tar0	Delete aa.txt	3 months ago
tar1	Stage 1 of the report	2 months ago
tar2	Add not helping indexes	last week
ReadMe.md	Update ReadMe.md	last week
database_backup.dmp.zip	Database Backup	last week

ReadMe.md

Cafeteria Database Project

This project contains a set of SQL queries and commands to manage a cafeteria's database. The database contains several tables to store information about employees, kitchens, meals, orders, and stock.

Project Goals

The main goals of this project are to:

- Organize and store data about the cafeteria's operations, such as employee salaries, meal prices, and stock inventory
- Provide a way to track orders and meals served
- Analyze the data to identify trends, such as the most popular meals or busiest times of day
- Optimize queries for better performance by adding appropriate indexes

Why is this needed?

A cafeteria database is essential for managing a busy food service operation. Without an organized way to store and retrieve data about meals, orders, and inventory, it would be difficult to keep track of the cafeteria's operations and identify areas for improvement. By optimizing queries and adding indexes, the database can perform faster and provide better insights into the cafeteria's operations.

Getting Started

To use the SQL queries and commands in this project, you will need to have a SQL database management system installed, such as MySQL or PostgreSQL. Once you have the database system set up, you can create the necessary tables and import the data from the CSV files included in this project.

Contributing

Contributions to this project are welcome! If you find any bugs or issues, please create an issue on this repository or submit a pull request with your proposed changes.

License

This project is licensed under the MIT License.

About

The cafeteria database project is a practical application of database management concepts, showcasing the design and implementation of a fully-functional database schema for a real-world cafeteria business.

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Contributors 2

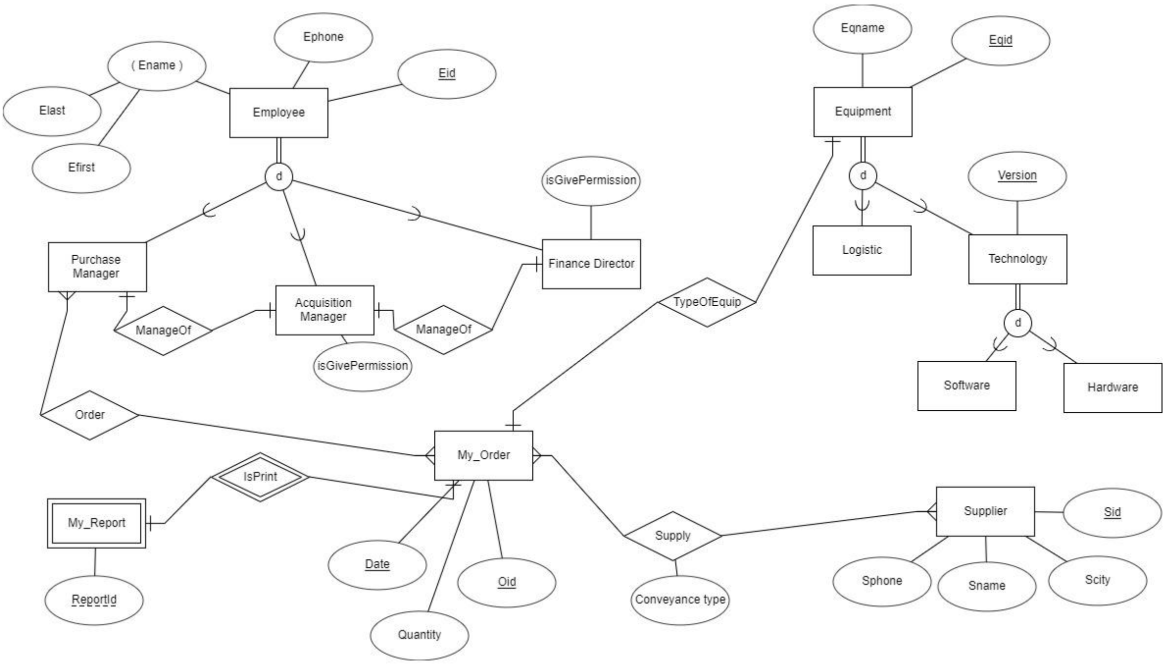
zeczen Eyal
RubenYoo

The link for the GitHub repository: github.com/zeczen/DatabaseProject

Integration - Three queries

In this phase of our project, we are integrating our database with another database focused on "Acquisition."

That is the ERD of the Acquisition database:



In order to seamlessly integrate our database with the Acquisition database, we have devised a comprehensive plan that involves executing three strategically designed queries. These queries have been meticulously crafted to combine the data from both databases, allowing us to establish a cohesive and synchronized information system.

1-Query

The following query generates a list of employees who are simultaneously working in both our database and the Acquisition database. It includes their ID from the Acquisition database, their salary in our database, and the quantity of orders they have completed in the Acquisition database. The query filters the results to only include employees whose order quantity is 5 or greater and whose salary is below 50000.

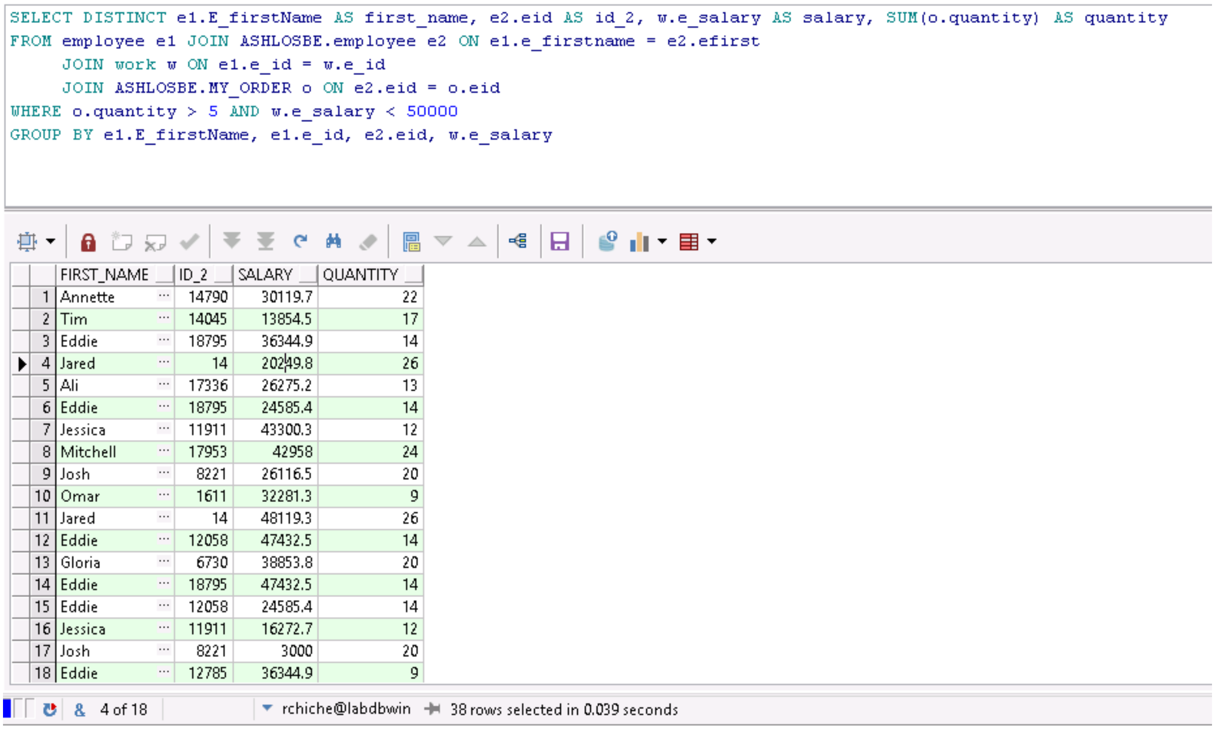
SQL Code

```sql
SELECT DISTINCT e1.E_firstName AS first_name, e2.eid AS id_2, w.e_salary AS
salary, SUM(o.quantity) AS quantity

FROM employee e1

        JOIN ASHLOSBE.employee e2 ON e1.e_firstname = e2.efirst

        JOIN work w ON e1.e_id = w.e_id

        JOIN ASHLOSBE.MY_ORDER o ON e2.eid = o.eid

WHERE o.quantity > 5 AND w.e_salary < 50000

GROUP BY e1.E_firstName, e1.e_id, e2.eid, w.e_salary;
```

Motivation

The motivation of this query allows for an evaluation of employees' eligibility and suitability for promotions based on their performance and compensation. By considering the employees who exist in both the databases, along with their respective IDs, salaries, and order quantities, the query provides valuable insights into their performance metrics and productivity.

Result



```
SELECT DISTINCT e1.E_firstName AS first_name, e2.eid AS id_2, w.e_salary AS salary, SUM(o.quantity) AS quantity
FROM employee e1 JOIN ASHLOSBE.employee e2 ON e1.e_firstname = e2.efirst
     JOIN work w ON e1.e_id = w.e_id
     JOIN ASHLOSBE.MY_ORDER o ON e2.eid = o.eid
WHERE o.quantity > 5 AND w.e_salary < 50000
GROUP BY e1.E_firstName, e1.e_id, e2.eid, w.e_salary
```

	FIRST_NAME	ID_2	SALARY	QUANTITY
1	Annette	14790	30119.7	22
2	Tim	14045	13854.5	17
3	Eddie	18795	36344.9	14
4	Jared	14	20249.8	26
5	Ali	17336	26275.2	13
6	Eddie	18795	24585.4	14
7	Jessica	11911	43300.3	12
8	Mitchell	17953	42958	24
9	Josh	8221	26116.5	20
10	Omar	1611	32281.3	9
11	Jared	14	48119.3	26
12	Eddie	12058	47432.5	14
13	Gloria	6730	38853.8	20
14	Eddie	18795	47432.5	14
15	Eddie	12058	24585.4	14
16	Jessica	11911	16272.7	12
17	Josh	8221	3000	20
18	Eddie	12785	36344.9	9

2-Query

This query retrieves a list of company names and their corresponding phone numbers from the supplier table of the Acquisition database. It selects only those companies that are not present in our database but exist exclusively in the Acquisition database.

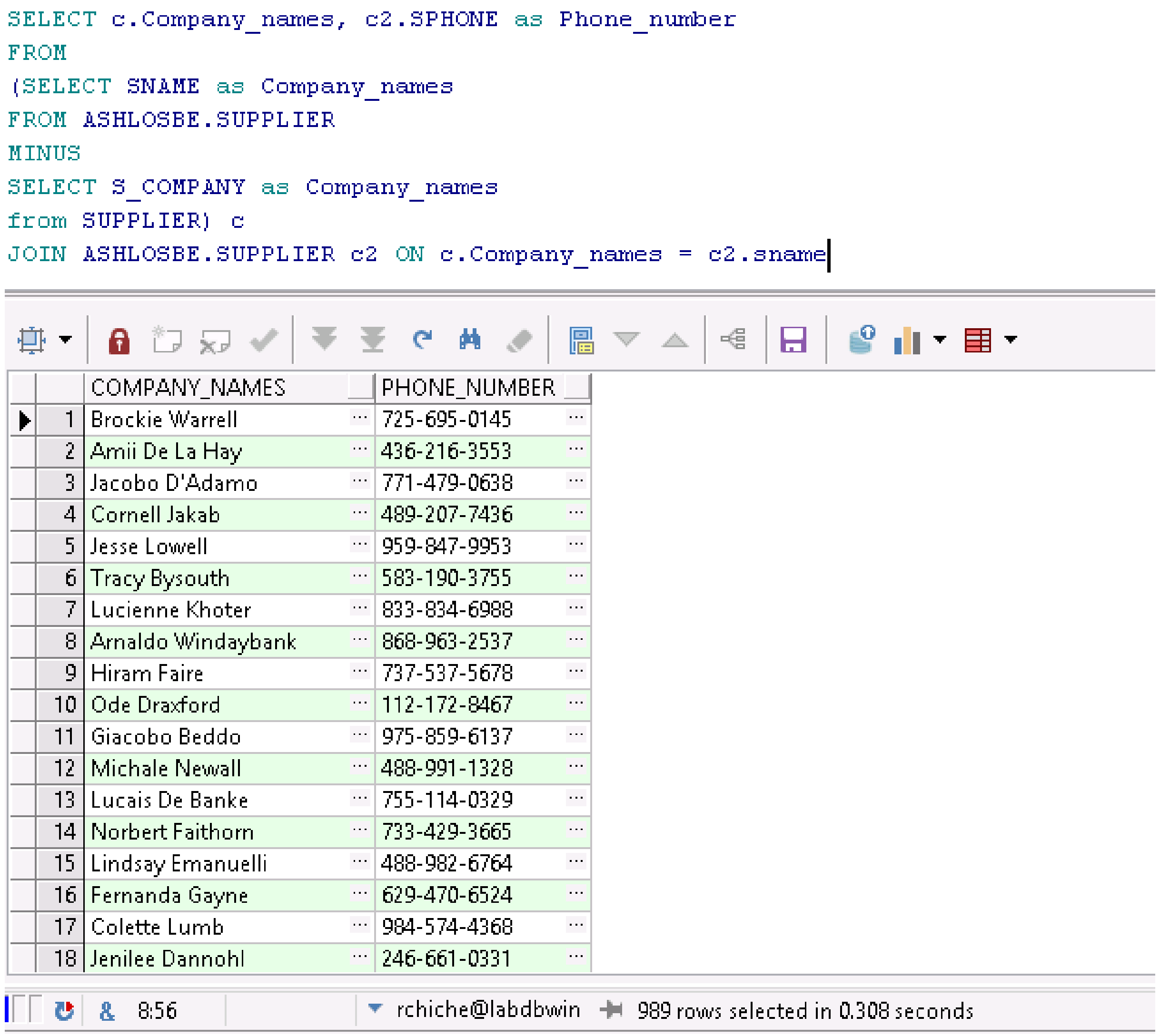
SQL Code

```
SELECT c.Company_names, c2.SPHONE AS Phone_number
FROM
   (SELECT SNAME AS Company_names
    FROM ASHLOSBE.SUPPLIER
             MINUS
        SELECT S_COMPANY AS Company_names
    FROM SUPPLIER) c
       JOIN ASHLOSBE.SUPPLIER c2 ON c.Company_names = c2.SNAME;
```

