University of British Columbia, Vancouver

Department of Computer Science

CPSC 304 Project
Cover Page

Milestone #:____4

Date:_28, October 2021

Group Number:_____77

Name	Student Number	CS Alias (Use rid)	Preferred E-mail Address
Wenkai Cai	90787698	q6d3d	wenkai101cai@gmail.com
Heng Wu	31696198	v4b4c	black14369@gmail.com
Michelle Chen	65406415	u0f7w	hsiangyi1025@gmail.com

Repository Link:

Project Link:

Project Description:

In this project, my team designed a database application that models a fried chicken chain store focusing on the database of the order for tracking including the order number, the name of the meal, the total amount of the meal, and each items' price. There are three types of subclasses for meals and have their own attributes of sizes or flavors. Meals are sent to workstations, users can sign different meals to different workstations to allow division of work and improve efficiency. Each workstation has a dedicated staff that will bring the meal to each customer. Customers need to input their personal details before starting order and can choose to be a member or not.

At this point, we had already finished creating our Graphical User Interfaces (GUI) under the University of British Columbia computer science personal website, which has already been pre-load the tables and data in the database through our modified SQL script. Users can publish or check the data by following the instructions on our GUI.

The final schema was slightly different from previous milestones. M2 was made complicated to meet the requirements for the rubric. Our final version is from our first milestone and made some changes and corrections. We corrected the mistakes pointed out by our project TA. Another change we made was to add an extra attribute "ID" to the Meal entity and make it a PK, the reason for this is because meals have subclasses and each subclass has attributes to determine the details for meals. Making names as PK overlaps and makes the attributes in subclasses redundant. For the relationship tables, added ON DELETE CASCADE to all FK's making them more sensitive. For example, if we delete a tuple from the Orders table, then the tuples in Contains table that refer to Orders should also be deleted because it no longer exists.
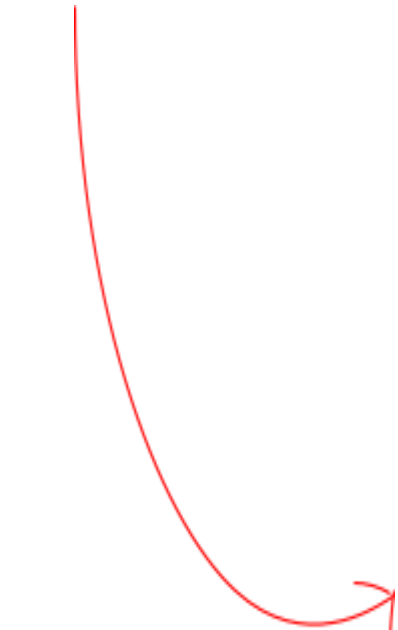
List of all SQL queries:

1. **INSERTION**: Operation for Customer data

2. **UPDATE:** Operation for Customer information

3. **DELETION:** Operation for Orders

4. **SELECTION:** Operation for Find meal ID

5. **PROJECTION:** Operation for List Meal data

6. **JOIN:** Operation for Order details

7. **Aggregation with Group By:** Count total Staff with store that has at least one Staff

    Aggregation with Group By: for finding number of staff served customer with phone number 6040000026

8. **Aggregation with Having:** Find Orders that ordered more than 5 Meals

9. **Nested Aggregation with Group By:** for finding the price of the smallest Meal with price greater than input

    Nested Aggregation with Group By: for find the total number of customer that have the same Member_Id 1000

10. **DIVISION:** display orders number that ordered all meals

Screenshots of the sample output of the queries using the GUI:

1. **INSERTION:** Operation for Customer data:
function insertCustomer() at line 333

Retrieved data from Customer:

