Title: DB Assignment 2

Your Name: Deare

Date: 2024/9/19

Part1

- Create a database and import a CSV file:

CREATE database Assignmen2;-- Create new database for homework 2 named Assignment2
USE Assignmen2; -- Move into database
-How did I import csv file?
-- 1. created CSV files using text editor
-- 2. Entered the data, separating fields with commas (,).
    -- For example
    -- chefID,name,specialty
    -- 1,John Doe,Italian
-- 3. Saved the file with the .csv extension
-- 4. found my database in Navigator right click and choose 'table data import Wizard' to import my CSV file and right-click 'refresh all'.
Here are all the CSV files that I imported:

chefs.csv	2024/9/19 18:39	XLS Worksheet	1 KB
foods.csv	2024/9/19 18:54	XLS Worksheet	1 KB
restaurants.csv	2024/9/19 18:41	XLS Worksheet	1 KB
serves.csv	2024/9/19 18:56	XLS Worksheet	1 KB
works.csv	2024/9/19 18:53	XLS Worksheet	1 KB

-- **********
SHOW TABLES;-- Show all the table names that I imported in the database in assignment 2

```
-- ********************************************************************************
SHOW TABLES;-- show all the table names that I imported in database in assignment 2

-- ********************************************************************************
-- Due to when I import wizard as chefs.csv file did not mention to add PK
-- Therefore I alter the chefs table add primary key chefID and indicate primary key ch
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Tables_in_assignmen2
chefs
foods
restaurants
serves
works

- Add PK and FK

When I imported the wizard as chefs.csv file did not mention adding PK, Therefore I altered the chef's table added the primary key chefID, and indicated pimary key chefID CAN NOT be NULL. Even though I know the primary key in defult can not be null I insist on indicating my PK chefID can not be NULL.

-- ALTER TABLE chefs: modify the table chef

-- MODIFY chefID INT NOT NULL: ensures that the chefID column(attribute) is NOT NULL, meaning it cannot contain null values.

-- ADD PRIMARY KEY (chefID): adds the chefID attribute as the primary key for the chefs table.

-- **********

```
ALTER TABLE chefs
MODIFY chefID INT NOT NULL,
ADD PRIMARY KEY (chefID);
```

-- **********

-- Same reason as chefs entity, as given Rlation model the entity restaurants needs to add PK as restID

-- ALTER TABLE chefs: modify the table restaurants

-- MODIFY restID INT NOT NULL: ensures that the restID column(attribute) is NOT NULL, meaning it cannot contain null values.

-- ADD PRIMARY KEY (restID): adds the restID attribute as the primary key for the restaurants table.

-- **********

```
ALTER TABLE restaurants
MODIFY restID INT NOT NULL,
ADD PRIMARY KEY (restID);
```

-- **********

-- Same reason as above, as given Rlation model the entity foods needs to add PK as foodID

-- ALTER TABLE foods: modify the table foods

-- MODIFY foodID INT NOT NULL: ensures that the foodID column(attribute) is NOT NULL, meaning it cannot contain null values.

-- ADD PRIMARY KEY (foodID): adds the foodID attribute as the primary key for the foods table.

-- **********