Motivation

The motivation of this query is to identify The companies that are present in the Acquisition database because they may represent potential business partners or suppliers that we have not yet established relationships with. By identifying these companies, the query opens up opportunities to explore and evaluate the potential benefits of adding them to our database, such as establishing new business relationships and diversifying our supplier base.

Result



```
SELECT c.Company_names, c2.SPHONE as Phone_number
FROM
(SELECT SNAME as Company_names
FROM ASHLOSBE.SUPPLIER
MINUS
SELECT S_COMPANY as Company_names
from SUPPLIER) c
JOIN ASHLOSBE.SUPPLIER c2 ON c.Company_names = c2.sname
```

	COMPANY_NAMES	PHONE_NUMBER
1	Brockie Warrell	725-695-0145
2	Amii De La Hay	436-216-3553
3	Jacobo D'Adamo	771-479-0638
4	Cornell Jakab	489-207-7436
5	Jesse Lowell	959-847-9953
6	Tracy Bysouth	583-190-3755
7	Lucienne Khoter	833-834-6988
8	Arnaldo Windaybank	868-963-2537
9	Hiram Faire	737-537-5678
10	Ode Draxford	112-172-8467
11	Giacobo Beddo	975-859-6137
12	Michale Newall	488-991-1328
13	Lucais De Banke	755-114-0329
14	Norbert Faithorn	733-429-3665
15	Lindsay Emanuelli	488-982-6764
16	Fernanda Gayne	629-470-6524
17	Colette Lumb	984-574-4368
18	Jenilee Dannohl	246-661-0331

8:56 rchiche@labdbwin 989 rows selected in 0.308 seconds

3-Query

This query retrieves the list of the kitchens that make meals with a price of 45 or greater, and if an equipment exists for that specific meal.

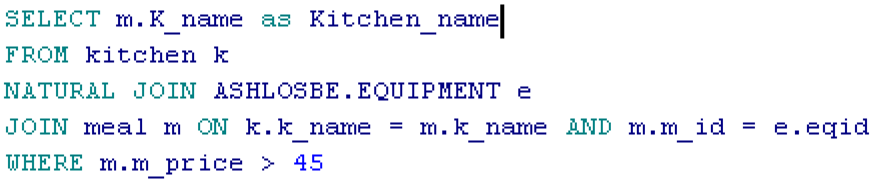
SQL Code

```sql
SELECT m.K_name as Kitchen_name
FROM kitchen k
        NATURAL JOIN ASHLOSBE.EQUIPMENT e
        JOIN meal m ON k.k_name = m.k_name AND m.m_id = e.eqid
WHERE m.m_price > 45;
```

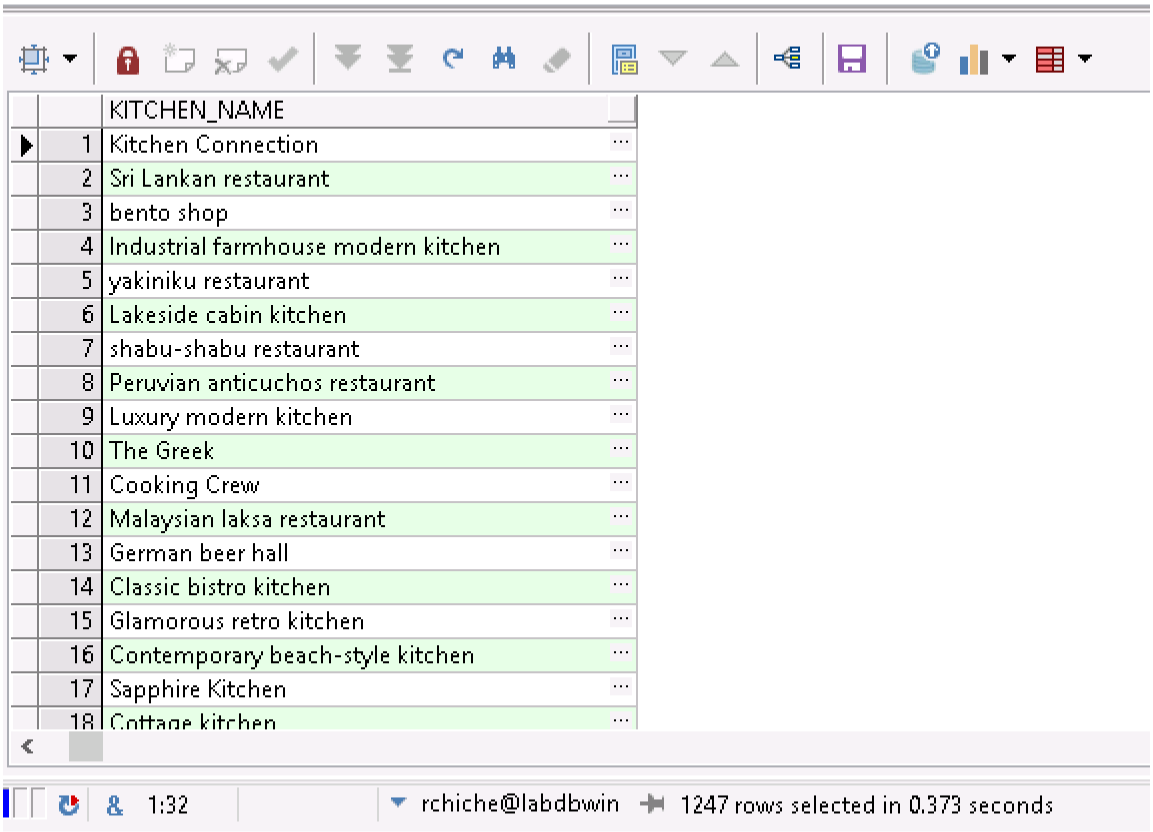
Motivation

The motivation of this query is to enable evaluating the relationship between meal prices and performance. By considering the kitchen names, equipment associations, and meal prices, it becomes possible to assess the value proposition of meals. This analysis can help identify opportunities to adjust prices, optimize ingredient costs, or enhance the overall dining experience based on customer preferences and budget considerations.

Result



```
SELECT m.K_name as Kitchen_name
FROM kitchen k
NATURAL JOIN ASHLOSBE.EQUIPMENT e
JOIN meal m ON k.k_name = m.k_name AND m.m_id = e.eqid
WHERE m.m_price > 45
```



	KITCHEN_NAME
1	Kitchen Connection
2	Sri Lankan restaurant
3	bento shop
4	Industrial farmhouse modern kitchen
5	yakiniku restaurant
6	Lakeside cabin kitchen
7	shabu-shabu restaurant
8	Peruvian anticuchos restaurant
9	Luxury modern kitchen
10	The Greek
11	Cooking Crew
12	Malaysian laksa restaurant
13	German beer hall
14	Classic bistro kitchen
15	Glamorous retro kitchen
16	Contemporary beach-style kitchen
17	Sapphire Kitchen
18	Cottage kitchen

1:32 rchiche@labdbwin 1247 rows selected in 0.373 seconds

Views

1-View

Employee Details

This view combines information from the Employee and Work tables to provide a comprehensive view of employee details, including their salary and the kitchen they work in.

```
CREATE VIEW EmployeeDetails AS
SELECT E.E_Id, E.E_FirstName, E.E_LastName, E.E_Telephone, E.E_Address,
E.E_Age, W.E_Salary, W.K_Name
FROM Employee E
JOIN Work W ON E.E_Id = W.E_Id;
```

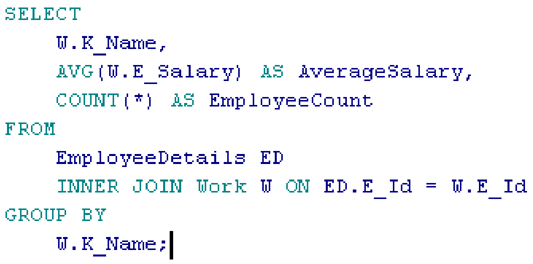
```
CREATE VIEW EmployeeDetails AS
SELECT E.E_Id, E.E_FirstName, E.E_LastName, E.E_Telephone, E.E_Address, E.E_Age, W.E_Salary, W.K_Name
FROM Employee E
JOIN Work W ON E.E_Id = W.E_Id;
```

1-Query

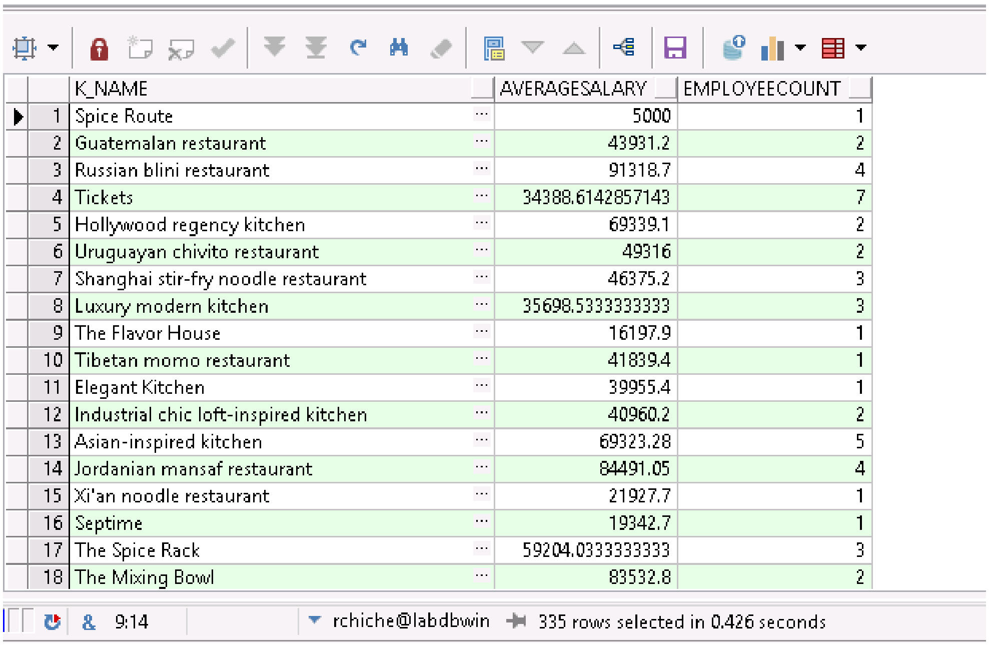
Get the average salary of employees for each kitchen, along with the total number of employees in each kitchen.

```
SELECT
    W.K_Name,
    AVG(W.E_Salary) AS AverageSalary,
    COUNT(*) AS EmployeeCount
FROM
    EmployeeDetails ED
    INNER JOIN Work W ON ED.E_Id = W.E_Id
GROUP BY
    W.K_Name;
```

