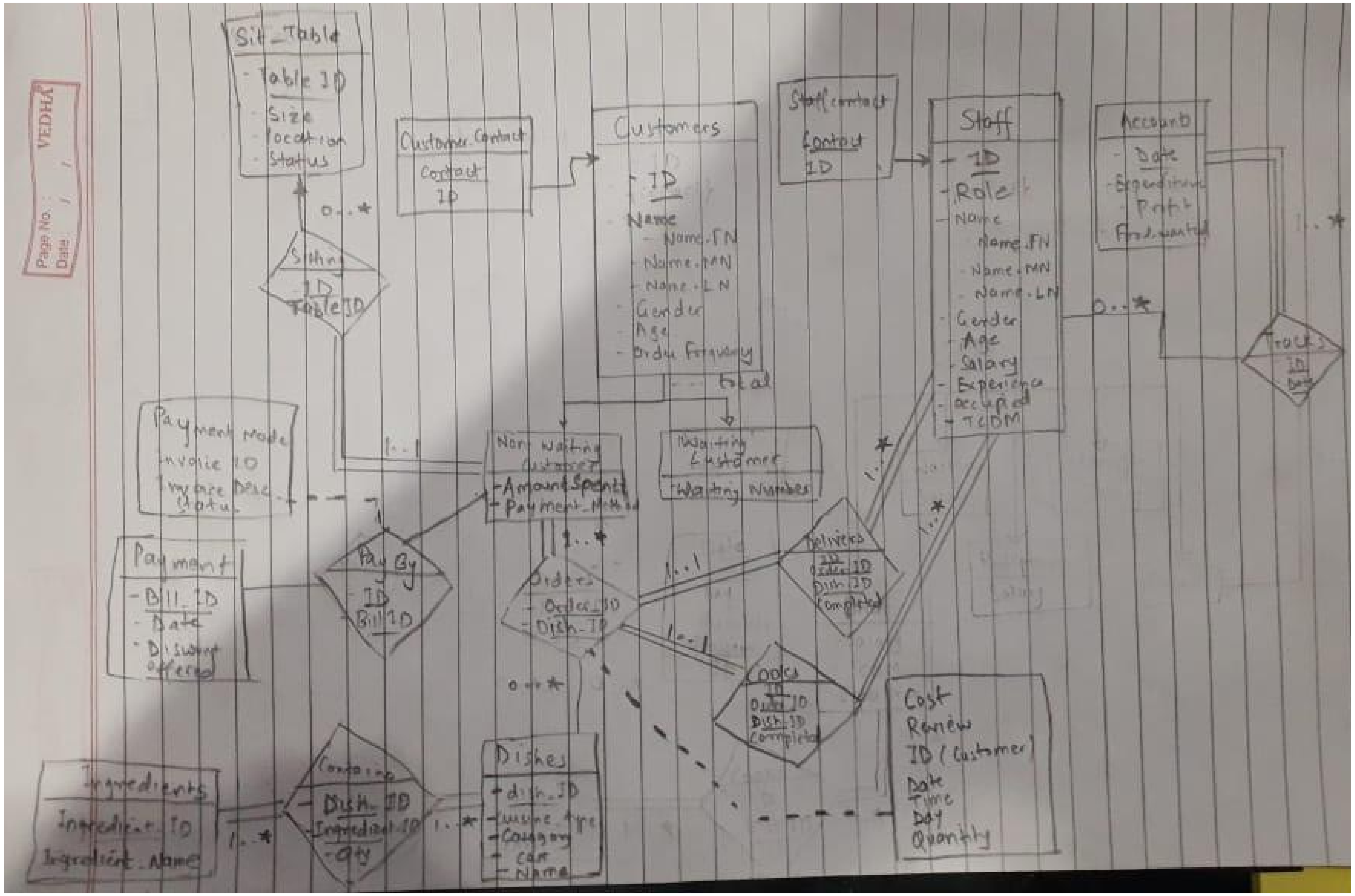
1. *Data model - Original Schema, normalised schema to 3NF/BCNF, additional constraints, triggers. If you do any denormalization after this for performance, document it. Specify DB indexes for your project along with reasons for each index (typically these involve the frequency and complexity of SQL queries hitting the DB balanced by the added cost to write to the DB). The complete DDL should be ready and presented as an appendix.*

   Following is our original schema. Since it is already in 3NF decomposed form, we need not decompose it further.

   The participation and involvement constraints are depicted and marked in the ER diagram. Other relational constraints of primary key, foreign key reference, not null constraints, integrity key constraints and other checks are mentioned in the Data Definition Language Script.

   All the required and essential triggers are mentioned in the triggers.sql file


Page No. : Date: / /
VEDHA
Sit_Table
Table ID
Size
location
Status
0..*
Sitting
ID
Table ID
Customer Contact
Contact
ID
Customers
ID
Name
Name.FN
Name.MN
Name.LN
Gender
Age
Order Frequency
total
Staff contact
Contact
ID
Staff
ID
Role
Name
Name.FN
Name.MN
Name.LN
Gender
Age
Salary
Experience
occupied
TCDM
Account
Date
Expenditure
Profit
Food wanted
1..*
0..*
Tracks
ID
Date
Payment Mode
Invoice ID
Invoice Desc
Status
1..1
Non waiting Customer
Amount Spent
Payment Method
Waiting Customer
Waiting Number
1..*
1..*
1..*
Payment
Bill_ID
Date
Discount offered
Pay By
ID
Bill ID
Orders
Order_ID
Dish_ID
1..1
1..1
Delivers
ID
Order ID
Dish ID
Completed
Cooks
ID
Dish ID
Dish ID
Completed
0..*
Cost
Review
ID (Customer)
Date
Time
Day
Quantity
Ingredients
Ingredient ID
Ingredient Name
1..*
Contains
Dish_ID
Ingredient ID
Qty
1..*
Dishes
dish_ID
cuisine type
Category
cat
Name

Since the Staff entity is involved in 3 relations (Cooks, Delivers and Tracks) of which 2(Cooks and Delivers) will be queried for frequently, we index the Staff relation based on the ID. We set the IDs incrementally starting from 1. Thus, the Staff relation will completely be stored in a sorted order based on the IDs. Thus, whenever this relation is queried using IDs, it will take very less time to fetch the required rows.