Phone	Member ID	Name
6040000000		Alex Wang
6040000001	1000	Donata Manuel
6040000002	2000	Vipin Widukind
6040000003	3000	Elke Margarita
6040000004	4000	Praveena Dina
6040000005	5000	Donald Trump
6040000006		Pumpkin Jee
6040000007	1000	Mona Chen
6040000008	2000	Jay Haris
6040000009	3000	Hicho Mayo
6040000010	4000	Takeshi Naoki
6040000011	5000	Steve Jobs
6040000012		Mona Chen
6040000013	1000	Uri Oha
6040000014	2000	Samara Hip
6040000015	3000	Part timne
6040000016	4000	Praveena Dina
6040000017	5000	Jay Haris
6040000018		Joseph Sale
6040000019		Harry Hong
6040000020	1000	Jonas Neubauer
6040000021	1000	Jack Jore
6040000022	4000	Macoola Mattata
6040000023	4000	Jason Kim
6040000024	5000	Jeff Moore
6040000025		John Hopkins
6040000026		Steve Boi
6040000027	2000	Doge Coin
6040000028	2000	Baby Doge
6040000029	4000	Shib Shib

Insert

Phone: 6046046046
Member ID: 4000
Name:Liang Amy

INSERT: Customer data

Phone: `6046046046`

Member: `4000`

Name: `Liang Amy`

`Insert`

Retrieved data from Customer:

Phone	Member ID	Name
6040000000		Alex Wang
6040000001	1000	Donata Manuel
6040000002	2000	Vipin Widukind
6040000003	3000	Elke Margarita
6040000004	4000	Praveena Dina
6040000005	5000	Donald Trump
6040000006		Pumpkin Jee
6040000007	1000	Mona Chen
6040000008	2000	Jay Haris
6040000009	3000	Hicho Mayo
6040000010	4000	Takeshi Naoki
6040000011	5000	Steve Jobs
6040000012		Mona Chen
6040000013	1000	Uri Oha
6040000014	2000	Samara Hip
6040000015	3000	Part timne
6040000016	4000	Praveena Dina
6040000017	5000	Jay Haris
6040000018		Joseph Sale
6040000019		Harry Hong
6040000020	1000	Jonas Neubauer
6040000021	1000	Jack Jore
6040000022	4000	Macoola Mattata
6040000023	4000	Jason Kim
6040000024	5000	Jeff Moore
6040000025		John Hopkins
6040000026		Steve Boi
6040000027	2000	Doge Coin
6040000028	2000	Baby Doge
6040000029	4000	Shib Shib
6046046046	4000	Liang Amy

2. **UPDATE:** Operation for Customer information
function updateCustomer() at line 422

Before:	Insert	After:

Before:

Retrieved data from Customer:

Phone	Member	Name
6040000000		Alex Wang
6040000001	1000	Donata Manuel
6040000002	2000	Vipin Widukind
6040000003	3000	Elke Margarita
6040000004	4000	Praveena Dina
6040000005	5000	Donald Trump
6040000006		Pumpkin Jee
6040000007	1000	Mona Chen
6040000008	2000	Jay Haris
6040000009	3000	Hicho Mayo
6040000010	4000	Takeshi Naoki
6040000011	5000	Steve Jobs
6040000012		Mona Chen
6040000013	1000	Uri Oha
6040000014	2000	Samara Hip
6040000015	3000	Part timne
6040000016	4000	Praveena Dina
6040000017	5000	Jay Haris
6040000018		Joseph Sale
6040000019		Harry Hong
6040000020	1000	Jonas Neubauer
6040000021	1000	Jack Jore
6040000022	4000	Macoola Mattata
6040000023	4000	Jason Kim
6040000024	5000	Jeff Moore
6040000025		John Hopkins
6040000026		Steve Boi
6040000027	2000	Doge Coin
6040000028	2000	Baby Doge
6040000029	4000	Shib Shib
1		1
6046046046	4000	Liang Amy

Insert

Old Phone: 6046046046 New Phone: 6040000030

UPDATE: Customer information

The values are case sensitive and if you enter in the v

Old Phone: [6046046046]

New Phone: [6040000030]

[Update]

After:

Retrieved data from Customer:

