Databases and Information Systems Project

Restaurant Management System(RMS)

Final Report

Team Info

Name	Roll Number	Email-ID
Guttu Sai Abhishek	180050036	180050036@iitb.ac.in
Mulinti Shaik Wajid	180050063	180050063@iitb.ac.in
Sai Phanindra Ramasahayam	180050084	180050084@iitb.ac.in
Sanapathi Sumanth Balaji	180050091	180050091@iitb.ac.in

Requirements

Manager(M) should be able to

1. Manage ingredients
2. Manage items in menu
3. Know the current orders in restaurant
4. Know which tables are empty and which are occupied
5. Know what are popular dishes served in restaurant and popular day for customers to visit restaurant
6. Add new staff to restaurant
7. Know when amount of ingredients falls below threshold

Cashier(Ca) should be able to

1. Know the orders whose bill is not paid yet
2. Bill the order and accept money

Head waiter(H) should be able to

1. Order items on behalf of customers who can not order online
2. Manages orders and table booking requests

Customer(Cu) should be able to

1. Have an account
2. Order food online
3. Know which tables are booked
4. Know the dishes which the restaurant serves

5. Book a table
6. Know what were his/her previous orders
7. Know whether a item is veg/non-veg/spicy etc
8. Review and rate food items
9. Receive notification if his/her account gets credited with Rcoins

List of use cases

These use cases are in reference to the screen designs(/interfaces) we have given in previous deliverable-3:

1. Manager can add new ingredients
2. Manager can update ingredients
3. Manager can add new items and set its cost, tags etc
4. Manager can update existing items
5. Manager should be able to view current orders in restaurant
6. Manager should be able to see the current status of tables i-e whether they are occupied or not
7. Manager can view most popular dish in restaurant
8. Manager can view most popular day for customers to come to restaurant
9. Manager can add new staff or delete staff
10. Manager should receive notifications when the quantity of ingredients falls below threshold
11. Customer can create an account
12. Customer can log into account
13. Customer can change his personal details
14. Customer can see table status while he is booking a table
15. Customer can see the menu with tags
16. Customer can add items to cart
17. Customer can book a table
18. Customer can change quantity of items in cart
19. Customer can place order of the items in cart
20. Customer can see his previous orders
21. Customer can filter based on tags, sort based on price, rating
22. Customer can rate an(or all) "item" from his/her previous order

23. Customer get a notification if Rcoins are credited to his account
24. Cashier can see all the orders whose payment is not done
25. Cashier can update status of payment to paid after payment
26. Head waiter can order on behalf of customers who don't have an account or internet
27. Head waiter sets order status to 'Served' as needed
28. Head waiter manages(accepts/rejects) booking of tables
29. Head waiter updates status of tables (available or occupied)

Design

- ♣ Original Schema is normalised and its DDL along with constraints, indices, triggers can be found in the appendix 1
- ♣ The primary keys have index on them by default. We have put index to some of our non-primary key attributes which are used frequently in practice. The indices we enforced are on(relation_name(attr1, attr2,..)):
  table_request (status), my_order(status), person (username, session_id)
- ♣ There are two triggers the first one sends a notification to manager when the amount of ingredients falls below the threshold value and the second one sends a notification to customer when rcoins are added to his account
- ♣ Some constraints like status of my_table should be 'available' or 'occupied', quantity in order_item is > 0 and some other similar constraints can be found in DDL in appendix 1
- ♣ Technology choices and rationale:
  - We have used Nodejs for backend, Postgresql for database and pug for scripting
  - Pug is used since it is easy to write clean code in it than html
  - Postgresql is used because it is open source and comes with all features which are needed like triggers, handling failures, concurrency etc
  - Nodejs is used since it is good at performance with a heavy number of small computation costs(due to its asynchronous and non-blocking nature) which is the case here
- ♣ All the functional dependencies can be found in appendix 2

Test Results:

Deliverable 4 for test cases can be found here. All the test cases are passed.

How much of the requirements we could complete?

All of the requirements mentioned in the first section Requirements are completed

Github link:

The repo link can be found here.