```
ALTER TABLE foods
MODIFY foodID INT NOT NULL,
ADD PRIMARY KEY (foodID);
```

-- **********

-- ADD Foreign Key chefID in works Table: chefID in works references chefID in the chefs table

-- ADD Foreign Key restID in works Table: restID in works references restID in the

restaurants table

-- **********

ALTER TABLE works

ADD FOREIGN KEY (chefID) REFERENCES chefs(chefID),

ADD FOREIGN KEY (restID) REFERENCES restaurants(restID);

-- **********

-- ADD Foreign Key restID in serves Table: restID in serves references restID in the restaurants table

-- ADD Foreign Key foodID in serves Table: foodID in serves references foodID in the foods table

-- **********

ALTER TABLE serves

ADD FOREIGN KEY (restID) REFERENCES restaurants(restID),

ADD FOREIGN KEY (foodID) REFERENCES foods(foodID);

- Show the schema of all the table

-- **********

-- Show the schema of chefs, restaurants, works, foods, and serves

-- To double-check their PK and FK

-- **********

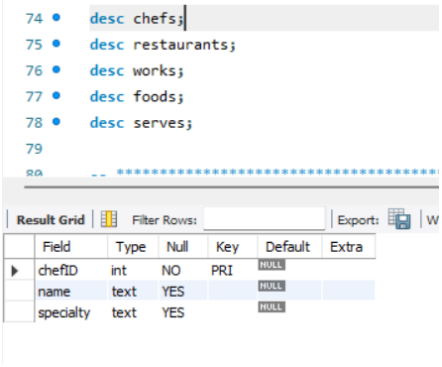
desc chefs;

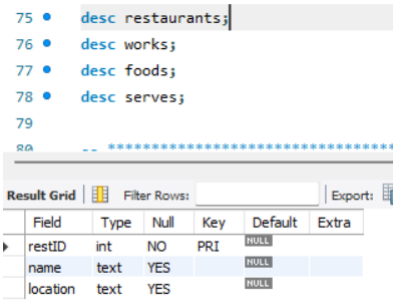
desc restaurants;

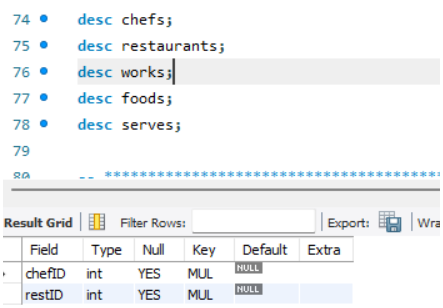
desc works;

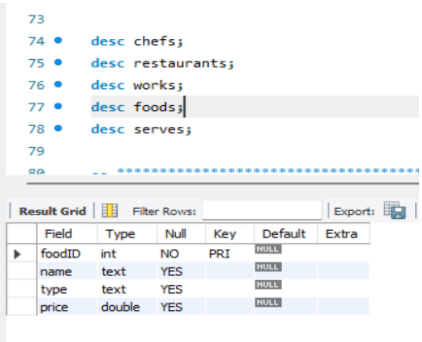
desc foods;

desc serves;









```
desc chefs;
desc restaurants;
desc works;
desc foods;
desc serves;
```

Result Grid | Filter Rows: | Export: | Wrap

Field	Type	Null	Key	Default	Extra
restID	int	YES	MUL	NULL	
foodID	int	YES	MUL	NULL	
date_sold	text	YES		NULL	

-- **********

-- Show the table with its attribute and its value that I using MySQL Workbench's Import Wizard which are

-- chefs.csv

-- restaurants.csv

-- works.csv

-- foods.csv

-- serves.csv

-- Because I want to check all the tables with their values.

-- **********

SELECT * FROM chefs;

SELECT* FROM foods;

SELECT * FROM restaurants;

SELECT* FROM serves;

SELECT* FROM works;

```
SELECT * FROM chefs;
SELECT* FROM foods;
SELECT * FROM restaurants;
SELECT* FROM serves;
SELECT* FROM works;
-- ****************************
-- PART 2
-- ****************************
```

Result Grid | Filter Rows:

chefID	name	specialty
1	John Doe	Italian
2	Jane Smith	Japanese
3	Alice Johnson	Mexican
4	Robert Brown	French
5	Emily Davis	Robert Brown
6	Michael Wilson	Indian

```
SELECT * FROM chefs;
SELECT* FROM foods;
SELECT * FROM restaurants;
SELECT* FROM serves;
SELECT* FROM works;
-- ****************************
-- PART 2
-- ****************************
```

Result Grid | Filter Rows: | Edit:

foodID	name	type	price
1	Pasta	Italian	12
2	Lasagna	Italian	15
3	Sushi	Rice	10
4	Tempura	Seafood	14
5	Tacos	Mexican	8
6	Burrito	Mexican	11
7	Coq au Vin	French	18
8	Crepes	French	9
9	Pad Thai	Thai	13
10	Green Curry	Thai	11
11	Butter Chic...	Indian	12
12	Paneer Tikka	Indian	15