Result



```
SELECT
    W.K_Name,
    AVG(W.E_Salary) AS AverageSalary,
    COUNT(*) AS EmployeeCount
FROM
    EmployeeDetails ED
    INNER JOIN Work W ON ED.E_Id = W.E_Id
GROUP BY
    W.K_Name;
```



	K_NAME	AVERAGESALARY	EMPLOYEECOUNT
1	Spice Route	5000	1
2	Guatemalan restaurant	43931.2	2
3	Russian blini restaurant	91318.7	4
4	Tickets	34388.6142857143	7
5	Hollywood regency kitchen	69339.1	2
6	Uruguayan chivito restaurant	49316	2
7	Shanghai stir-fry noodle restaurant	46375.2	3
8	Luxury modern kitchen	35698.5333333333	3
9	The Flavor House	16197.9	1
10	Tibetan momo restaurant	41839.4	1
11	Elegant Kitchen	39955.4	1
12	Industrial chic loft-inspired kitchen	40960.2	2
13	Asian-inspired kitchen	69323.28	5
14	Jordanian mansaf restaurant	84491.05	4
15	Xi'an noodle restaurant	21927.7	1
16	Septime	19342.7	1
17	The Spice Rack	59204.0333333333	3
18	The Mixing Bowl	83532.8	2

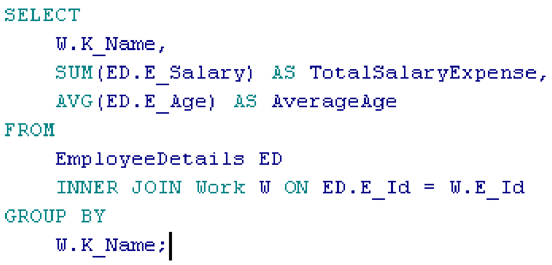
9:14 rchiche@labdbwin 335 rows selected in 0.426 seconds

2-Query

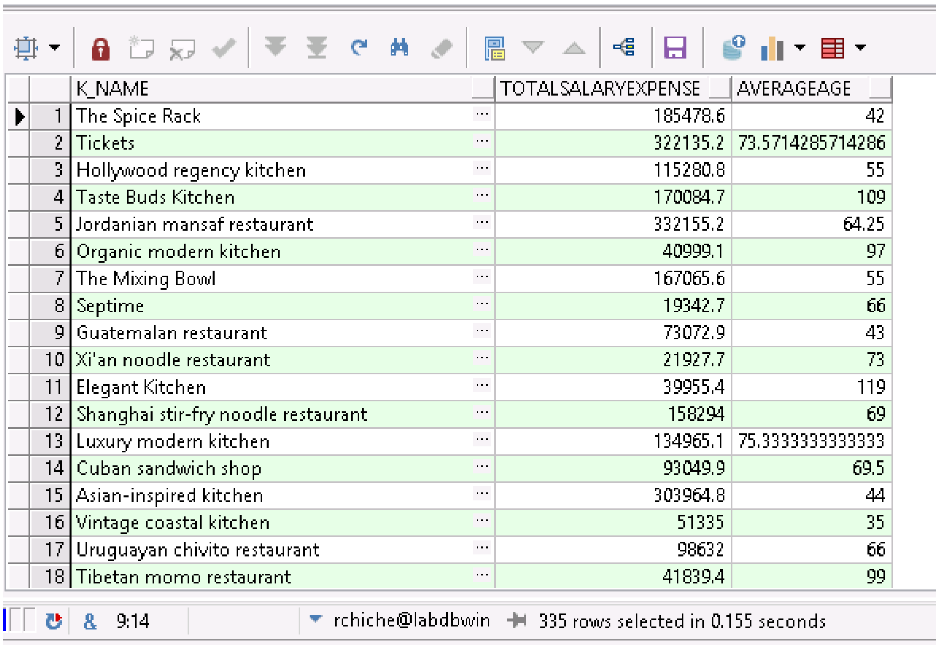
Get the total salary expense for each kitchen, along with the average age of employees in each kitchen.

```sql
SELECT
   W.K_Name,
   SUM(ED.E_Salary) AS TotalSalaryExpense,
   AVG(ED.E_Age) AS AverageAge
FROM
   EmployeeDetails ED
       INNER JOIN Work W ON ED.E_Id = W.E_Id
GROUP BY
   W.K_Name;
```

Result



```sql
SELECT
    W.K_Name,
    SUM(ED.E_Salary) AS TotalSalaryExpense,
    AVG(ED.E_Age) AS AverageAge
FROM
    EmployeeDetails ED
    INNER JOIN Work W ON ED.E_Id = W.E_Id
GROUP BY
    W.K_Name;
```



	K_NAME	TOTALSALARYEXPENSE	AVERAGEAGE
1	The Spice Rack	185478.6	42
2	Tickets	322135.2	73.5714285714286
3	Hollywood regency kitchen	115280.8	55
4	Taste Buds Kitchen	170084.7	109
5	Jordanian mansaf restaurant	332155.2	64.25
6	Organic modern kitchen	40999.1	97
7	The Mixing Bowl	167065.6	55
8	Septime	19342.7	66
9	Guatemalan restaurant	73072.9	43
10	Xi'an noodle restaurant	21927.7	73
11	Elegant Kitchen	39955.4	119
12	Shanghai stir-fry noodle restaurant	158294	69
13	Luxury modern kitchen	134965.1	75.3333333333333
14	Cuban sandwich shop	93049.9	69.5
15	Asian-inspired kitchen	303964.8	44
16	Vintage coastal kitchen	51335	35
17	Uruguayan chivito restaurant	98632	66
18	Tibetan momo restaurant	41839.4	99

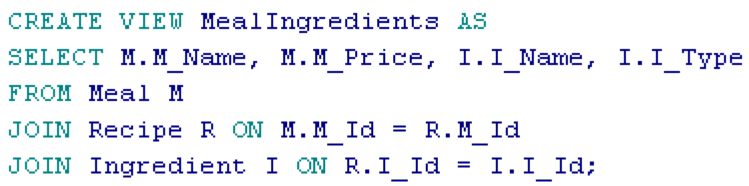
9:14 rchiche@labdbwin 335 rows selected in 0.155 seconds

2-View

Meal Ingredients

This view combines information from the Meal and Recipe tables to provide a list of meals along with their corresponding ingredients.

```sql
CREATE VIEW MealIngredients AS
SELECT M.M_Name, M.M_Price, I.I_Name, I.I_Type
FROM Meal M
        JOIN Recipe R ON M.M_Id = R.M_Id
        JOIN Ingredient I ON R.I_Id = I.I_Id;
```



```sql
CREATE VIEW MealIngredients AS
SELECT M.M_Name, M.M_Price, I.I_Name, I.I_Type
FROM Meal M
JOIN Recipe R ON M.M_Id = R.M_Id
JOIN Ingredient I ON R.I_Id = I.I_Id;
```

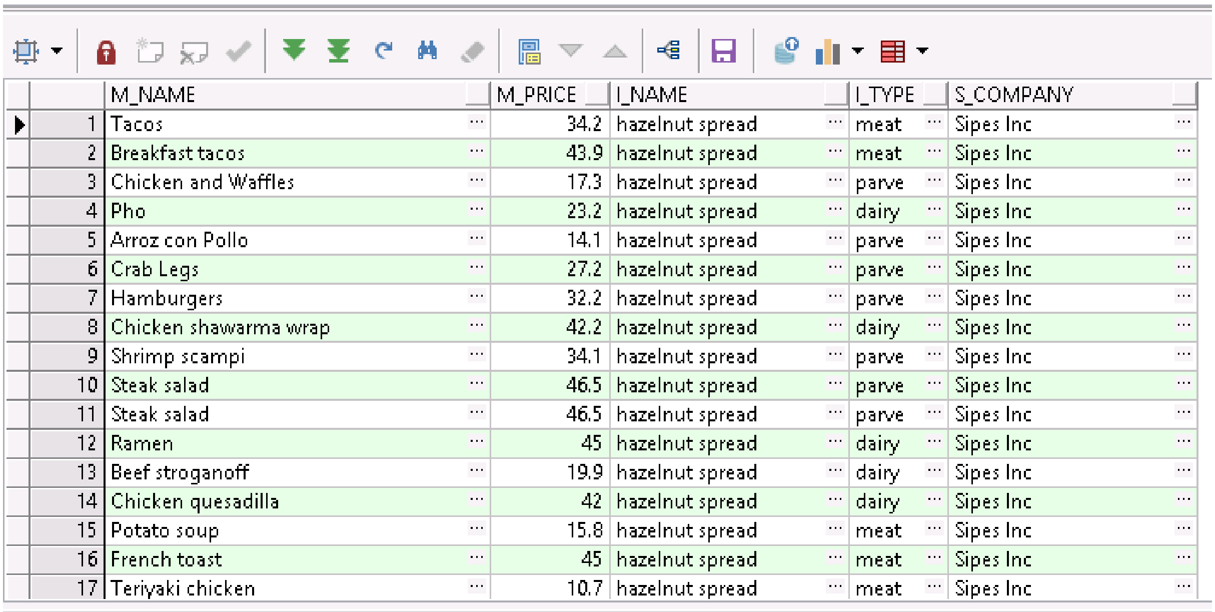
1-Query

Get the list of meals along with their ingredients and the corresponding supplier companies for each ingredient.

```sql
SELECT MI.M_Name,
      MI.M_Price,
      MI.I_Name,
      MI.I_Type,
      S.S_Company
FROM MealIngredients MI
        INNER JOIN Ingredient I ON MI.I_Name = I.I_Name
        INNER JOIN Supplier S ON I.S_Company = S.S_Company;
```

Result

```
SELECT
    MI.M_Name,
    MI.M_Price,
    MI.I_Name,
    MI.I_Type,
    S.S_Company
FROM
    MealIngredients MI
    INNER JOIN Ingredient I ON MI.I_Name = I.I_Name
    INNER JOIN Supplier S ON I.S_Company = S.S_Company;
```



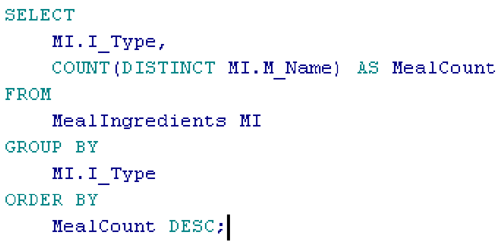
	M_NAME	M_PRICE	I_NAME	I_TYPE	S_COMPANY
1	Tacos	34.2	hazelnut spread	meat	Sipes Inc
2	Breakfast tacos	43.9	hazelnut spread	meat	Sipes Inc
3	Chicken and Waffles	17.3	hazelnut spread	parve	Sipes Inc
4	Pho	23.2	hazelnut spread	dairy	Sipes Inc
5	Arroz con Pollo	14.1	hazelnut spread	parve	Sipes Inc
6	Crab Legs	27.2	hazelnut spread	parve	Sipes Inc
7	Hamburgers	32.2	hazelnut spread	parve	Sipes Inc
8	Chicken shawarma wrap	42.2	hazelnut spread	dairy	Sipes Inc
9	Shrimp scampi	34.1	hazelnut spread	parve	Sipes Inc
10	Steak salad	46.5	hazelnut spread	parve	Sipes Inc
11	Steak salad	46.5	hazelnut spread	parve	Sipes Inc
12	Ramen	45	hazelnut spread	dairy	Sipes Inc
13	Beef stroganoff	19.9	hazelnut spread	dairy	Sipes Inc
14	Chicken quesadilla	42	hazelnut spread	dairy	Sipes Inc
15	Potato soup	15.8	hazelnut spread	meat	Sipes Inc
16	French toast	45	hazelnut spread	meat	Sipes Inc
17	Teriyaki chicken	10.7	hazelnut spread	meat	Sipes Inc