Appendix 1: DDL

```
DROP TABLE IF EXISTS table_request;
DROP TABLE IF EXISTS table_order;
DROP TABLE IF EXISTS my_table;
DROP TABLE IF EXISTS rating;
DROP TABLE IF EXISTS order_item;
DROP TABLE IF EXISTS my_order;
DROP TABLE IF EXISTS cart;
DROP TABLE IF EXISTS item_inventory;
DROP TABLE IF EXISTS item_item_tag;
DROP TABLE IF EXISTS item;
DROP TABLE IF EXISTS item_tag;
DROP TABLE IF EXISTS inventory;
DROP TABLE IF EXISTS customer;
DROP TABLE IF EXISTS notification;
DROP TABLE IF EXISTS staff_time_slot;
DROP TABLE IF EXISTS staff;
DROP TABLE IF EXISTS phone;
DROP TABLE IF EXISTS person;


CREATE TABLE person(
        id serial primary key,
        username text not null,
        password text not null,--what say?
        name text not null,
        address_house_no text,
        address_street text,
        address_city text,
        address_state text,
        address_country text,
        address_pin_code text check(address_pin_code ~ '
           ^[0-9]+$'), -- todo: pincode should have only
           six digits ?
        session_id text,
        unique(username)
);
CREATE TABLE phone(
        id int not null,
        phone_number text not null check(phone_number ~ '
           ^[+]?[0-9␣]+$'), --todo: ten digits ?
        primary key (id, phone_number),
        foreign key (id) references person on delete
           cascade,
        unique(phone_number)
```

```
41 );
42
43 CREATE TABLE staff(
44         id int primary key,
45         salary numeric not null,
46         dob date,
47         role_name text check(role_name in ('manager', 'head
              -waiter', 'cashier')),
48         foreign key (id) references person on delete
              cascade
49 );
50 CREATE TABLE staff_time_slot(
51         staff_id int not null,
52         time_slot_id int not null check(time_slot_id<24 and
               time_slot_id>-1),
53         primary key(staff_id, time_slot_id),
54         foreign key (staff_id) references staff on delete
              cascade
55 );
56 CREATE TABLE notification(
57         id serial primary key,
58         info text not null,
59         time_stamp timestamp not null,
60         person_id int not null,
61         foreign key (person_id) references person on delete
               cascade--even if person is deleted,
              notification stays
62 );
63 CREATE TABLE customer(
64         id int primary key,
65         rcoins numeric,
66         foreign key (id) references person on delete
              cascade
67 );
68 CREATE TABLE inventory(
69         id int primary key,
70         name text not null,
71         quantity_remaining numeric not null check(
              quantity_remaining>=0),
72         threshold numeric not null,
73         units text not null,
74         unique(name)
75 );
76 CREATE TABLE item_tag(
77         id int primary key,
78         type text,
```

```
79          unique(type)
80  );
81  CREATE TABLE item(
82          id int primary key,
83          name text not null,
84          price numeric not null,
85          unique(name)
86  );
87  CREATE TABLE item_item_tag(
88          item_id int,
89          tag_id int,
90          primary key(item_id, tag_id),
91          foreign key (item_id) references item on delete
              cascade,
92          foreign key (tag_id) references item_tag on delete
              cascade --once item_tag is deleted,
              corresponding i_i_t entry is deleted
93  );
94  CREATE TABLE item_inventory(
95          item_id int,
96          inventory_id int,
97          quantity_needed numeric,
98          primary key(item_id, inventory_id),
99          foreign key (item_id) references item on delete
              cascade,
100         foreign key (inventory_id) references inventory on
              delete cascade--once inventory is deleted,
              corresponding i_i entry is deleted
101 );
102 CREATE TABLE cart(
103         customer_id int,
104         item_id int,
105         quantity int not null check (quantity >= 0),
106         primary key(customer_id, item_id),
107         foreign key (customer_id) references customer on
              delete cascade,
108         foreign key (item_id) references item on delete
              cascade --once item is deleted, it is
              automatically removed from cart
109 );
110 CREATE TABLE my_order(
111         id serial primary key,
112         customer_id int,--customer_id in order can be null,
               if head waiter places order
113         ordered_time timestamp not null default
              current_timestamp,
```

```
114         served_time timestamp,
115         completed_time timestamp,
116         amount_paid numeric check(amount_paid>=0),
117         rcoins_used numeric check(rcoins_used>=0),
118         status text check(status in ('order-placed', 'order
              -served', 'order-completed')),
119         foreign key (customer_id) references customer on
              delete set null
120 );
121 CREATE TABLE order_item(
122         order_id int,
123         item_id int,
124         quantity int not null check(quantity > 0),
125         total_price numeric not null,
126         primary key(order_id, item_id),
127         foreign key (order_id) references my_order on
              delete cascade,
128         foreign key (item_id) references item on delete
              cascade
129 );
130 CREATE TABLE rating(
131         order_id int,
132         item_id int,
133         stars int check(stars in (1, 2, 3, 4, 5)),
134         review text,
135         primary key(order_id, item_id),
136         foreign key (order_id) references my_order on
              delete cascade,
137         foreign key (item_id) references item on delete
              cascade, --item ratings will be erased once the
              item is deleted
138         check(stars is not null or review is not null)
139 );
140 CREATE TABLE my_table(
141         id int primary key,
142         capacity int not null,
143         location text check(location in ('window-side', '
              non-window-side')) not null,--can add any other?
144         status text check(status in ('occupied', 'available
              ')) not null
145 );
146 CREATE TABLE table_order(
147         order_id int,
148         table_id int,
149         primary key (order_id, table_id),
```