ID for Staff

We use have same motivation and reasoning as Staff to add following attributes as Database indices for the corresponding tables:

Dish_ID for Dishes

Ingredient_ID for Ingredients

ID for Customers

Table_ID for Sit_Table

Bill_ID for Payment

Since the attribute added as a Database Index in all the above cases is actually present in the corresponding table initially itself, the cost of adding DB index is 0.

Since Order will be queried frequently in the system, we need to add a DB index to it. However, since the primary key of Orders is a (int,int) pair which cannot be effectively converted into a scalar index, we need to introduce a New DB index for the same.

Order Serial Number for Orders

There will be a non-zero cost incurred while adding a DB index for Orders but the advantage of adding this DB index in query processing and retrieval will outweigh the cost.

2. *Data Generation and loading - specify your initial data set (generated or downloaded) and the data load scripts. If you are generating data, specify the parameters of data generation and the specific set of values that you picked for data generation, if you are selecting a subset of downloaded data, specify how and why you picked that subset in terms of parameters and their values.*

   We will be generating the dataset using DDL scripts. Following are the different parameters of data generation of our initial datasets:

   Accounts -> (Date,Restaurant_profit, Expenditure, Total_food_wasted (in kgs))

   Date: 01-01-2016 to 01-01-2021

   Restaurant_profit: (-5000) Rs to 10000 Rs

   Expenditure: 50000 Rs to 75000 Rs

   Total_food_wasted: 50 kg to 500 kg

   Staff -> (ID, {contact}, name.FN(FirstName),name.MN(middle_name),name.LN(last_name),gender,Age, Salary, Experience, Occupied, TCDM, Role)

   ID: 1 to 50

   {Contact}: set of 10 digit strings

   Name.FN: random strings or NULL

   Name.MN: random strings or NULL

   Name.LN: random strings or NULL

   Gender: Male / Female / Prefer not to say

   Age: 5 to 60

   Salary: 20000 Rs to 200000 Rs

   Experience: 2 yrs to 20 yrs

   Occupied: 0 (if Free) else 1(If Occupied)

   TCDM: 0 to 10000

   Role: The profession name (String)

Dishes -> (Dish_ID, cuisine(Chinese, Continental,...), Category(Starter, Main, Dessert, etc), cost, name)

Dish_ID: 1 to 30

Cuisine: Chinese / Italian / Continental / South Indian / North Indian

Category: Starter / Main / Dessert / Breakfast

Cost: 50 to 500

Name: name of the dish

Ingredients -> (Ingredient_ID, Quality)

Ingredient_ID: 1 to 20

Ingredient_Name: Name of the ingredient (String)

Contains -> (Ingredient_ID, Dish_ID, Quantity_used)

Ingredient_ID: 1 to 20

Dish_ID: 1 to 30

Quantity_used: 10 gm to 500 gm

Non_waiting customer: inherits Customers (Amount spent, Payment Method)

ID: 1 to 50

Amount spent: 500 to 2000

Payment Method: Credit Card / Debit Card / Cash / Others

Orders -> (order_ID, ID, Dish_ID, Date, Time, Day, Quantity_ordered, Review {1,2,3,4,5}, Cost)

Order_ID: 1 to 100

ID(of customer): 1 to 50

Dish_ID: 1 to 30

Date: 01-01-2016 to 01-01-2021

Time: 9:00 to 22:00

Day: Sun / Mon / … / Sat

Quantity_ordered: 1 / 2 / 3 / 4

Review: 1 / 2 / 3 / 4 / 5

Cost: 50 to 500

Cooks -> (ID,Order_ID, Dish_ID, Completed)

ID: 1 to 50

Order_ID: 1 to 100

Dish_ID: 1 to 30

Completed: 0 (If not Complete) or 1(If complete)

Customers -> (ID, {contact}, name.FN(FirstName),name.MN(middle_name),name.LN(last_name),gender,Age, Order_Frequency)

ID: 1 to 50

{Contact} = set of 10 digit strings

Name.FN: random strings or NULL

Name.MN: random strings or NULL

Name.LN: random strings or NULL

Gender: Male / Female / Prefer not to say

Age: 5 to 65

Order Frequency: 0 to 20

Payment -> (Bill_ID, Date, Discount_offered)

Bill_ID: 1 to 70

Date: 01-01-2016 to 01-01-2021

Discount_offered: 0-100%

Pay By -> (ID, Bill_ID, Payment_mode, Invoice_ID, Invoice_description, Status {“Paid”,”In progress”, “Failed”})

ID(of customer): 1 to 50

Bill_ID: 1 to 70

Payment_mode: Credit Card / Debit Card / Cash / Others

Invoice_ID: 1 to 30

Invoice_description: Summary of order

Status: Paid / In progress / Failed

Sit_Table -> (Table_ID,Size,Status{Ordering, Ordered but not eating, eating, finished}, location{“Window Side”, “Centre”, “Door Side”})

Table_ID: 1 to 10

Size: 2 / 3 / 4 / 5 / 6

Status: Ordering / Ordered but not eating / eating / finished

Location: Window Side / Centre / Door Side

Sitting -> (ID,Table_ID)