2-Query

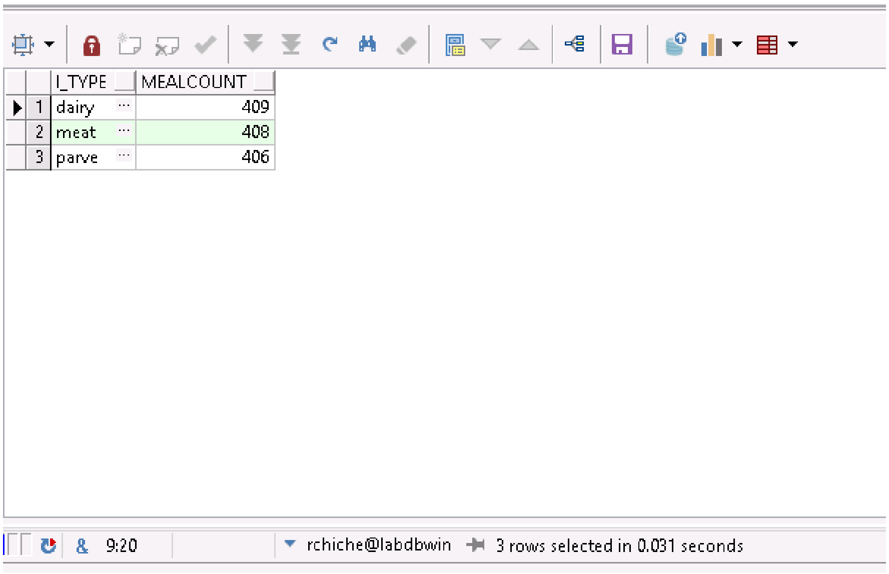
Get the count of meals for each ingredient type, sorted in descending order.

```
SELECT MI.I_Type,
      COUNT(DISTINCT MI.M_Name) AS MealCount
FROM MealIngredients MI
GROUP BY MI.I_Type
ORDER BY MealCount DESC;
```

Result



```
SELECT
    MI.I_Type,
    COUNT(DISTINCT MI.M_Name) AS MealCount
FROM
    MealIngredients MI
GROUP BY
    MI.I_Type
ORDER BY
    MealCount DESC;
```



	I_TYPE	MEALCOUNT
1	dairy	409
2	meat	408
3	parve	406

9:20 rchiche@labdbwin 3 rows selected in 0.031 seconds

Procedures

1-Procedure

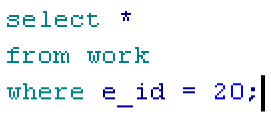
Update Employee Salary

```
CREATE OR REPLACE PROCEDURE UpdateEmployeeSalary(

   p_EmployeeID IN NUMBER,

   p_NewSalary IN NUMBER

)

IS

BEGIN

   UPDATE Work

   SET E_Salary = p_NewSalary

   WHERE E_Id = p_EmployeeID;


   COMMIT;

END;
```

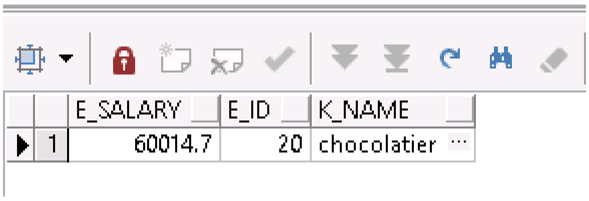
Motivation

The motivation of that procedure is when an employee's salary needs to be updated due to factors like performance reviews, promotions, or changes in job responsibilities, this procedure provides a convenient way to make the necessary updates in the database.

Before

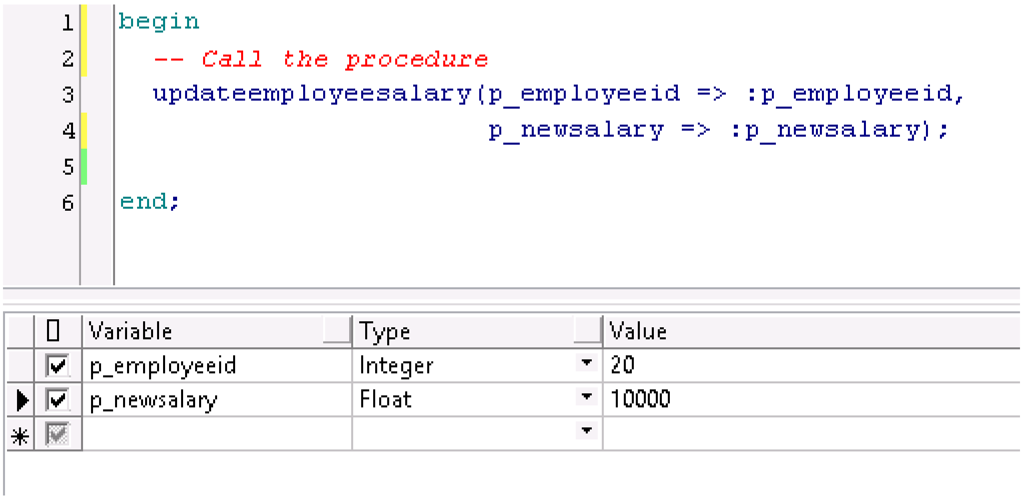


```
select *
from work
where e_id = 20;
```



	E_SALARY	E_ID	K_NAME
▶ 1	60014.7	20	chocolatier ···

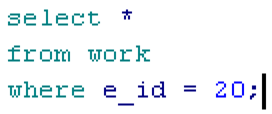
Procedure call



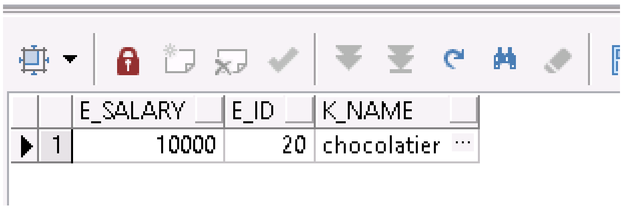
```
begin
  -- Call the procedure
  updateemployeesalary(p_employeeid => :p_employeeid,
                       p_newsalary => :p_newsalary);

end;
```

☐	Variable	Type	Value
☑	p_employeeid	Integer	20
▶ ☑	p_newsalary	Float	10000
✱ ☑			

After



```
select *
from work
where e_id = 20;
```



	E_SALARY	E_ID	K_NAME
▶ 1	10000	20	chocolatier ···

2-Procedure

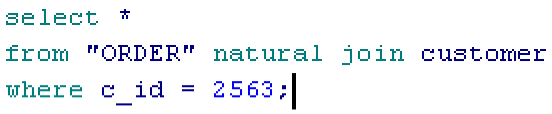
Delete Customer And Orders

```
CREATE OR REPLACE PROCEDURE DeleteCustomerAndOrders(
   p_CustomerID IN NUMBER
)
IS
BEGIN
   DELETE FROM "ORDER"
   WHERE C_Id = p_CustomerID;

   DELETE FROM Customer
   WHERE C_Id = p_CustomerID;

   COMMIT;
END;
```

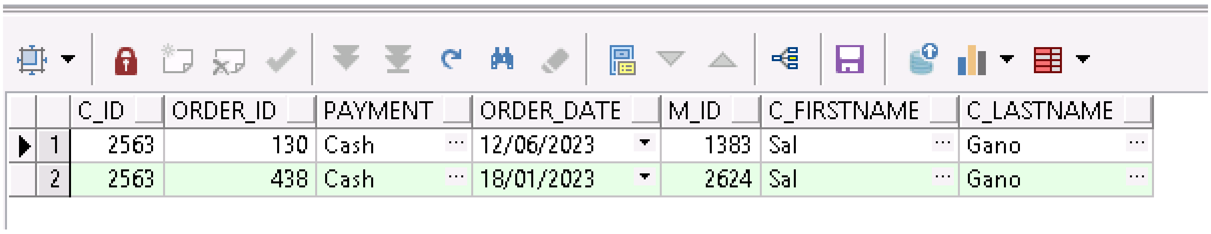
Motivation

The motivation behind the procedure DeleteCustomerAndOrders is to provide a means of deleting a customer record from the Customer table along with all associated orders from the "ORDER" table. This procedure allows for the efficient removal of customer data and related order information from the database.

Before

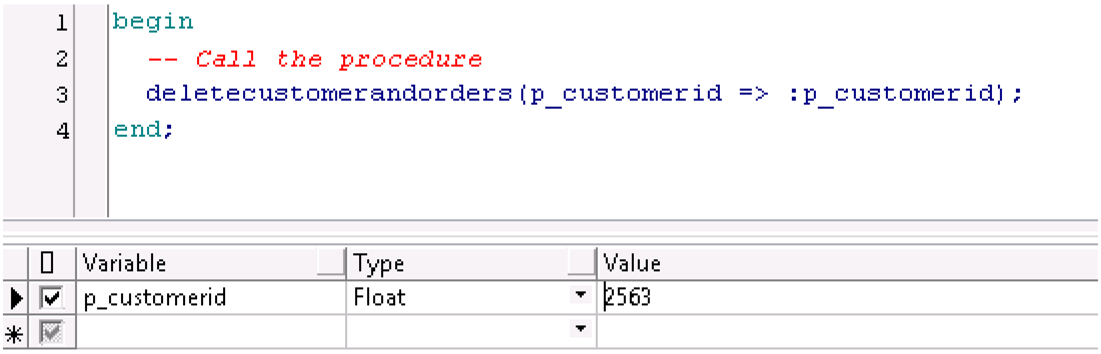


```
select *
from "ORDER" natural join customer
where c_id = 2563;
```



	C_ID	ORDER_ID	PAYMENT	ORDER_DATE	M_ID	C_FIRSTNAME	C_LASTNAME
1	2563	130	Cash	12/06/2023	1383	Sal	Gano
2	2563	438	Cash	18/01/2023	2624	Sal	Gano

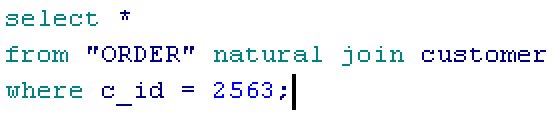
Procedure call



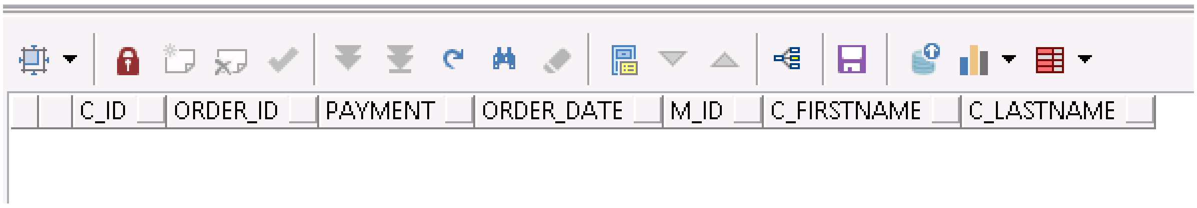
```
begin
  -- Call the procedure
  deletecustomerandorders(p_customerid => :p_customerid);
end;
```

	□	Variable	Type	Value
▶	☑	p_customerid	Float	2563
*	☑			

After



```
select *
from "ORDER" natural join customer
where c_id = 2563;
```



	C_ID	ORDER_ID	PAYMENT	ORDER_DATE	M_ID	C_FIRSTNAME	C_LASTNAME

Functions

1-Function

Generate Reports

```
CREATE OR REPLACE FUNCTION GenerateReport(p_report_name IN VARCHAR2) RETURN NUMBER
IS
 v_entity_count NUMBER := 0;
BEGIN
 IF p_report_name = 'Employee' THEN
   -- Employee report
   FOR emp IN (
     SELECT E_Id, E_FirstName, E_LastName, E_Age, E_Salary, K_Name
     FROM Employee NATURAL JOIN Work
   )
   LOOP
     v_entity_count := v_entity_count + 1;
     DBMS_OUTPUT.PUT_LINE('Employee ID: ' || emp.E_Id);
     DBMS_OUTPUT.PUT_LINE('Employee Name: ' || emp.E_FirstName || ' ' ||
emp.E_LastName);
     DBMS_OUTPUT.PUT_LINE('Employee Age: ' || emp.E_Age);
     DBMS_OUTPUT.PUT_LINE('Employee Salary: ' || emp.E_Salary);
     DBMS_OUTPUT.PUT_LINE('Employee Kitchen: ' || emp.K_Name);
     DBMS_OUTPUT.PUT_LINE('-------------------');
   END LOOP;

 ELSIF p_report_name = 'Customer' THEN
   -- Customer orders report
   FOR order_info IN (
     SELECT o.Order_Id, c.C_FirstName, c.C_LastName, m.M_Name, o.Order_Date
     FROM "ORDER" o
     JOIN Customer c ON o.C_Id = c.C_Id
     JOIN Meal m ON o.M_Id = m.M_Id
   )
   LOOP
     v_entity_count := v_entity_count + 1;
     DBMS_OUTPUT.PUT_LINE('Order ID: ' || order_info.Order_Id);
     DBMS_OUTPUT.PUT_LINE('Customer Name: ' || order_info.C_FirstName || ' ' ||
order_info.C_LastName);
     DBMS_OUTPUT.PUT_LINE('Meal: ' || order_info.M_Name);
     DBMS_OUTPUT.PUT_LINE('Order Date: ' || order_info.Order_Date);
     DBMS_OUTPUT.PUT_LINE('-------------------');
   END LOOP;

 ELSE
   v_entity_count := -1;
 END IF;

 RETURN v_entity_count;
END;
```