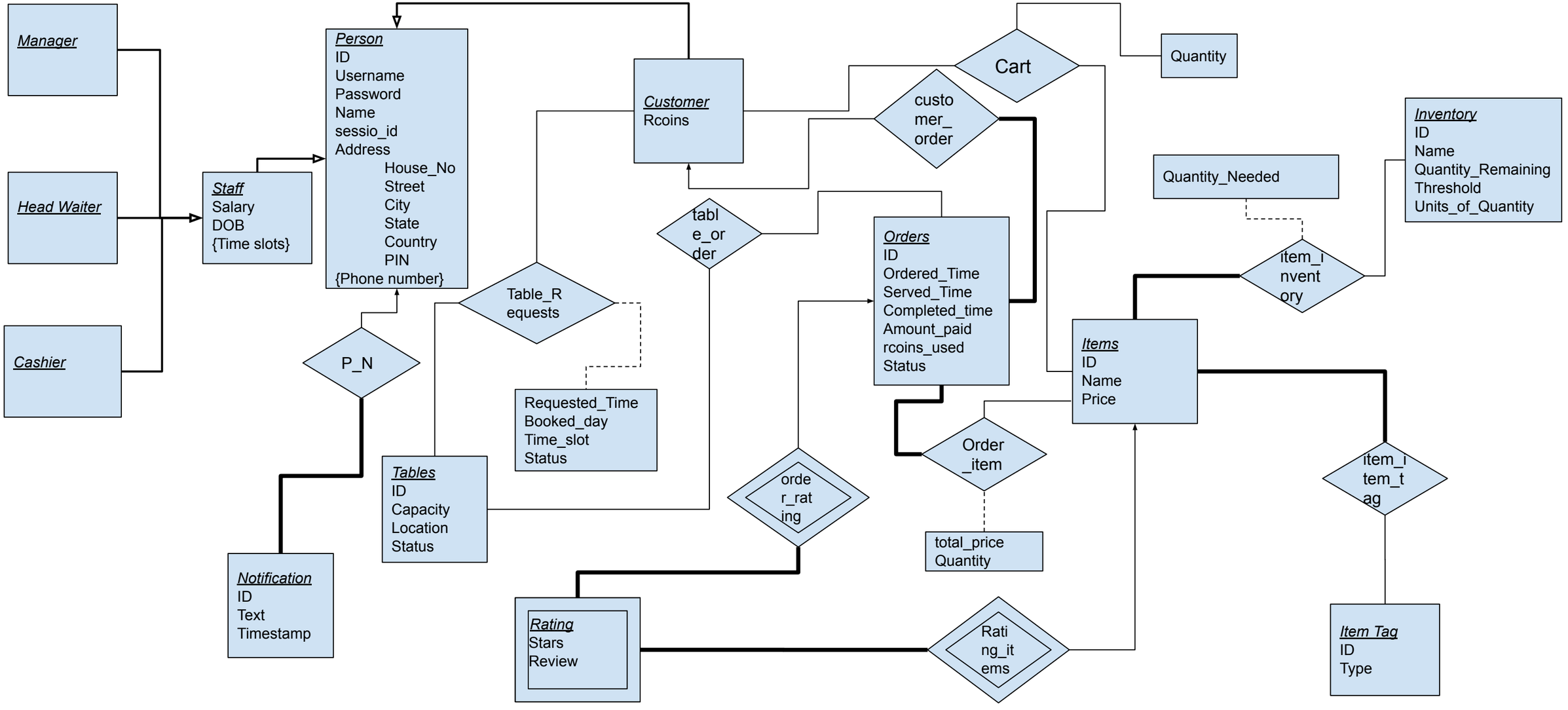
```
150         foreign key (order_id) references my_order on
              delete cascade,
151         foreign key (table_id) references my_table on
              delete cascade
152 );
153 CREATE TABLE table_request(
154         request_id serial primary key,
155         table_id int,
156         customer_id int,
157         requested_time timestamp not null default
              current_timestamp,
158         booked_day date not null,
159         time_slot int check(time_slot>-1 and time_slot<24)
              not null,
160         status text check(status in ('request-placed', '
              request-accepted', 'request-denied')),
161         foreign key (table_id) references my_table on
              delete cascade,
162         foreign key (customer_id) references customer on
              delete cascade--cascading makes finding table-
              availability-status easy
163 );
164
165 --index
166
167 CREATE INDEX table_status_index
168 ON table_request (status);
169
170 CREATE INDEX my_order_status_index
171 ON my_order(status);
172
173 CREATE INDEX username_index
174 ON person (username);
175
176 CREATE INDEX person_session_id_index
177 ON person (session_id);
178
179 --trigger one
180 create or replace function temp1() returns trigger as
181 $$
182 begin
183 insert into notification(info,time_stamp,person_id)
184 (select 'The␣following␣item␣has␣fallen␣below␣threshold'||
      new.name||'threshold␣is'||cast(new.threshold as text)||'
      quantity␣remaining␣is'||cast(new.quantity_remaining as
      text),now(),staff.id from staff where role_name='manager
```

```
    ') ;
return new;
end;
$$
language plpgsql;

create trigger low_inventory after update or insert on
    inventory
for each row
when (new.quantity_remaining<new.threshold)
execute procedure temp1();

--trigger two
create or replace function temp2() returns trigger as
$$
begin
insert into notification values(DEFAULT,cast((new.rcoins-
    old.rcoins) as text)||' rcoins have been added to your
    account, the new total is '||cast(new.rcoins as text),
    now(),new.id);
return new;
end;
$$
language plpgsql;

create trigger gift after update of rcoins on customer
for each row
when (new.rcoins>old.rcoins)
execute procedure temp2();
```