ID: 1 to 50

Table_ID: 1 to 10

Tracks -> (ID,Date)

ID: 1 to 50

Date: 01-01-2016 to 01-01-2021

Delivers -> (ID,order_ID, Dish_ID, completed)

ID: 1 to 50

Order_ID: 1 to 100

Dish_ID: 1 to 30

Completed: 0 (If not Complete) or 1(If complete)

Waiting Customer: inherits Customers (Waiting_Number)

ID: 1 to 50

Waiting_Number: 1 to 50

–

All of the above ranges are guidelines, to implement which, we are drawing variables from different probability distributions. Eg, the entry of customers in a restaurant is a poisson point process with some parameters. The daily operating expenditures and profits are gaussian distributed etc. Also, the higher the experience of a chef, the salaries that (s)he will receive are sampled from a higher set of values.

_

3. *Complete screen design for the screen corresponding to each of your use cases from project deliverable 2. How will your controller be built? There are 2 ways of building controllers - externalised where the flow from screen to screen will be driven by a specification that you can place and change outside of your program OR internal in which case the code to handle some button press will itself hardcode which screen will be generated. The former is obviously better since it's data driven but needs some other controller software such as Angular, Node.js etc.*

Screen Designs:

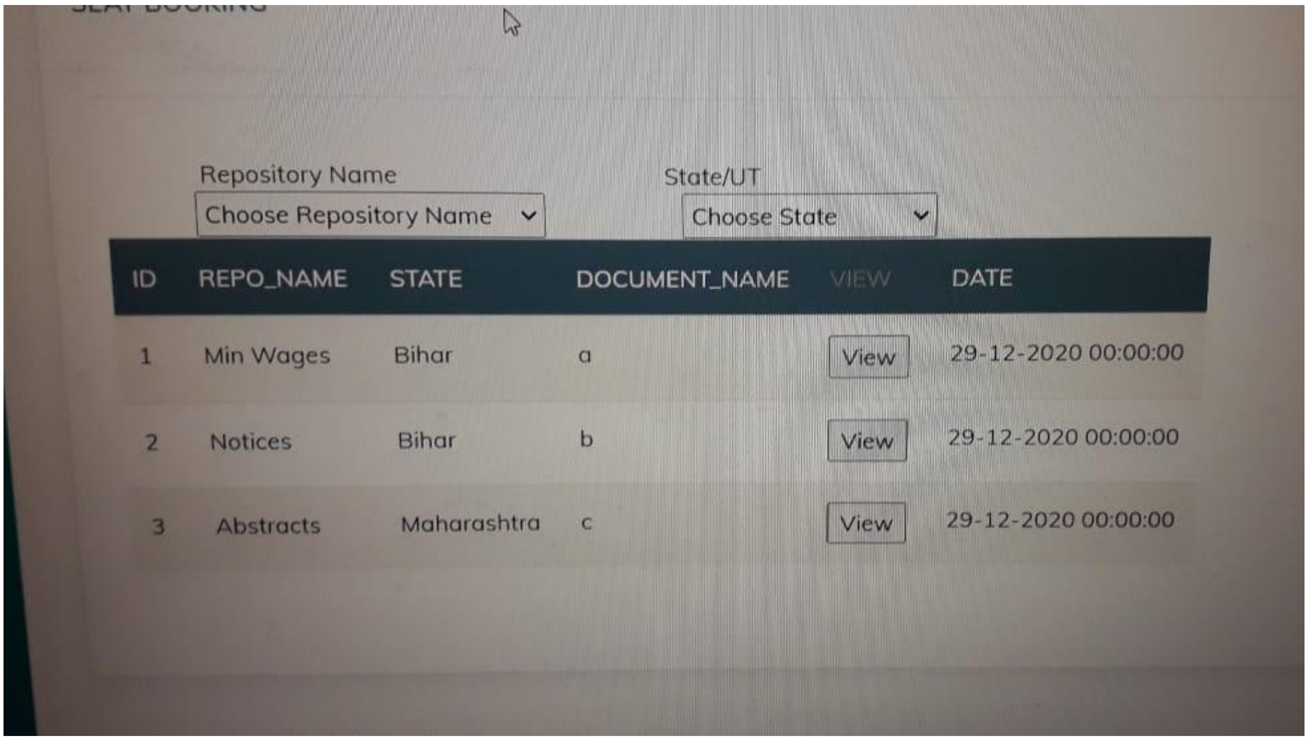
Search Engine:

It will have a text box in the frontend where the user can enter his query. Note that this is not a generic search engine but it’s domain is only that of the dishes (name/ID).

We will query the database in our controller logic based on the input from the textbox in the view and redirect to a result page whose get_test method of the controller will have the filtered results based on the search. If there are no results then the user will be notified and will not be redirected anywhere (data driven controller).