Phone	Member	Name
6040000000		Alex Wang
6040000001	1000	Donata Manuel
6040000002	2000	Vipin Widukind
6040000003	3000	Elke Margarita
6040000004	4000	Praveena Dina
6040000005	5000	Donald Trump
6040000006		Pumpkin Jee
6040000007	1000	Mona Chen
6040000008	2000	Jay Haris
6040000009	3000	Hicho Mayo
6040000010	4000	Takeshi Naoki
6040000011	5000	Steve Jobs
6040000012		Mona Chen
6040000013	1000	Uri Oha
6040000014	2000	Samara Hip
6040000015	3000	Part timne
6040000016	4000	Praveena Dina
6040000017	5000	Jay Haris
6040000018		Joseph Sale
6040000019		Harry Hong
6040000020	1000	Jonas Neubauer
6040000021	1000	Jack Jore
6040000022	4000	Macoola Mattata
6040000023	4000	Jason Kim
6040000024	5000	Jeff Moore
6040000025		John Hopkins
6040000026		Steve Boi
6040000027	2000	Doge Coin
6040000028	2000	Baby Doge
6040000029	4000	Shib Shib
1		1
6040000030	4000	Liang Amy

3. DELETION: Operation for Orders
function deleteOrders() at line 367

Retrieved data from Orders:

Order Number	Phone
500	6040000000
501	6040000001
502	6040000002
503	6040000003
504	6040000004
505	6040000005
506	6040000006
507	6040000007
508	6040000008
509	6040000009
510	6040000010
511	6040000011
512	6040000012
513	6040000013
514	6040000014
515	6040000015
516	6040000016
517	6040000017
518	6040000018
519	6040000019
520	6040000020
521	6040000021
522	6040000022
523	6040000023
524	6040000024
525	6040000025
526	6040000000
527	6040000001
528	6040000004
529	6040000004
530	6040000009
531	6040000009
532	6040000009
533	6040000014
534	6040000015

Retrieved data from Contains:

Order Number	Meal ID
500	1
500	7
500	15
500	16
500	37
501	2
501	17
501	18
501	38
502	3
502	8
502	20
502	39
503	4

Delete order with order number: 501

DELETE: Orders

The values are case sensitive and if you enter

Order Number: [501]

[Insert]

Retrieved data from Orders:

Order Number	Phone
500	6040000000
502	6040000002
503	6040000003
504	6040000004
505	6040000005
506	6040000006
507	6040000007
508	6040000008
509	6040000009
510	6040000010
511	6040000011
512	6040000012
513	6040000013
514	6040000014
515	6040000015
516	6040000016
517	6040000017
518	6040000018
519	6040000019
520	6040000020
521	6040000021
522	6040000022
523	6040000023
524	6040000024
525	6040000025
526	6040000000
527	6040000001
528	6040000004
529	6040000004
530	6040000009
531	6040000009
532	6040000009
533	6040000014
534	6040000015

Retrieved data from Contains:

Order Number	Meal ID
500	1
500	7
500	15
500	16
500	37
502	3
502	8
502	20
502	39
503	4

4. **SELECTION:** Operation for Find meal ID

function handleFindMeal() at line 448

	Insert	
Display the Member table Submit Meal ID found **ID**	Meal Name: Coke Meal Price: 4 **SELECTION: Find meal ID** Insert meal name and a minimum price to find Meal Name: Coke Meal Price: 4 Submit	**Display the Member table** Submit Meal ID found **ID** 17 20 23

5. **PROJECTION:** Operation for List Meal data
function handleProjection() at line 470

Before:

Data from Meal

meal_id	meal_name	meal
1	Fried Whole Chicken	15
2	Fried Whole Chicken	15
3	Fried Whole Chicken	14
4	Fried Whole Chicken	17
5	Fried Whole Chicken	18
6	Fried Whole Chicken	17
7	Chicken Wings	15
8	Chicken Wings	15
9	Chicken Wings	14
10	Chicken Wings	13
11	Chicken Strips	10
12	Chicken Strips	10
13	Chicken Strips	11
14	Chicken Strips	12
15	Coke	3
16	Coke	4
17	Coke	5
18	Coke	3
19	Coke	4
20	Coke	5
21	Coke	3
22	Coke	4
23	Coke	5
24	Sprite	3
25	Sprite	4
26	Sprite	5
27	Sprite	3
28	Sprite	4
29	Sprite	5
30	Green Tea	4
31	Green Tea	5
32	Green Tea	4
33	Green Tea	5
34	Juice	8
35	Juice	8
36	Juice	8
37	Original Fries	6
38	Original Fries	7
39	Original Fries	8
40	Poutine Fries	8
41	Poutine Fries	9
42	Poutine Fries	10