Motivation

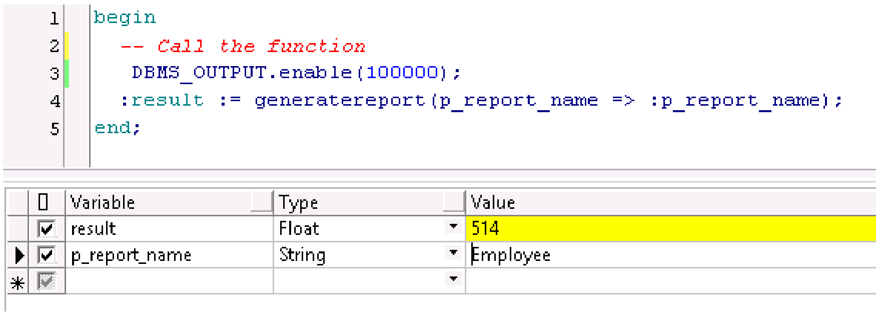
The motivation behind the GenerateReport function is to provide a flexible and reusable solution for generating different types of reports in a database system. By accepting the report name as input, the function can dynamically retrieve the necessary data from the tables and generate the report accordingly.

The function allows for easy extensibility, as additional report types can be handled by adding appropriate conditions and queries within the function. This way, you can generate different reports based on the specific needs of your application or business.

The function also provides the capability to count the number of entities in the generated report. This can be useful for tracking the size or volume of data being included in the report, providing insights into the database's state.

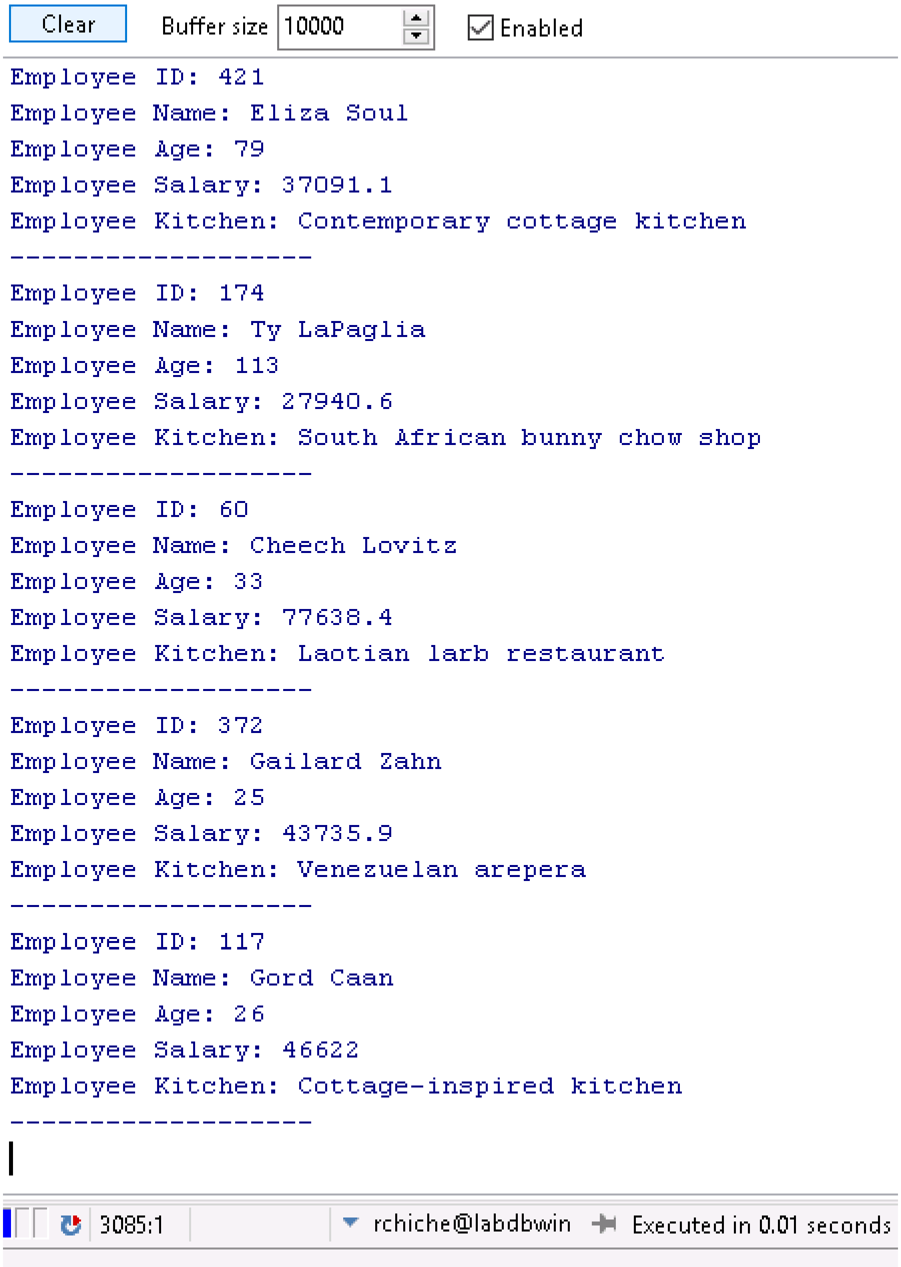
Overall, the GenerateReport function aims to simplify the process of generating reports by encapsulating the logic and allowing for flexibility, reusability, and the inclusion of entity count information.

Call result



```
begin
  -- Call the function
   DBMS_OUTPUT.enable(100000);
  :result := generatereport(p_report_name => :p_report_name);
end;
```

	Variable	Type	Value
☑	result	Float	514
☑	p_report_name	String	Employee
☑			



```
Employee ID: 421
Employee Name: Eliza Soul
Employee Age: 79
Employee Salary: 37091.1
Employee Kitchen: Contemporary cottage kitchen
--------------------
Employee ID: 174
Employee Name: Ty LaPaglia
Employee Age: 113
Employee Salary: 27940.6
Employee Kitchen: South African bunny chow shop
--------------------
Employee ID: 60
Employee Name: Cheech Lovitz
Employee Age: 33
Employee Salary: 77638.4
Employee Kitchen: Laotian larb restaurant
--------------------
Employee ID: 372
Employee Name: Gailard Zahn
Employee Age: 25
Employee Salary: 43735.9
Employee Kitchen: Venezuelan arepera
--------------------
Employee ID: 117
Employee Name: Gord Caan
Employee Age: 26
Employee Salary: 46622
Employee Kitchen: Cottage-inspired kitchen
--------------------
```

2-Function

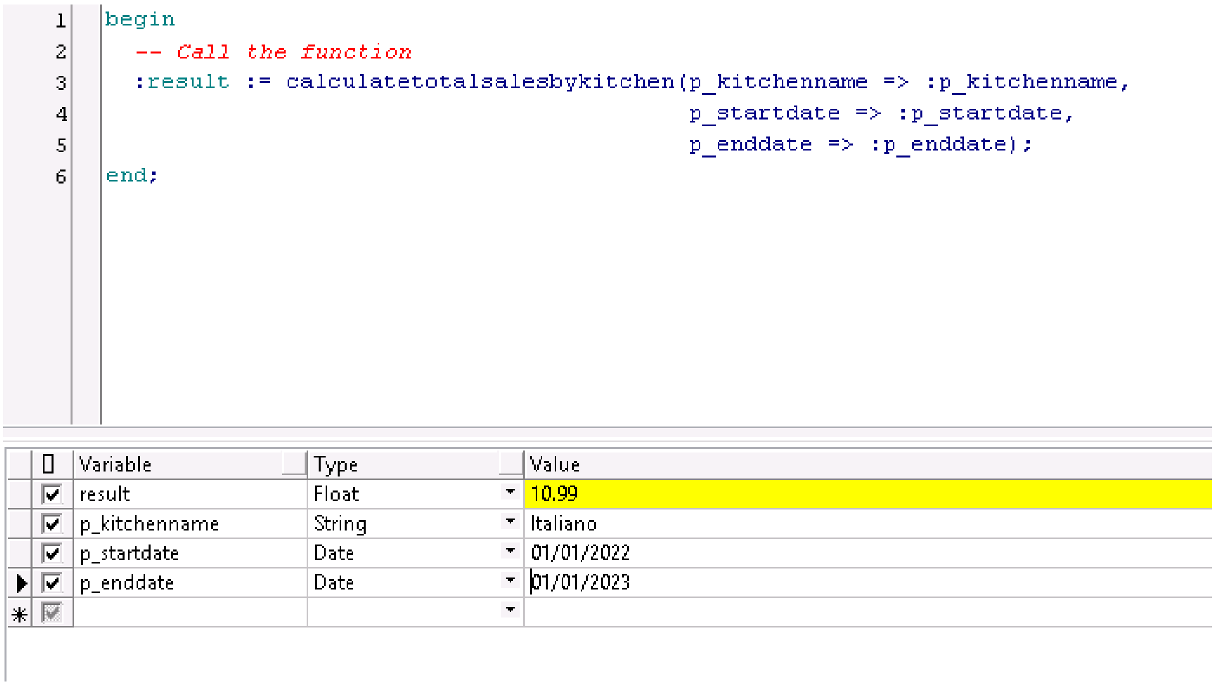
Calculate Total Sales By Kitchen

```
CREATE OR REPLACE FUNCTION CalculateTotalSalesByKitchen(
   p_KitchenName IN VARCHAR2,
   p_StartDate   IN DATE,
   p_EndDate     IN DATE
)
RETURN NUMBER
IS
   v_TotalSales NUMBER := 0;
BEGIN
   SELECT SUM(M_Price)
   INTO v_TotalSales
   FROM Meal
   INNER JOIN "ORDER" ON "ORDER".M_Id = Meal.M_Id
   WHERE Meal.K_Name = p_KitchenName
     AND "ORDER".Order_Date BETWEEN p_StartDate AND p_EndDate;

   -- If no sales found, set total sales to 0
   IF v_TotalSales IS NULL THEN
       v_TotalSales := 0;
   END IF;

   RETURN v_TotalSales;
END;
```

Motivation

The procedure CalculateTotalSalesByKitchen is needed to determine the total sales generated by a specific kitchen within a given date range. This information is essential for evaluating kitchen performance, conducting financial analysis, generating sales reports, making data-driven decisions, implementing performance incentives, tracking goals, and allocating resources effectively. The procedure enables businesses to assess revenue generation, make informed decisions, and monitor the sales performance of individual kitchens for better management and financial planning.