```
92 •  SELECT * FROM restaurants;
93 •  SELECT* FROM serves;
94 •  SELECT* FROM works;
95    -- ******************************
96    -- PART 2
97    -- ******************************
```

Result Grid | Filter Rows:

restID	name	location
1	La Trattoria	New York
2	Sushi Haven	San Francisco
3	Taco Town	Los Angeles
4	Bistro Paris	Paris
5	Thai Delight	Bangkok
6	Indian Spice	Delhi

```
93 •  SELECT* FROM serves;
94 •  SELECT* FROM works;
95    -- ************************
96    -- PART 2
97    -- ************************
```

Result Grid | Filter Rows:

restID	foodID	date_sold
1	1	2024-01-10
1	2	2024-01-11
2	3	2024-01-12
2	4	2024-01-13
3	5	2024-01-14
3	6	2024-01-15
4	7	2024-01-16
4	8	2024-01-17
5	9	2024-01-18
5	10	2024-01-19
6	11	2024-01-20
6	12	2024-01-21

```
90 •  SELECT * FROM chefs;
91 •  SELECT* FROM foods;
92 •  SELECT * FROM restaurants;
93 •  SELECT* FROM serves;
94 •  SELECT* FROM works;
95    -- ****************************
96    -- PART 2
97    -- ****************************
```

Result Grid | Filter Rows:

chefID	restID
1	1
2	2
3	3
4	4
5	5
6	6
1	3
2	1
3	4
4	2
5	6
6	5

-- **********

Part2

Question 1: Average Price of Food at Each Restaurant

Identify question	SQL Component	Explanation
Average Price of Food at Each Restaurant	SELECT r.name as restaurant_name, AVG(f.price) AS Average_Price	Selects the restaurant's name(r.name) the name column from the restaurant's table, AVG(f.price) aggregate that can get the average price of all the foods assign all this value as Average_Price
	FROM restaurants r	Identifies the 'restaurants' table for the query and

		uses 'r' as an alias for simpler referencing.
	JOIN serves s ON r.restID = s.restID	Join the 'restaurants' table with the 'serves' table using 'restID' ensures that the query retrieves only the rows where the foodID in the serves table matches the foodID in the foods table to link each restaurant to the foods it serves.
	JOIN foods f ON s.foodID = f.foodID	Join the 'serves' table with the 'foods' table using 'foodID' ensures that the query retrieves only the rows where the restID in the restaurant table matches the restID in the serves table to access the detailed data about the foods, including prices.
	GROUP BY r.restID, r.name	Group the results by each restaurant's ID and name to ensure that the average price is calculated for each restaurant separately.

Screenshot of the query results from Workbench:

```
SELECT r.name as restaurant_name, AVG(f.price) AS Average_Price
FROM restaurants r
JOIN serves s ON r.restID = s.restID
JOIN foods f ON s.foodID = f.foodID
GROUP BY r.restID, r.name;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

restaurant_name	Average_Price
La Trattoria	13.5
Sushi Haven	12
Taco Town	9.5
Bistro Paris	13.5
Thai Delight	12
Indian Spice	13.5

Q2:Maximum Food Price at Each Restaurant

Identify question	SQL Component	Explanation
This means find the maximum food price at each restaurant.	SELECT r.name AS restaurant_name, MAX(f.price) AS maximum_price	Selects the restaurant's name(r.name) and calculates the MAX(f.price) maximum price of the foods served at each restaurant.
	FROM restaurants r	Identifies the 'restaurants' table for the query and uses 'r' as an alias.
	JOIN serves s ON r.restID = s.restID	Joins the 'restaurants' table with the 'serves' table

		to link each restaurant to the food served there via the restaurant ID.
	JOIN foods f ON s.foodID = f.foodID	Joins the 'serves' table with the 'foods' table, matching on the food ID to retrieve prices for each food item.
	GROUP BY r.restID, r.name;	Group the results by restaurant ID and name to calculate the maximum price of food at each restaurant separately.