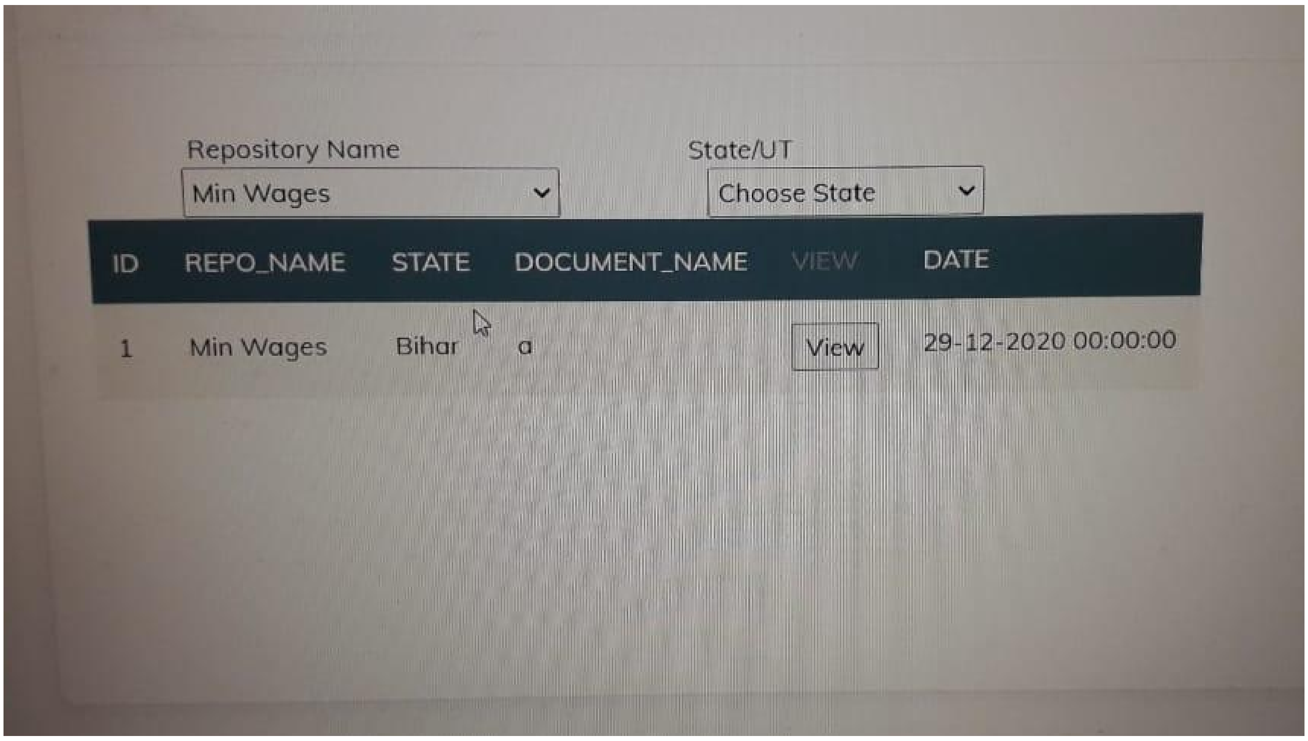
An alternative we are considering is to have multiple dropdowns based on cuisine, dish name, category, price range and filter the grid view menu display based on all these filters (using WHERE clause in SQL backend):

Sample shown below: (this was made for a different application but our screen design will be similar)


Repository Name: Choose Repository Name

State/UT: Choose State

ID	REPO_NAME	STATE	DOCUMENT_NAME	VIEW	DATE
1	Min Wages	Bihar	a	View	29-12-2020 00:00:00
2	Notices	Bihar	b	View	29-12-2020 00:00:00
3	Abstracts	Maharashtra	c	View	29-12-2020 00:00:00


Repository Name: Min Wages

State/UT: Choose State

ID	REPO_NAME	STATE	DOCUMENT_NAME	VIEW	DATE
1	Min Wages	Bihar	a	View	29-12-2020 00:00:00

Place an Order:

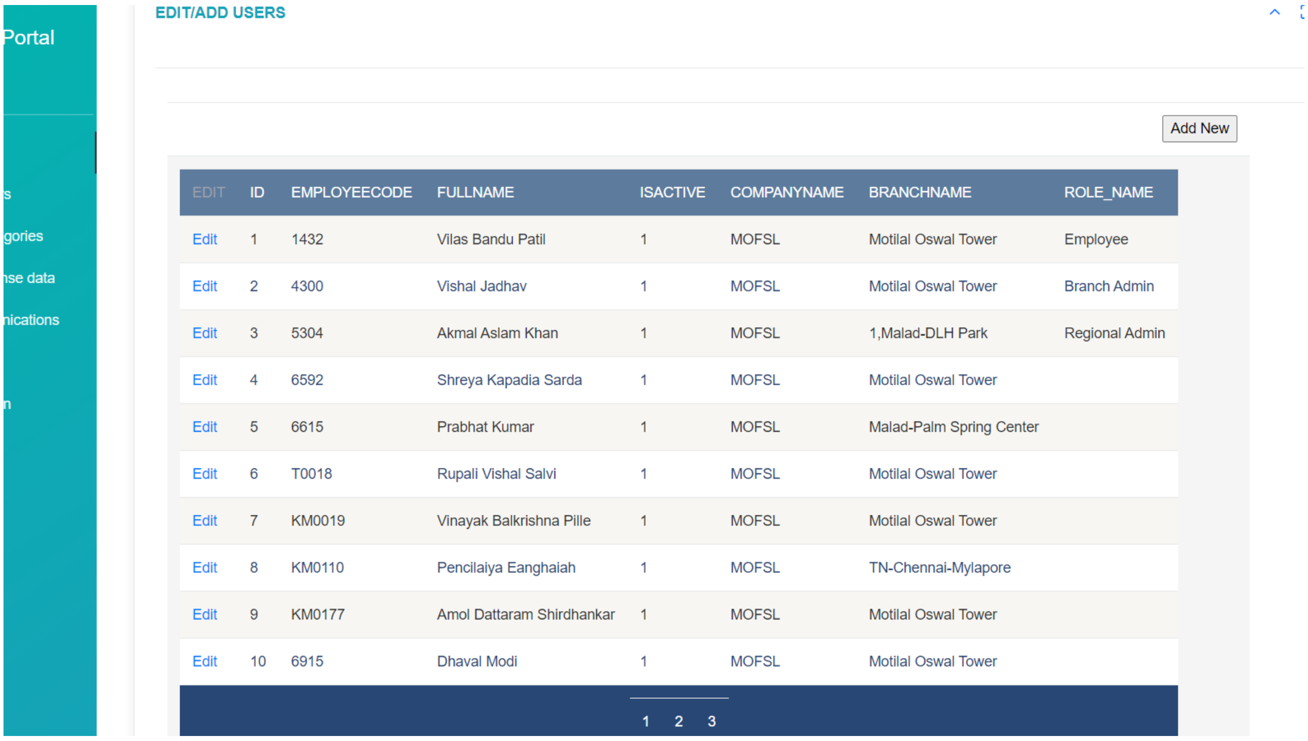
We will have a grid view of the menu with dish name, price and an order button on each row. We will have a row trigger for each row so if the user clicks on the order button of the row will be added to the order. We will also have a dropdown (with minimum of 1 to maximum of 5) in each row asking for the quantity (default will be 1). And there will be a view order button below the grid view on pressing which, user will be shown a popup with his quantities and orders and then on pressing the confirm order button it will be sent to the manager. The order immediately goes into the order relation also.