Appendix 2: BCNF

Functional Dependencies	
Table	Functional Dependencies
Person	id→ username,password,name,address_house_no,address_street,address_city, address_state,address_country,address_pin_code username→ id,password,name,address_house_no,address_street,address_city, address_state,address_country,address_pin_code
phone	no non-trivial fd's
time_slot	id→ start_time,end_time
staff	id → salary,dob,role_name
notification	id → info,time_stamp,person_id
customer	id→rcoins
inventory	id→name,quantity_remaining,threshold,units
item_tag	id→type
item	id→name,price
item_item_tag	no non-trivial fd's
item_inventory	item_id,inventory_id→quantity_needed
cart	customer_id,item_id → quantity
my_order	id→customer_id,ordered_time,served_time,completed_time,amount_paid, rcoins_used,status
order_item	order_id,item_id → quantity,total_price
rating	order_id,item_id → stars,review
my_table	id→capacity,location status
table_order	no non-trivial fd's
table_request	request_id → table_id,customer_id,requested_time,start_time,end_time,status

The above table shows all non trivial fd's and as can be seen from the the table the lhs of all fd's is a primary key except in the case of second fd of table 'person' where we have username on lhs but this is allowed as username is superkey in this case. So the schema satisfies the conditions of BCNF.


Manager
Head Waiter
Cashier
Person
ID
Username
Password
Name
sessio_id
Address
House_No
Street
City
State
Country
PIN
{Phone number}
Staff
Salary
DOB
{Time slots}
Customer
Rcoins
Cart
Quantity
custo
mer_
order
Inventory
ID
Name
Quantity_Remaining
Threshold
Units_of_Quantity
Quantity_Needed
item_i
nvent
ory
tabl
e_or
der
Orders
ID
Ordered_Time
Served_Time
Completed_time
Amount_paid
rcoins_used
Status
Table_R
equests
P_N
Requested_Time
Booked_day
Time_slot
Status
Items
ID
Name
Price
Tables
ID
Capacity
Location
Status
orde
r_rat
ing
Order
_item
item_i
tem_t
ag
total_price
Quantity
Notification
ID
Text
Timestamp
Rating
Stars
Review
Rati
ng_it
ems
Item Tag
ID
Type