Insert
Select 1: meal_id
Select2: meal_name

PROJECTION: List Meal data

(meal_id, meal_name, meal_price)

Select 1: [meal_id]

Select 2: [meal_name]

Select 3: [ ]

[Submit]

After:

Data from Meal

meal_id	meal_name
1	Fried Whole Chicken
2	Fried Whole Chicken
3	Fried Whole Chicken
4	Fried Whole Chicken
5	Fried Whole Chicken
6	Fried Whole Chicken
7	Chicken Wings
8	Chicken Wings
9	Chicken Wings
10	Chicken Wings
11	Chicken Strips
12	Chicken Strips
13	Chicken Strips
14	Chicken Strips
15	Coke
16	Coke
17	Coke
18	Coke
19	Coke
20	Coke
21	Coke
22	Coke
23	Coke
24	Sprite
25	Sprite
26	Sprite
27	Sprite
28	Sprite
29	Sprite
30	Green Tea
31	Green Tea
32	Green Tea
33	Green Tea
34	Juice
35	Juice
36	Juice
37	Original Fries
38	Original Fries
39	Original Fries
40	Poutine Fries
41	Poutine Fries
42	Poutine Fries

6. JOIN: Operation for Order details

function handleOrderDetails() at line 527

	Insert Order Number: 531	
Order details **Number ID Name Price**	**JOIN: Order details** Find the orders details by order number, allowing users to know the price and items Order Number: 531 Submit	Order details **Number ID Name Price** 531 2 Fried Whole Chicken 15 531 15 Coke 3 531 41 Poutine Fries 9

7. Aggregation with Group By: Count total Staff with store that has at least one Staff

function handleGroupByTotalStaff() at line 547

	Click 'submit'	
Total Staff **Store ID Total Staff**	**Aggregation with Group By: Count total Staff with store that has at least one Staff** Submit	Total Staff **Store ID Total Staff** 1000 5 1004 7 1001 5 1002 5 1003 5

Aggregation with Group By: for finding number of staff served customer with phone number 6040000026

function handleGroupByServeBy at line 564

	Click 'submit'	
Customer been served by #staff **CustomerPhone #Staff**	**Aggregation with Group By: Number of staff serve this customer** Find number of staff served customer with phone number 6040000026. Submit	Customer been served by #staff **CustomerPhone #Staff** 6040000026 2

8. **Aggregation with Having:** Find Orders that ordered more than 5 Meals

function handleGroupByHavingItemsOrdered() at line 580

Before:	Click 'submit'	After:
Items Ordered **Order Number Total Meals**	**Aggregation with Having: Find Orders that ordered more than 5 Meals** Submit	Items Ordered **Order Number Total Meals**

After data:

Order Number	Total Meals
507	9
510	9
512	10
514	6
533	42
534	42

9. **Nested Aggregation with Group By:** for finding the price of the smallest Meal with price greater than input

function handleNestedAggregationGroupBy() at line 597

Before:	Insert Price: 8	After:
Items Ordered **Name MIN**	**Nested Aggregation with Group By: MIN** Find the price of the smallest Meal with price>input, for each meal_n Price: 8 Submit	

After data:

Name	MIN
Chicken Wings	13
Chicken Strips	10
Fried Whole Chicken	14
Poutine Fries	9

Nested Aggregation with Group By: for find the total number of customer that have the same Member_Id 1000

function handleNestedGroup() at line 615

Before:	Click 'submit'	After:
Customer memberID	**Nested Aggregation with Group By** Find the total number of customer that have the same Member_Id 1000 Submit	**Customer memberID** 5 1000

10. **DIVISION:** with display orders that ordered all Meals

function handleDivisionOrderNumOrderedAllMeals() at line 633

DIVISION: Display Orders that ordered all Meals 送出	**DIVISION: Display Orders that ordered all Meals** 送出 --- **Display the Member table** 送出 **Number** 533 534