An alternative to the above UI is the card view which we saw in Lab 5. There will be cards of dishes in the menu page with price and image mentioned and each card will have the dropdown mentioned above and on clicking the order button in the card similar action as above will take place.

Manage Employee Information:

This is only visible to the admin. This will have a grid view of the database columns with each row having an update, create and delete option. (the CRUD feature of grid view). On selecting, editing a form will open up filled with the current values of the employee tuple and the admin can change the values and click the update button. For creation a similar form will be used.

An example is shown below: (for employee relation under control of an HR admin). Ours will be the same screen and form design but for inventory and employee update purpose. The website below is from one done at an internship.

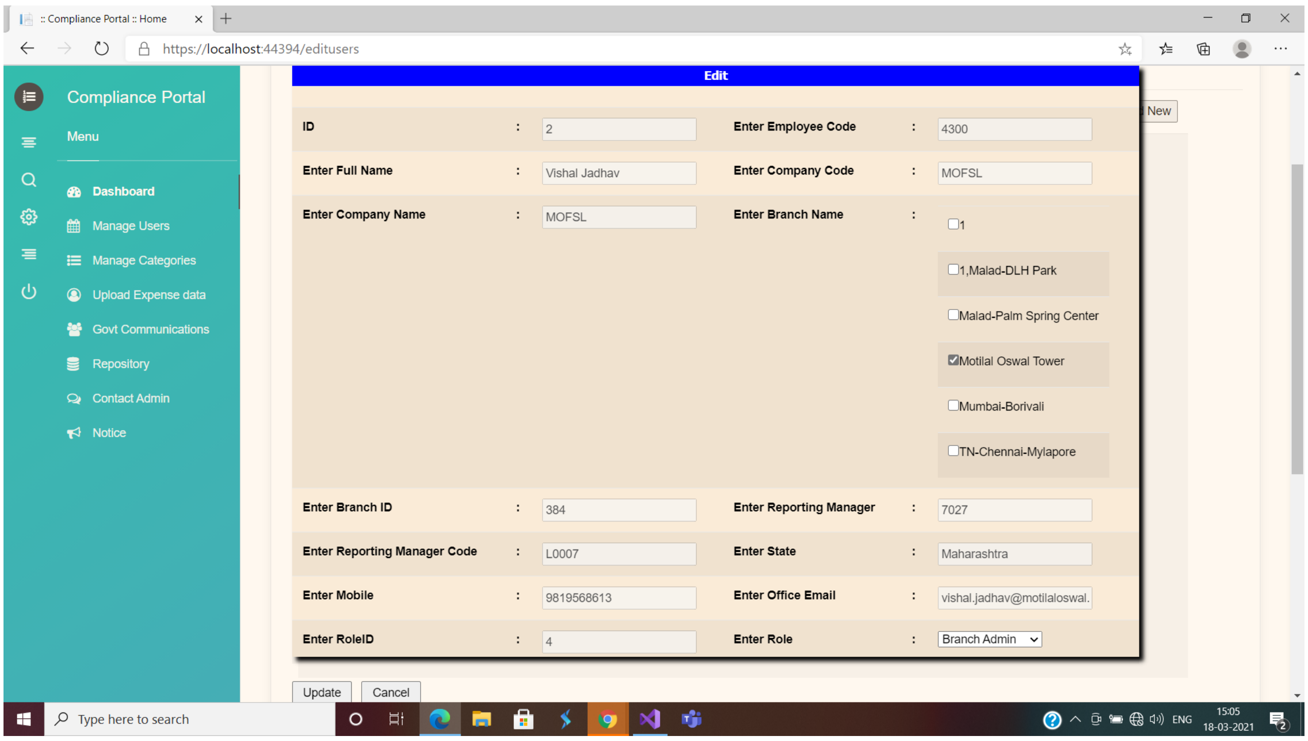


EDIT/ADD USERS

Add New

EDIT	ID	EMPLOYEECODE	FULLNAME	ISACTIVE	COMPANYNAME	BRANCHNAME	ROLE_NAME
Edit	1	1432	Vilas Bandu Patil	1	MOFSL	Motilal Oswal Tower	Employee
Edit	2	4300	Vishal Jadhav	1	MOFSL	Motilal Oswal Tower	Branch Admin
Edit	3	5304	Akmal Aslam Khan	1	MOFSL	1,Malad-DLH Park	Regional Admin
Edit	4	6592	Shreya Kapadia Sarda	1	MOFSL	Motilal Oswal Tower	
Edit	5	6615	Prabhat Kumar	1	MOFSL	Malad-Palm Spring Center	
Edit	6	T0018	Rupali Vishal Salvi	1	MOFSL	Motilal Oswal Tower	
Edit	7	KM0019	Vinayak Balkrishna Pille	1	MOFSL	Motilal Oswal Tower	
Edit	8	KM0110	Pencilaiya Eanghaiah	1	MOFSL	TN-Chennai-Mylapore	
Edit	9	KM0177	Amol Dattaram Shirdhankar	1	MOFSL	Motilal Oswal Tower	
Edit	10	6915	Dhaval Modi	1	MOFSL	Motilal Oswal Tower	