Call result



```
begin
  -- Call the function
  :result := calculatetotalsalesbykitchen(p_kitchenname => :p_kitchenname,
                                          p_startdate => :p_startdate,
                                          p_enddate => :p_enddate);
end;
```

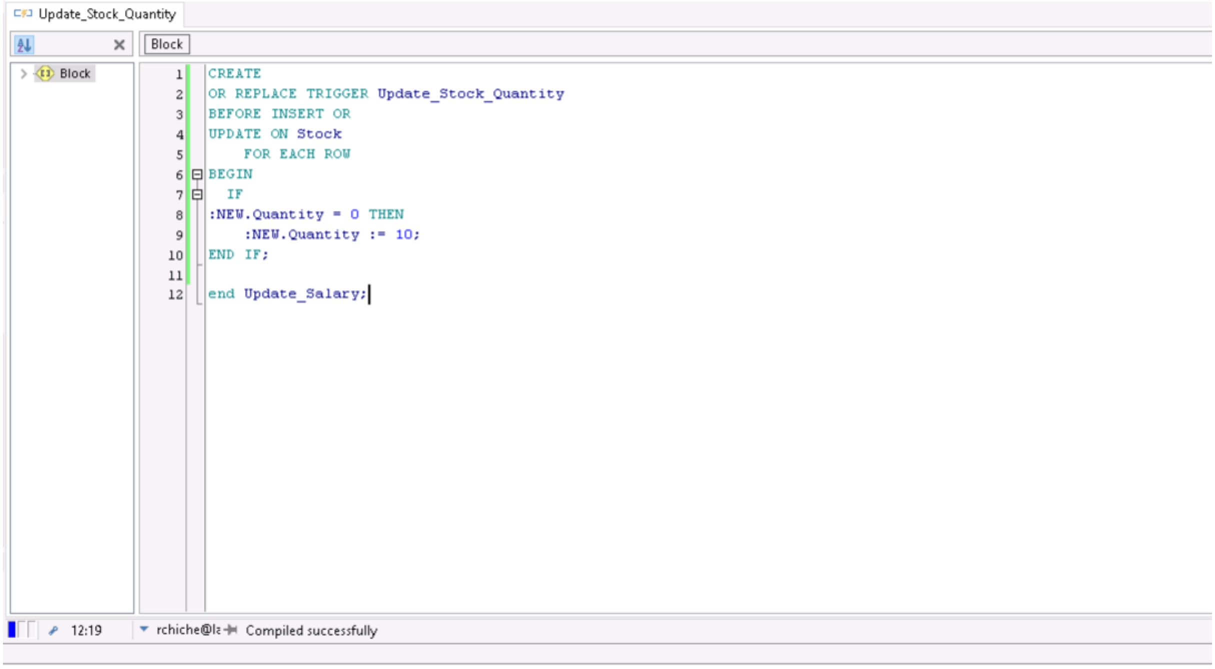
	□	Variable	Type	Value
	☑	result	Float	10.99
	☑	p_kitchenname	String	Italiano
	☑	p_startdate	Date	01/01/2022
▶	☑	p_enddate	Date	01/01/2023
✳	☑			

Triggers

1-Trigger

Update Stock Quantity

```
CREATE
OR REPLACE TRIGGER Update_Stock_Quantity
BEFORE INSERT OR
UPDATE ON Stock
   FOR EACH ROW
BEGIN
 IF
:NEW.Quantity = 0 THEN
   :NEW.Quantity := 10;
END IF;
END;
```



```
CREATE
OR REPLACE TRIGGER Update_Stock_Quantity
BEFORE INSERT OR
UPDATE ON Stock
    FOR EACH ROW
BEGIN
  IF
:NEW.Quantity = 0 THEN
    :NEW.Quantity := 10;
END IF;

end Update_Salary;
```

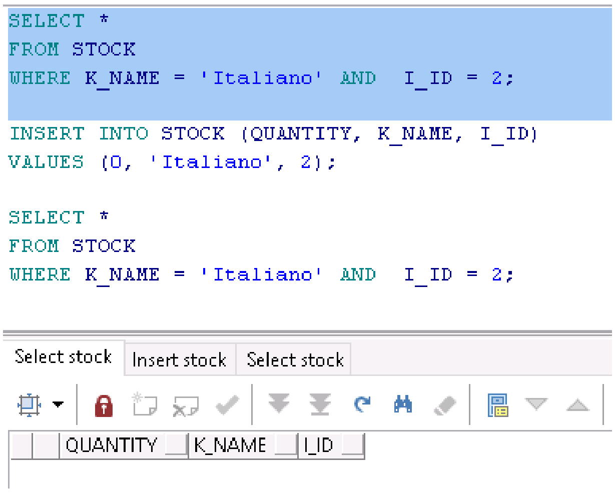
Compiled successfully

Motivation

This trigger is activated whenever a row is inserted or updated in the "Stock" table and the quantity field has a value of zero, it will automatically be updated to 10. This trigger helps ensure that the stock quantity never reaches zero and assumes that the managers buy 10 units of that ingredient when it runs out.

Trigger call

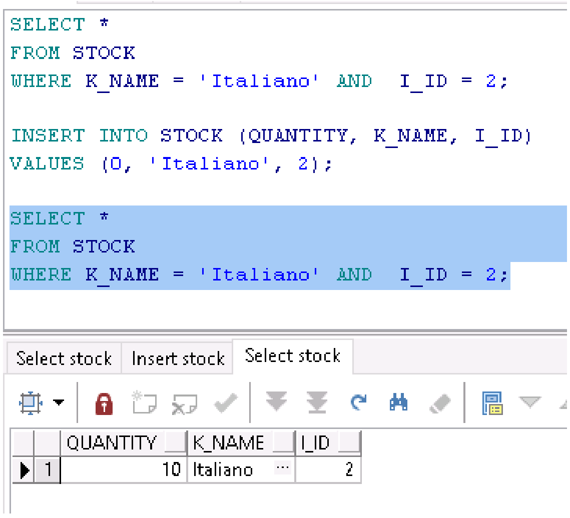
Before



```
SELECT *
FROM STOCK
WHERE K_NAME = 'Italiano' AND  I_ID = 2;
INSERT INTO STOCK (QUANTITY, K_NAME, I_ID)
VALUES (0, 'Italiano', 2);

SELECT *
FROM STOCK
WHERE K_NAME = 'Italiano' AND  I_ID = 2;
```

	QUANTITY	K_NAME	I_ID

After



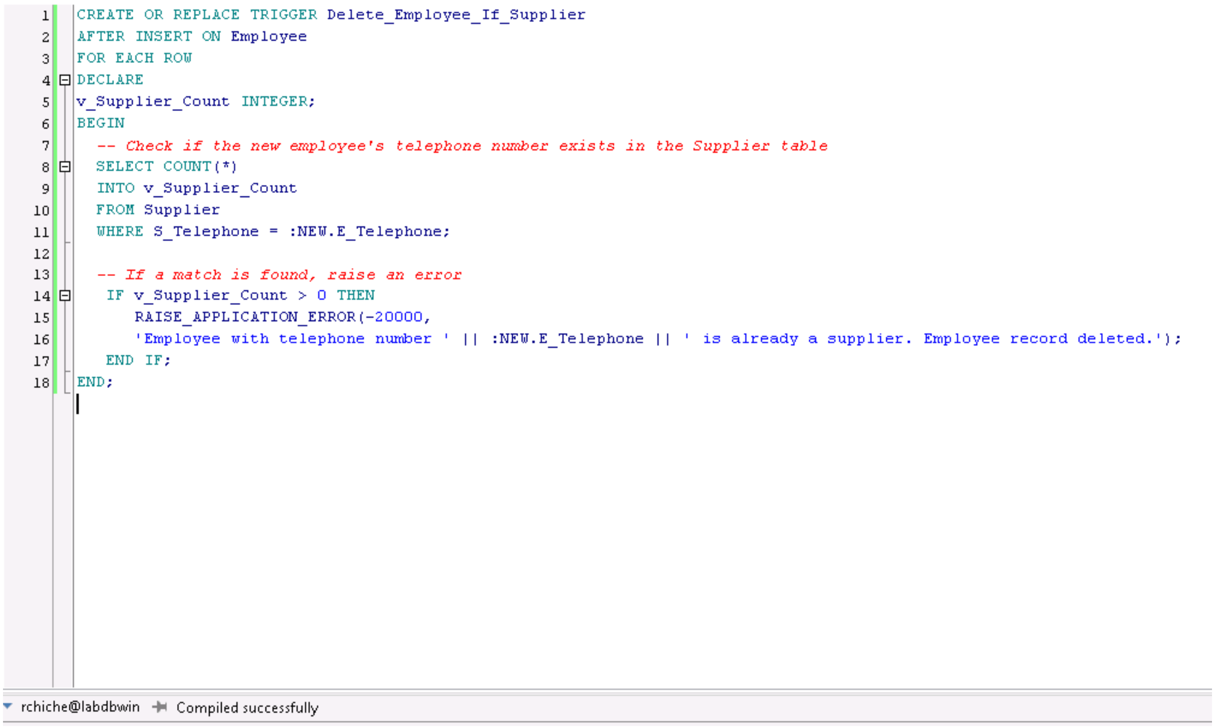
```
SELECT *
FROM STOCK
WHERE K_NAME = 'Italiano' AND  I_ID = 2;

INSERT INTO STOCK (QUANTITY, K_NAME, I_ID)
VALUES (0, 'Italiano', 2);

SELECT *
FROM STOCK
WHERE K_NAME = 'Italiano' AND  I_ID = 2;
```

	QUANTITY	K_NAME	I_ID
1	10	Italiano	2

2-Trigger

Delete Employee If Supplier

```
CREATE OR REPLACE TRIGGER Delete_Employee_If_Supplier

BEFORE INSERT ON Employee

FOR EACH ROWDECLARE

BEGIN  -- Check if the new employee's telephone number exists in the Supplier table

 SELECT COUNT(*)  INTO v_Supplier_Count

 FROM Supplier  WHERE S_Telephone = :NEW.E_Telephone;

 -- If a match is found, raise an error

   IF v_Supplier_Count > 0 THEN

      RAISE_APPLICATION_ERROR(-20000,

     'Employee with telephone number ' || :NEW.E_Telephone || ' is already a supplier. Employee record deleted.');

   END IF;

END;
```



```
CREATE OR REPLACE TRIGGER Delete_Employee_If_Supplier
AFTER INSERT ON Employee
FOR EACH ROW
DECLARE
v_Supplier_Count INTEGER;
BEGIN
  -- Check if the new employee's telephone number exists in the Supplier table
  SELECT COUNT(*)
  INTO v_Supplier_Count
  FROM Supplier
  WHERE S_Telephone = :NEW.E_Telephone;

  -- If a match is found, raise an error
   IF v_Supplier_Count > 0 THEN
      RAISE_APPLICATION_ERROR(-20000,
      'Employee with telephone number ' || :NEW.E_Telephone || ' is already a supplier. Employee record deleted.');
   END IF;
END;
```

rchiche@labdbwin Compiled successfully

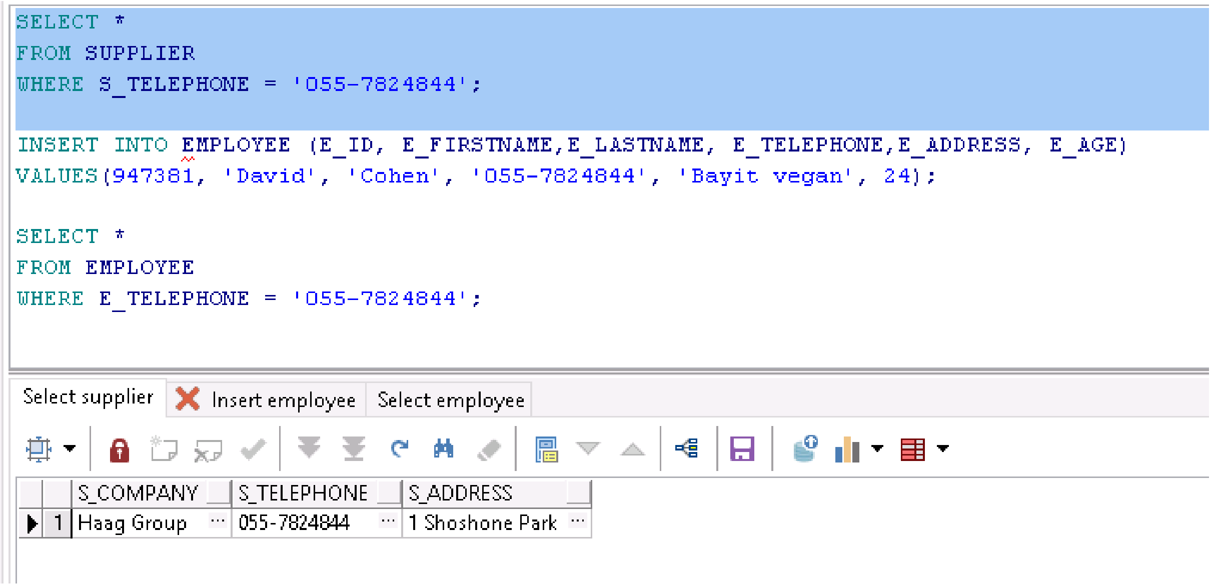
Motivation

The trigger checks if a new row in the "Employee" table has the same telephone number as any "S_Telephone" field in the "Supplier" table. If a match is found, the trigger will raise an error with a specific message.