Screenshot of the query results from Workbench:

```
SELECT r.name AS restaurant_name, MAX(f.price) AS maximum_price
FROM restaurants r
JOIN serves s ON r.restID = s.restID
JOIN foods f ON s.foodID = f.foodID
GROUP BY r.restID, r.name;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

restaurant_name	maximum_price
La Trattoria	15
Sushi Haven	14
Taco Town	11
Bistro Paris	18
Thai Delight	13
Indian Spice	15

Q3:Count of Different Food Types Served at Each Restaurant

Identify question	SQL Component	Explanation
Means Count how many distinct food types are served at each restaurant.	SELECT r.name AS restaurant_name, COUNT(DISTINCT f.type) AS Number_OF_Different_Food_Types	Selects the restaurant's name(r.name) named as restaurant_name and count the distinct food types served at each restaurant COUNT(DISTINCT f.type), this function counts the number of distinct (unique) food types served at each restaurant. The DISTINCT keyword ensures that only count each food type once per restaurant.
	FROM restaurants r	Identifies the 'restaurants' table for the query and uses 'r' as an alias.
	JOIN serves s ON r.restID = s.restID	Joins the 'restaurants' table with the 'serves' table to associate each restaurant with the specific food servings, based on restaurant ID
	JOIN foods f ON s.foodID = f.foodID	Joins the 'serves' table with the 'foods' table to

		access the types of foods served, based on the food ID.
	GROUP BY r.restID, r.name;	group by the restaurant so the count of unique food types is calculated for each restaurant.

Screenshot of the query results from Workbench:

```
137 •  SELECT r.name AS restaurant_name, COUNT(DISTINCT f.type) AS Number_OF_Different_Food_Types
138    FROM restaurants r
139    JOIN serves s ON r.restID = s.restID
140    JOIN foods f ON s.foodID = f.foodID
141    GROUP BY r.restID, r.name;
142
143    -- ******************************************************************************
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

restaurant_name	Number_OF_Different_Food_Types
La Trattoria	1
Sushi Haven	2
Taco Town	1
Bistro Paris	1
Thai Delight	1
Indian Spice	1

Q4: Average Price of Foods Served by Each Chef

Identify question	SQL Component	Explanation
Calculate the average price of foods served at the restaurant where each chef works.	SELECT c.name AS chef_name, AVG(f.price) AS average_food_price	c.name: This selects the name of the chef from the chefs table (aliased as c). AVG(f.price): This calculates the average price of the food aliases output as average_food_price
	FROM chefs c	Identifies the 'chefs' table for the query and uses 'c' as an alias.
	JOIN works w ON c.chefID = w.chefID	This joins the chefs table (c) with the works table (w). The condition c.chefID = w.chefID ensures that only the chefs who are listed in the works table are included in the result.
	JOIN restaurants r ON w.restID = r.restID	This joins the works table (w) with the restaurants table (r), linking each chef to the restaurants where they work.
	JOIN serves s ON r.restID = s.restID	This joins the restaurants table (r) with the serves table (s), which records which food items are served at which restaurants.
	JOIN foods f ON s.foodID = f.foodID	This joins the serves table (s) with the foods table (f) using the foodID so that access the price of each food item served at the restaurant where the chef works.

	GROUP BY c.chefID, c.name;	groups the results by each chef's chefID and name, ensuring that the average food price is calculated per chef.