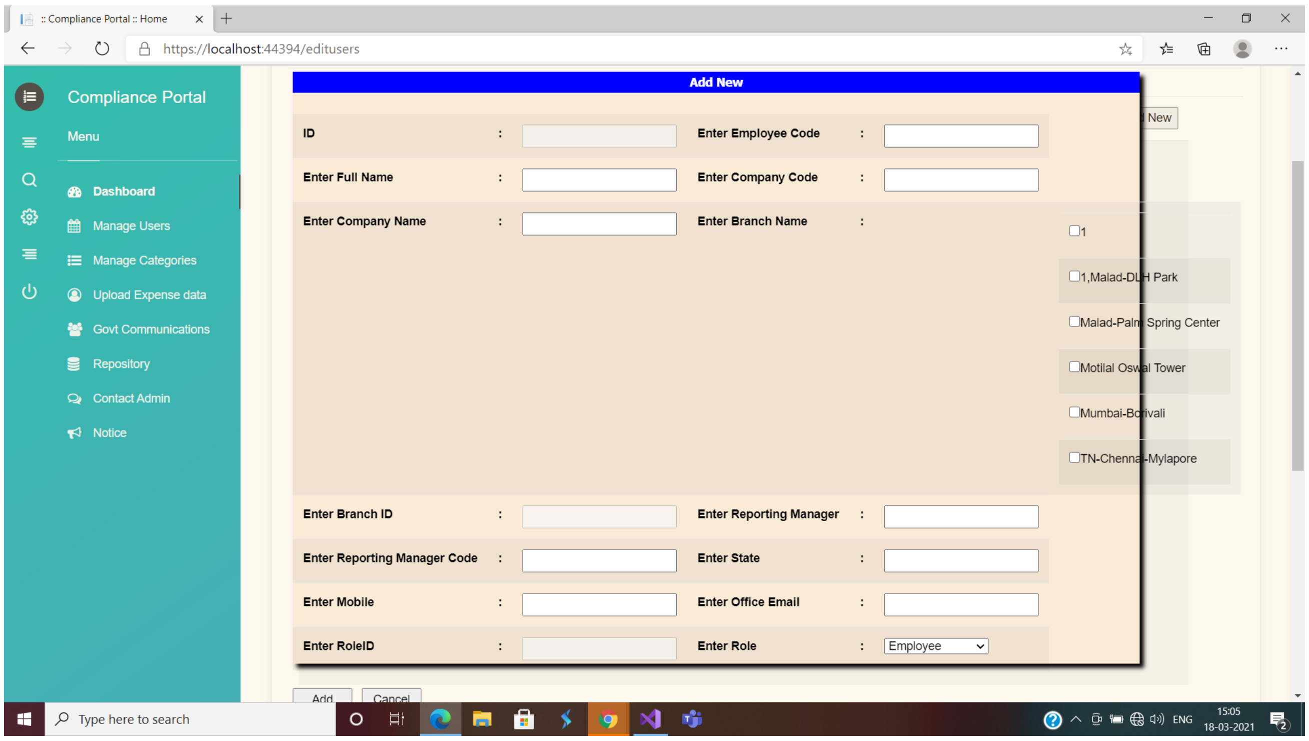
1 2 3



Edit

ID	:	2	Enter Employee Code	:	4300
Enter Full Name	:	Vishal Jadhav	Enter Company Code	:	MOFSL
Enter Company Name	:	MOFSL	Enter Branch Name	:	☐1 ☐1,Malad-DLH Park ☐Malad-Palm Spring Center ☑Motilal Oswal Tower ☐Mumbai-Borivali ☐TN-Chennai-Mylapore
Enter Branch ID	:	384	Enter Reporting Manager	:	7027
Enter Reporting Manager Code	:	L0007	Enter State	:	Maharashtra
Enter Mobile	:	9819568613	Enter Office Email	:	vishal.jadhav@motilaloswal.
Enter RoleID	:	4	Enter Role	:	Branch Admin

Update Cancel



Add New

ID	:		Enter Employee Code	:	
Enter Full Name	:		Enter Company Code	:	
Enter Company Name	:		Enter Branch Name	:	☐1 ☐1,Malad-DLH Park ☐Malad-Palm Spring Center ☐Motilal Oswal Tower ☐Mumbai-Borivali ☐TN-Chennai-Mylapore
Enter Branch ID	:		Enter Reporting Manager	:	
Enter Reporting Manager Code	:		Enter State	:	
Enter Mobile	:		Enter Office Email	:	
Enter RoleID	:		Enter Role	:	Employee

This will be done for all categories of employees (all who inherit from the people relation: the waiter, manager, owner, receptionist, chef). All the grid views will be in a single page one below the other.