The trigger's motivation is to ensure data consistency and prevent conflicts in the database. If an employee's telephone number matches a supplier's telephone number, it implies that the employee is already associated with the supplier role. In such cases, it might be desirable to remove the employee from the "Employee" table to avoid duplication or conflicts.

Trigger call

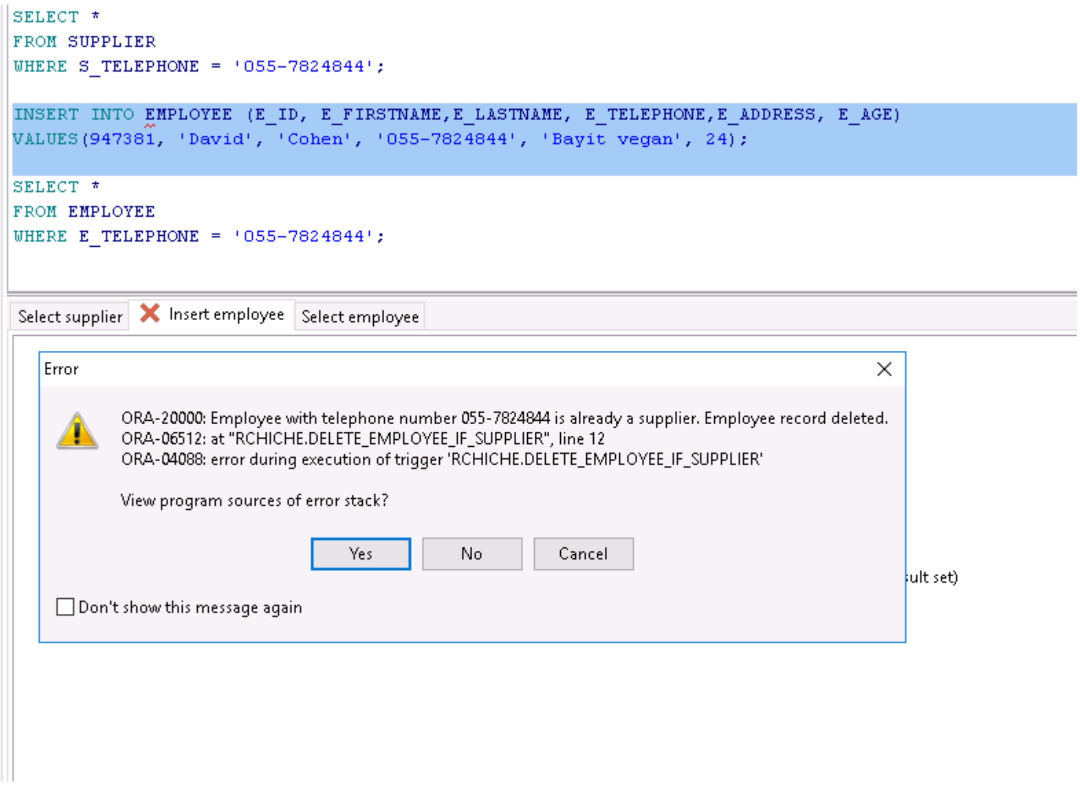
Before



```
SELECT *
FROM SUPPLIER
WHERE S_TELEPHONE = '055-7824844';

INSERT INTO EMPLOYEE (E_ID, E_FIRSTNAME,E_LASTNAME, E_TELEPHONE,E_ADDRESS, E_AGE)
VALUES(947381, 'David', 'Cohen', '055-7824844', 'Bayit vegan', 24);

SELECT *
FROM EMPLOYEE
WHERE E_TELEPHONE = '055-7824844';
```

	S_COMPANY	S_TELEPHONE	S_ADDRESS
1	Haag Group	055-7824844	1 Shoshone Park

After



```
SELECT *
FROM SUPPLIER
WHERE S_TELEPHONE = '055-7824844';

INSERT INTO EMPLOYEE (E_ID, E_FIRSTNAME,E_LASTNAME, E_TELEPHONE,E_ADDRESS, E_AGE)
VALUES(947381, 'David', 'Cohen', '055-7824844', 'Bayit vegan', 24);

SELECT *
FROM EMPLOYEE
WHERE E_TELEPHONE = '055-7824844';
```

```
ORA-20000: Employee with telephone number 055-7824844 is already a supplier. Employee record deleted.
ORA-06512: at "RCHICHE.DELETE_EMPLOYEE_IF_SUPPLIER", line 12
ORA-04088: error during execution of trigger 'RCHICHE.DELETE_EMPLOYEE_IF_SUPPLIER'
```

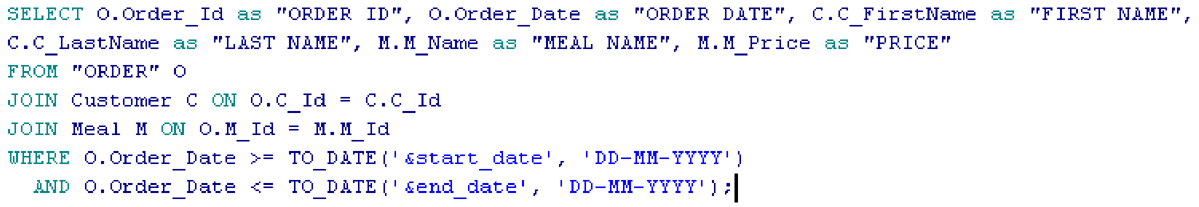
View program sources of error stack?

□ Don't show this message again

Reports

1-Report

```
SELECT O.Order_Id as "ORDER ID", O.Order_Date as "ORDER DATE",
C.C_FirstName as "FIRST NAME", C.C_LastName as "LAST NAME",
M.M_Name as "MEAL NAME", M.M_Price as "PRICE"

FROM "ORDER" OJOIN Customer C ON O.C_Id = C.C_Id

JOIN Meal M ON O.M_Id = M.M_IdWHERE O.Order_Date >=
TO_DATE('&start_date', 'DD-MM-YYYY')

 AND O.Order_Date <= TO_DATE('&end_date', 'DD-MM-YYYY');
```



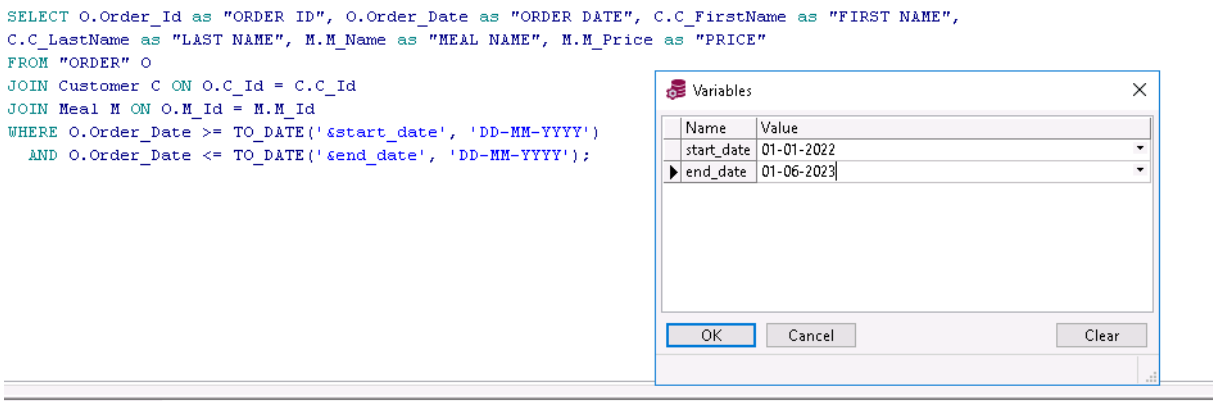
```
SELECT O.Order_Id as "ORDER ID", O.Order_Date as "ORDER DATE", C.C_FirstName as "FIRST NAME",
C.C_LastName as "LAST NAME", M.M_Name as "MEAL NAME", M.M_Price as "PRICE"
FROM "ORDER" O
JOIN Customer C ON O.C_Id = C.C_Id
JOIN Meal M ON O.M_Id = M.M_Id
WHERE O.Order_Date >= TO_DATE('&start_date', 'DD-MM-YYYY')
  AND O.Order_Date <= TO_DATE('&end_date', 'DD-MM-YYYY');
```

Motivation

The motivation of the query is to generate a report that lists all the orders made between two specified dates. The report includes details such as the order ID, order date, customer's first name, customer's last name, meal name, and meal price.
The purpose of this report could be to analyze and track the order history within a specific time range. It allows you to review the orders placed by customers during the specified period and gain insights into the sales and customer preferences during that time.

Result

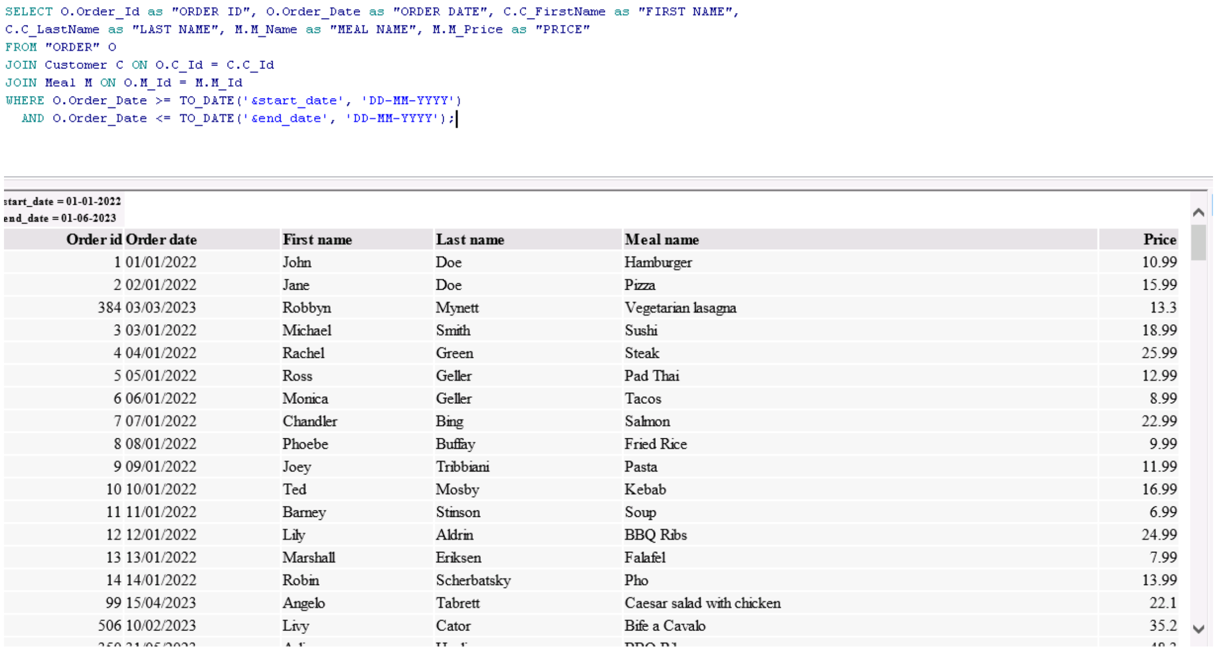
Entering the dates



```
SELECT O.Order_Id as "ORDER ID", O.Order_Date as "ORDER DATE", C.C_FirstName as "FIRST NAME",
C.C_LastName as "LAST NAME", M.M_Name as "MEAL NAME", M.M_Price as "PRICE"
FROM "ORDER" O
JOIN Customer C ON O.C_Id = C.C_Id
JOIN Meal M ON O.M_Id = M.M_Id
WHERE O.Order_Date >= TO_DATE('&start_date', 'DD-MM-YYYY')
  AND O.Order_Date <= TO_DATE('&end_date', 'DD-MM-YYYY');
```