Screenshot of the query results from Workbench:

```
SELECT c.name AS chef_name, AVG(f.price) AS average_food_price
FROM chefs c
JOIN works w ON c.chefID = w.chefID
JOIN restaurants r ON w.restID = r.restID
JOIN serves s ON r.restID = s.restID
JOIN foods f ON s.foodID = f.foodID
GROUP BY c.chefID, c.name;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

chef_name	average_food_price
John Doe	11.5
Jane Smith	12.75
Robert Brown	12.75
Alice Johnson	11.5
Emily Davis	12.75
Michael Wilson	12.75

Q5:Find the Restaurant with the Highest Average Food Price

Identify question	SQL Component	Explanation
Calculate the Highest Average Food Price of the restaurant.	SELECT r.name AS restaurant_name, AVG(f.price) AS average_food_pricece	r.name reffers to the restaurant table name column aliases output as restaurant_name, AVG(f.price) This calculates the average price of the food aliases output as average_food_price.
	FROM restaurants r	Identifies the 'restaurants' table for the query and uses 'r' as an alias.
	JOIN serves s ON r.restID = s.restID	Joins the restaurant table (r) with the serves table (s) based on restID.
	JOIN foods f ON s.foodID = f.foodID	Joins the serves table with the foods table (f) based on foodID, to get food price information
	GROUP BY r.restID, r.name	Groups the results by restaurant ID and restaurant name.
	ORDER BY average_food_price DESC	Sorts the results in descending order of the average food price, so the restaurant with the highest average food price appears at the top.
	LIMIT 1;	limits the result to only one row, which will be the restaurant with the highest average food price.

Screenshot of the query results from Workbench:

```
172   -- ************************************************************************
173   SELECT r.name AS restaurant_name, AVG(f.price) AS average_food_price
174   FROM restaurants r
175   JOIN serves s ON r.restID = s.restID
176   JOIN foods f ON s.foodID = f.foodID
177   GROUP BY r.restID, r.name
178   ORDER BY average_food_price DESC
179   LIMIT 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

restaurant_name	average_food_price
La Trattoria	13.5

Extra Credit: Determine which chef has the highest average price of the foods served at the restaurants where they work. Include the chef's name, the average food price, and the names of the restaurants where the chef works. Sort the results by the average food price in descending order.

Identify question	SQL Component	Explanation
Find out which chef has the highest average food price at the restaurants where they work, list their names, the average prices, and the names of the restaurants.	SELECT c.name AS chef_name, AVG(f.price) AS average_food_price, GROUP_CONCAT(DISTINCT r.name) AS restaurant_names	c.name AS chef_name: Retrieves the name of the chef. AVG(f.price) AS average_food_price: calculate the average price of foods same as I mentioned above, GROUP_CONCAT(DISTINCT r.name): Concatenates the distinct restaurant names for each chef into a single string then alias as restaurant_names
	FROM chefs c	Identifies the 'chefs table for the query and uses 'c' as an alias.
	JOIN works w ON c.chefID = w.chefID	Joins the works table (w) with the chefls table (c) based on chefID. To associate chefs with their respective workplaces.
	JOIN restaurants r ON w.restID = r.restID	Joins the restaurant table with the works table (w) based on restID, linking chefs to the specific restaurants they work in.
	JOIN serves s ON r.restID = s.restID	Joins the serves(s) table with the works reataurant (r) based on restID, Connecting restaurants to the food items they serve.
	JOIN foods f ON s.foodID = f.foodID	Joins the serves table(s) with the foods table (f) based on foodID, to get food price information
	GROUP BY c.chefID, c.name	GROUP BY c.chefID, c.name:Ensures that the data is grouped by chef, so each chef's name appears

		once per entry in the output.
	ORDER BY average_food_price DESC;	Sort the results by the average food price in descending order. This ensures that the chef with the highest average food price appears at the top of the list.

Screenshot of the query results from Workbench:

```
SELECT c.name AS chef_name, AVG(f.price) AS average_food_price, GROUP_CONCAT(DISTINCT r.name) AS restaurant_names
FROM chefs c
JOIN works w ON c.chefID = w.chefID
JOIN restaurants r ON w.restID = r.restID
JOIN serves s ON r.restID = s.restID
JOIN foods f ON s.foodID = f.foodID
GROUP BY c.chefID, c.name
ORDER BY average_food_price DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

chef_name	average_food_price	restaurant_names
Jane Smith	12.75	La Trattoria,Sushi Haven
Robert Brown	12.75	Bistro Paris,Sushi Haven
Emily Davis	12.75	Indian Spice,Thai Delight
Michael Wilson	12.75	Indian Spice,Thai Delight
John Doe	11.5	La Trattoria,Taco Town
Alice Johnson	11.5	Bistro Paris,Taco Town