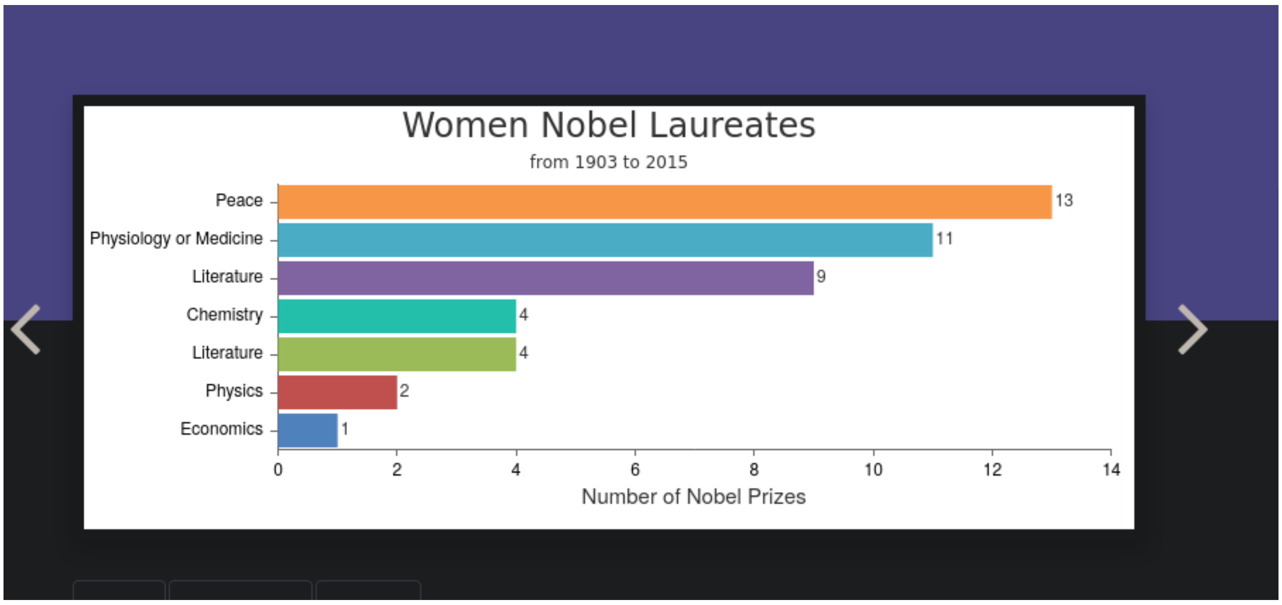
Analytics

This page will have statistics about the restaurant in mostly graph and chart format.

We will display the expenditure (based on ingredients purchased) and revenue generated on a daily basis. Food wasted in kgs will also be reported.

Then we will have a bar graph about the number of orders per dish to give an idea of which dishes are doing well and which are lagging behind (so as to promote them and maybe change a few things). The data for this will be fetched through aggregate functions and joins on the database relations. For charts we will use the Google Chart API/d3 in Nodejs or CanvasJS.Chart in ASP.NET MVC.

Our chart display will be such:



The other charts will be on attendance of employees, chefs with dishes of maximum rating and customers with number of orders.

Allot Order to Chef:

We will use a join on the dishes in the order and the chef with availability mentioned and each dish in the order will be allotted to a different chef if possible to increase parallelism. For every allotment there will be addition of a tuple in the cooks relation. Then we will simulate the cooks relation which implies the chef has started making the dish.
The UI will be a grid view on the dishes in the order and each row will have an allot button which will open up a form with the chef IDs and names in a dropdown and the manager can choose accordingly.

Order History:

Similar to Menu - either grid view or card view. We will show the dish ID, chef name and other attributes from the order relationship set such as date, time, quantity.

A grid view is very simple to implement since we directly expose a view on the order relation with some attributes hidden.

A card view is better for design since we can include images of the dish along with reviews in the individual cards. (long reviews in grid view is messy)

Order history page for each customer will be view-only.

Update Inventory:

This is similar to the update employee information and the similar UI. A CRUD and grid view on the inventory table with all columns since only the owner can see it. Some columns may be hidden from the frontend if required. Here the ingredients table and (maybe) the dishes are included in the inventory.

Serving Food:

This page is similar to allotting order to chef since the inherent function is the same.

A grid view of the dishes (join with orders) which are completed by the chef are displayed and the last column has a serve button which opens up a form with a dropdown to chose the waiter to serve and as soon as we click the serve button the corresponding order is considered completed and deleted from the orders relation.

Corresponding updates will be made in the backend to the employee performance table.

Automation:

In this Personalization page, we will be tracking customer behaviour and recommending items to the user based on the analytics details. A notification will be shown here if some ingredients have quantities below a threshold (set by the owner).

In the page, the top half will have a big heading saying URGENT ACTION NEEDED: and there we will be displayed the ingredients with quantities less than a given threshold (using a simple select query on the Ingredients relation).

Coming to the customer recommendations part so there are 2 ways:

-> Using the overall most popular dishes (in the Analytics section) - we will recommend the top k of those dishes (this will be dynamic and change since we will record both overall and every month’s most popular dishes) to every customer.

-> If the customer is regular we have stored his most frequently ordered dishes also and so they will also be displayed.

All things mentioned above will be visible in UI just as a list of bulleted items.

There will be a text box/ drop down to choose customers and after choosing them the output for the customer part will be filtered based on the input to the dropdown (using the WHERE clause in SQL). The common part to all customers (top k dishes) is always displayed and no filtering exists for that.

Controller Choice:

We will use the internalised way of building controllers like our Lab 5, since it is better suited with our design and way we have thought about our project till now and it takes different possibilities into account for the redirects. After doing Lab 5 we may use Node.js or our default of ASP.NET MVC. The data-driven part of this method will come from the backend database linked through our model. Code to handle some button press will itself hardcode which screen to be generated.

For example, in a search engine if no result is found we will remain on the same page else he will be taken to the results page which has further options of ordering/coming back to search.

Other examples where redirections fanout into many cases are (including error cases):

-> If the order list specified by the user has a dish not present in the dishes relation then it should remain in that page or display an error message. If the order is legal then it is sent to the manager for further processing and the user redirected to the home/menu page.