Name	Value
start_date	01-01-2022
end_date	01-06-2023

The report output



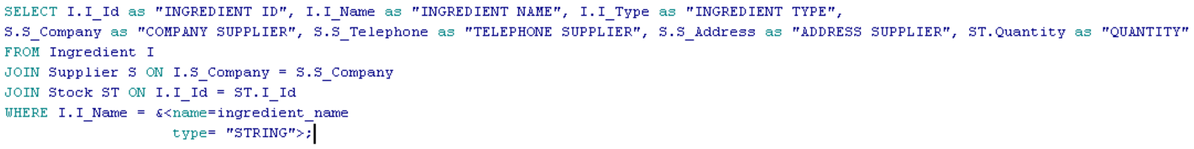
```
SELECT O.Order_Id as "ORDER ID", O.Order_Date as "ORDER DATE", C.C_FirstName as "FIRST NAME",
C.C_LastName as "LAST NAME", M.M_Name as "MEAL NAME", M.M_Price as "PRICE"
FROM "ORDER" O
JOIN Customer C ON O.C_Id = C.C_Id
JOIN Meal M ON O.M_Id = M.M_Id
WHERE O.Order_Date >= TO_DATE('&start_date', 'DD-MM-YYYY')
  AND O.Order_Date <= TO_DATE('&end_date', 'DD-MM-YYYY');
```

start_date = 01-01-2022
end_date = 01-06-2023

Order id	Order date	First name	Last name	Meal name	Price
1	01/01/2022	John	Doe	Hamburger	10.99
2	02/01/2022	Jane	Doe	Pizza	15.99
384	03/03/2023	Robbyn	Mynett	Vegetarian lasagna	13.3
3	03/01/2022	Michael	Smith	Sushi	18.99
4	04/01/2022	Rachel	Green	Steak	25.99
5	05/01/2022	Ross	Geller	Pad Thai	12.99
6	06/01/2022	Monica	Geller	Tacos	8.99
7	07/01/2022	Chandler	Bing	Salmon	22.99
8	08/01/2022	Phoebe	Buffay	Fried Rice	9.99
9	09/01/2022	Joey	Tribbiani	Pasta	11.99
10	10/01/2022	Ted	Mosby	Kebab	16.99
11	11/01/2022	Barney	Stinson	Soup	6.99
12	12/01/2022	Lily	Aldrin	BBQ Ribs	24.99
13	13/01/2022	Marshall	Eriksen	Falafel	7.99
14	14/01/2022	Robin	Scherbatsky	Pho	13.99
99	15/04/2023	Angelo	Tabrett	Caesar salad with chicken	22.1
506	10/02/2023	Livy	Cator	Bife a Cavalo	35.2

2-Report

```
SELECT I.I_Id as "INGREDIENT ID", I.I_Name as "INGREDIENT
NAME", I.I_Type as "INGREDIENT TYPE",

S.S_Company as "COMPANY SUPPLIER", S.S_Telephone as "TELEPHONE
SUPPLIER", S.S_Address as "ADDRESS SUPPLIER", ST.Quantity as
"QUANTITY"

FROM Ingredient I

JOIN Supplier S ON I.S_Company = S.S_Company

JOIN Stock ST ON I.I_Id = ST.I_Id

WHERE I.I_Name = &<name=ingredient_name

                  type= "STRING">;
```



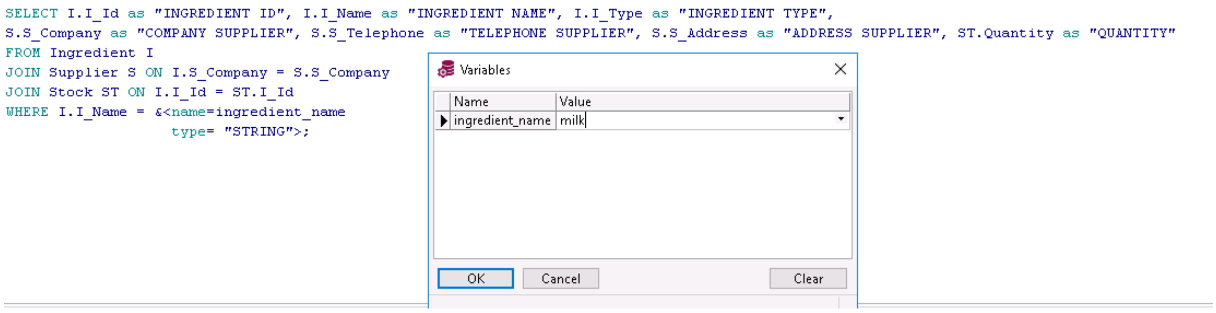
```
SELECT I.I_Id as "INGREDIENT ID", I.I_Name as "INGREDIENT NAME", I.I_Type as "INGREDIENT TYPE",
S.S_Company as "COMPANY SUPPLIER", S.S_Telephone as "TELEPHONE SUPPLIER", S.S_Address as "ADDRESS SUPPLIER", ST.Quantity as "QUANTITY"
FROM Ingredient I
JOIN Supplier S ON I.S_Company = S.S_Company
JOIN Stock ST ON I.I_Id = ST.I_Id
WHERE I.I_Name = &<name=ingredient_name
                   type= "STRING">;
```

Motivation

The motivation of the provided query is to generate a report that displays information about a specific ingredient, including its ID, name, type, supplier details, and the quantity available in stock.
This report can be useful for inventory management, procurement, or any scenario where you need to retrieve specific information about an ingredient, including its supplier details and stock availability. It allows you to track the ingredient's stock level and gather relevant information to support decision-making processes related to inventory, supplier management, or recipe planning.

Result

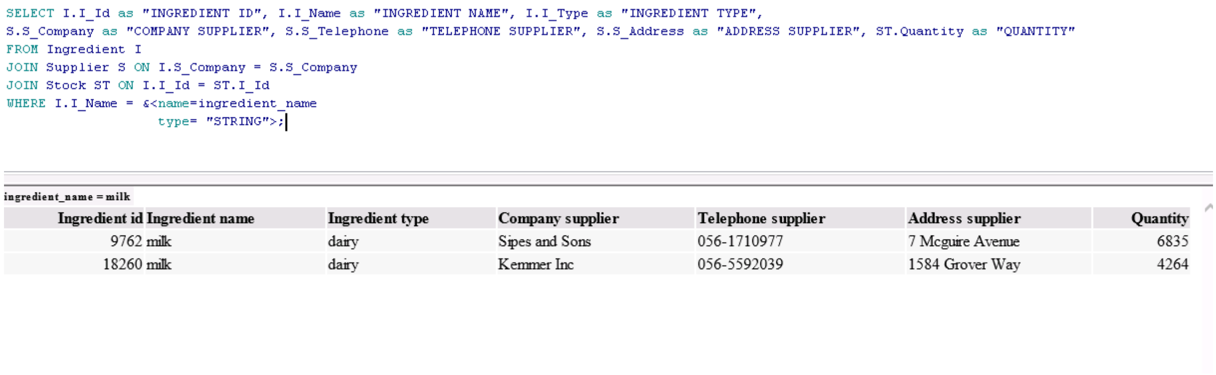
Entering the ingredient name



```
SELECT I.I_Id as "INGREDIENT ID", I.I_Name as "INGREDIENT NAME", I.I_Type as "INGREDIENT TYPE",
S.S_Company as "COMPANY SUPPLIER", S.S_Telephone as "TELEPHONE SUPPLIER", S.S_Address as "ADDRESS SUPPLIER", ST.Quantity as "QUANTITY"
FROM Ingredient I
JOIN Supplier S ON I.S_Company = S.S_Company
JOIN Stock ST ON I.I_Id = ST.I_Id
WHERE I.I_Name = &<name=ingredient_name
                   type= "STRING">;
```

Name	Value
ingredient_name	milk

The report output



```
SELECT I.I_Id as "INGREDIENT ID", I.I_Name as "INGREDIENT NAME", I.I_Type as "INGREDIENT TYPE",
S.S_Company as "COMPANY SUPPLIER", S.S_Telephone as "TELEPHONE SUPPLIER", S.S_Address as "ADDRESS SUPPLIER", ST.Quantity as "QUANTITY"
FROM Ingredient I
JOIN Supplier S ON I.S_Company = S.S_Company
JOIN Stock ST ON I.I_Id = ST.I_Id
WHERE I.I_Name = &<name=ingredient_name
                   type= "STRING">;
```

ingredient_name = milk

Ingredient id	Ingredient name	Ingredient type	Company supplier	Telephone supplier	Address supplier	Quantity
9762	milk	dairy	Sipes and Sons	056-1710977	7 Mcguire Avenue	6835
18260	milk	dairy	Kemmer Inc	056-5592039	1584 Grover Way	4264

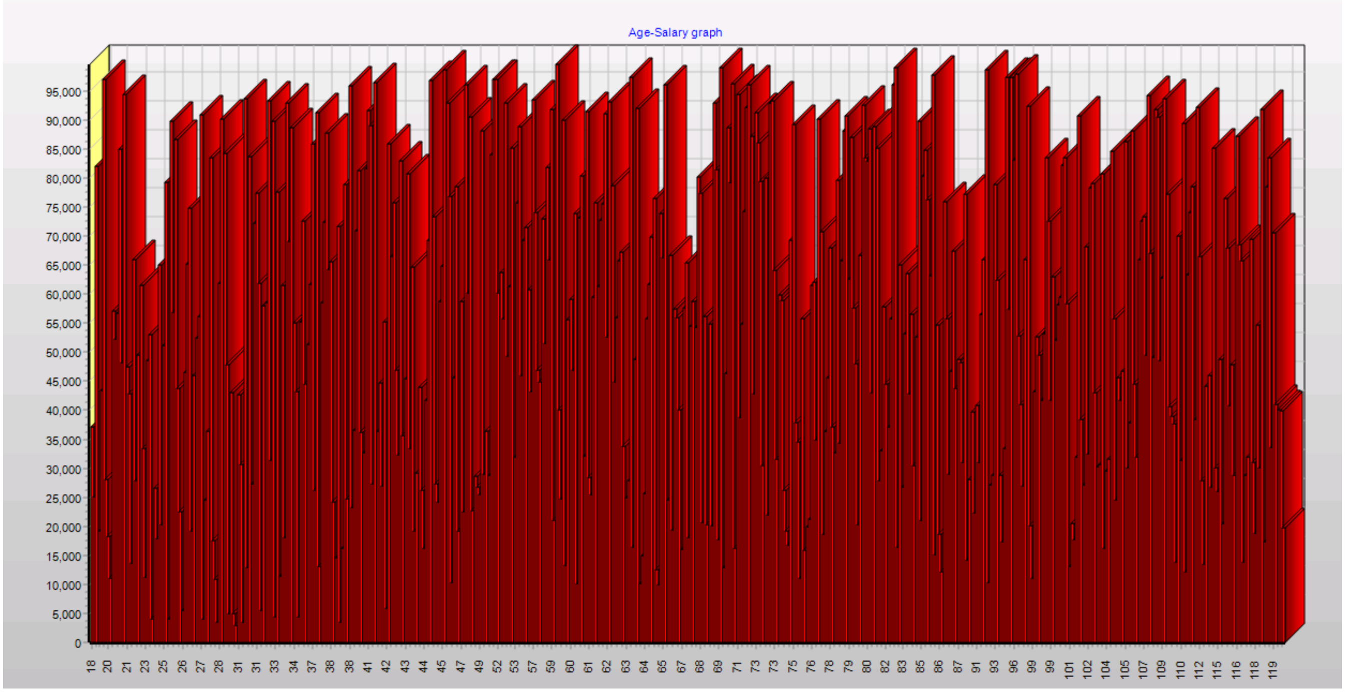
Graphs

1-Graph

Age-Salary Graph

The following 2D graph presents the relationship between Employee Age (x-axis) and Employee Salary (y-axis).

This graph can help visualize the distribution of salaries across different age groups in the organization. It is needed to analyze if there is any correlation between age and salary, identify any trends or patterns, and make informed decisions regarding salary structures, promotions, or workforce planning.



2-Graph

Type ingredient pie chart

The following pie chart represents the distribution of ingredients in stock for each type. The color of each piece represents the different types of ingredients: Deary, Parve and Meat, while the size of the piece represents the percentage or count of ingredients in stock for each type.

This pie chart is important as it provides a visual overview of the composition of the ingredient stock and highlights the proportion of each ingredient type in relation to the total stock. It helps in identifying the most common or abundant ingredient types and can assist in inventory management, procurement planning, and identifying potential shortages or excesses of specific ingredient types.

red - Meat

green - Parve

yellow - Dairy