-> In allotting orders to the chef if we allot to an absent employee the app should remain on the same page maybe with an error popup and redirect to the home/employee section if successful allocation happens. Similar case if order allotted to an absent waiter.

-> Quantity negative in update inventory step leads to an error and staying on the same page else redirect to home page.

There are several other cases.

4. *Identify all transactions corresponding to each use case and write out the SQL for this.*

SEARCH:(input params: search_text)

```
cursor.execute(select dish_name from dishes where (cuisine contains search_text or
category contains search_text or name contains search_text))
```

OR

FILTER: (Input params: cuisine_filter, cat_filter, name_filter, cost_filter)

```
cursor.execute(select dish_name from dishes where (
(cuisine_filter IS NULL OR cuisine LIKE '%' + cuisine_filter + '%') and
(cat_filter IS NULL OR category LIKE '%' + cat_filter + '%') and
(name_filter IS NULL OR name LIKE '%' + name_filter + '%') and
(cost_filter IS NULL OR cost <= cost_filter)
))
```

Place an Order:
Insert_Order function -> Input params: (cust, dish, quant)

```
Import datetime
from datetime import date
import calendar
my_date = date.today()
day = calendar.day_name[my_date.weekday()]
time = datetime.datetime.now().time().strftime("%H,%M")
cost = cursor.execute(select cost from dish where dish_ID = dish);
cursor.execute(Insert into orders(ID, Dish_ID, Date, Time, Day, Quantity_ordered,
Review, Cost) VALUES(cust, dish, my_date, time, day, quant, 0, cost );)
```

Update_Review function -> Input params: (order, dish, review)

```
cursor.execute(Update orders set review = review where order_ID = order and
dish_ID = dish)
```

Order history (input param: start-date, end-date):

```
cursor.execute(Select * from orders where date>start and date<end);
```

Update inventory: (input params: ingredient_id, quant (can be +ve as well as -ve))
(check if the person is authenticated to update)

```
Current_quantity = cursor.execute(select quantity from Ingredients where
Ingredient_ID = id)
if(current_quantity+quantity<0){
        Raise Exception("Incorrect quantity");
}
Else{
        cursor.execute(update ingredients set quantity=quantity+quant where
ingredient_id = id );
}
```

Serving Food: (input params: order, dish, waiter, chef_id)

```
//Update cooks relation
cursor.execute(Update cooks set completed = 1 where order_id=order and ID=chef_id
and dish_ID=dish)

//Update delivers relation
cursor.execute(Insert into delivers(id, order_id, dish_id, complete) VALUES(waiter,
order, dish,0))

//Show this on waiter interface
cursor.execute(Select table_id, ID from sitting, orders where order_id=order and
dish_id = dish and orders.ID = sitting.ID).fetchone()

//Update TCDM function
Cost = cursor.execute(select cost from dish where dish_id = dish_id)
cursor.execute(Update staff set tcdm = tcdm+cost where ID = id;)
```

Analytics: (input params: NULL)

1) Day-wise Revenue: cursor.execute(select(sum(cost)) from orders group by day;)
2) Sales-corresponding to each dish: cursor.execute(Select name, count(*) from orders natural join dishes group by dish_ID;)
3) Dishes performing best and worst:
   cursor.execute(Select name from orders natural join dishes order by count(*) desc group by dish_ID limit 5 union Select name from orders natural join dishes order by count(*) asc group by dish_ID limit 5)
4) Employee performance:
   waiter best performers:
   cursor.execute(select id, tcdm from staff where role='waiter' order by tcdm desc limit 5;)
   Chef best performers:
   cursor.execute(select id, tcdm from staff where role='chef' order by tcdm desc limit 5;)
5) Monthly profits vs expenditure: cursor.execute(Select sum(profits), sum(expenditure) from accounts group by YEAR(date), MONTH(date))
6) Food wastage: cursor.execute(Select sum(Total_food_wasted) from accounts group by YEAR(date), MONTH(date))

MANAGE EMPLOYEE INFORMATION:
Update: (input: role, first_name, mid_name, last_name, age, salary, experience)
cursor.execute(Update staff set role=role, name.FN = first_name, name.MN = mid_name, name.LN = last_name, age = age, salary = salary, experience = experience)

Delete: (param: id)
cursor.execute(Delete from staff where id = id)

Insert: (input: role, first_name, mid_name, last_name, age, salary, experience, gender, contact)
cursor.execute(Insert into staff (contact, name.FN,name.MN,name.LN,gender,Age, Salary, Experience, Occupied, TCDM, Role) VALUES($1, $2,$3,$4,$5,$6,$7,$8,$9,$10,$11), [contact, first_name,mid_name, last_name, gender, age, salary, experience,0,0,role])

ALLOT ORDER TO CHEF: (input params: order, dish, chef)
cursor.execute(Insert into cooks(id, order_id, dish_id, completed) VALUES(chef, order, dish,0);)
//Update TCDM function
Cost = cursor.execute(select cost from dish where dish_id = dish_id);
cursor.execute(Update staff set tcdm = tcdm+cost where ID = id);

AUTOMATION:

POPULAR DISH RECOMMENDATION:

```
cursor.execute(“select * from orders natural join dishes order by
count(dish_ID) group by dish_ID limit 5”);
```

USER SPECIFIC RECOMMENDATION:
```
        cursor.execute(“select * from orders natural join dishes where ID = uid order
by count(dish_ID) group by dish_ID limit 5”);
```

INGREDIENTS YOU NEED TO ORDER:
```
        cursor.execute(“Select ingredient_ID from ingredients where quantity<10;”)
```