IIT PALAKKAD

DBMS Final Report

Restaurant Management System

Team Members:

Chapati Anusha - 112101010

Ch Laasya Priya - 112101012

Karumudi Harika - 112101023

Instructors:

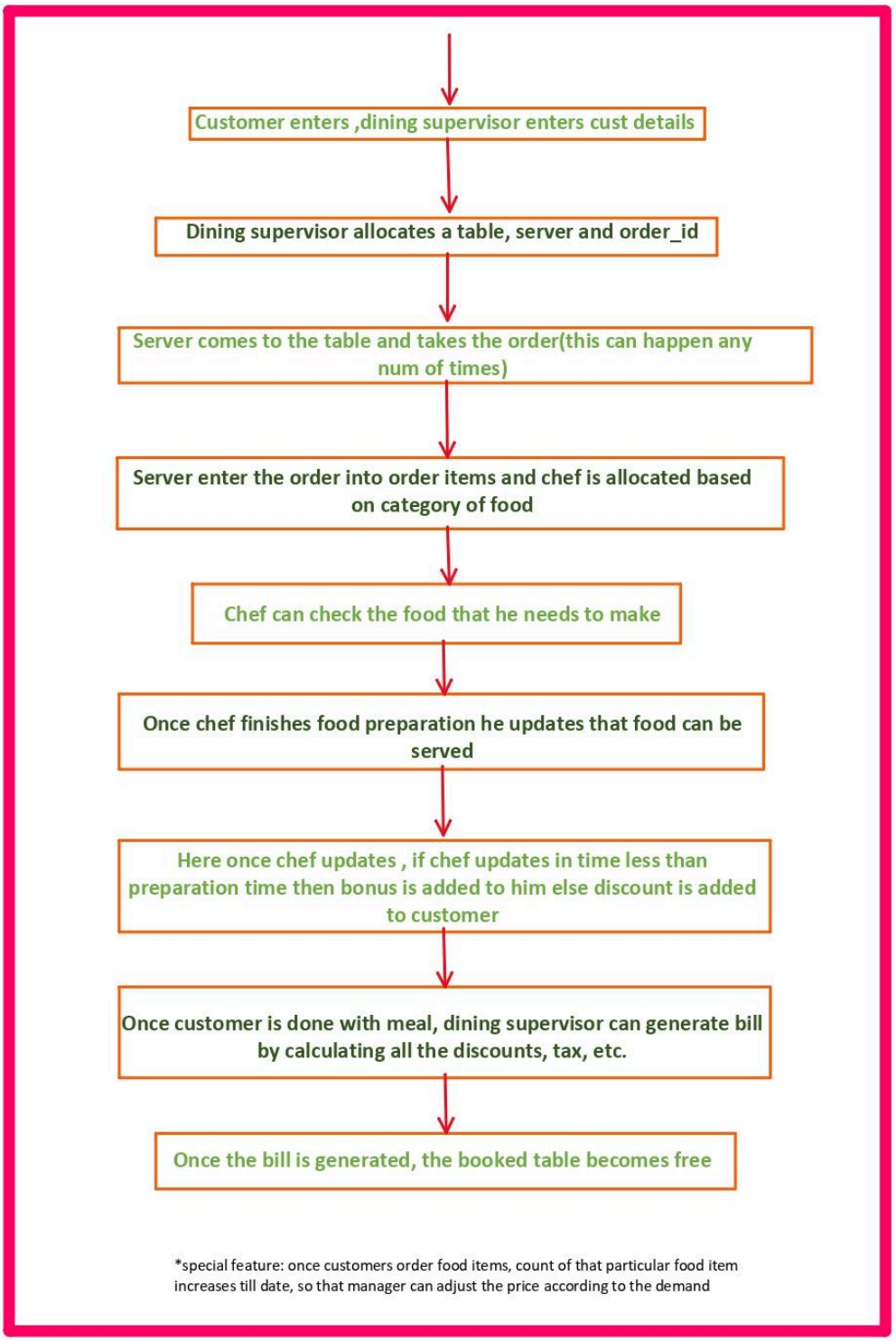
Dr. Koninika Pal

Dr. Sahely Bhadra

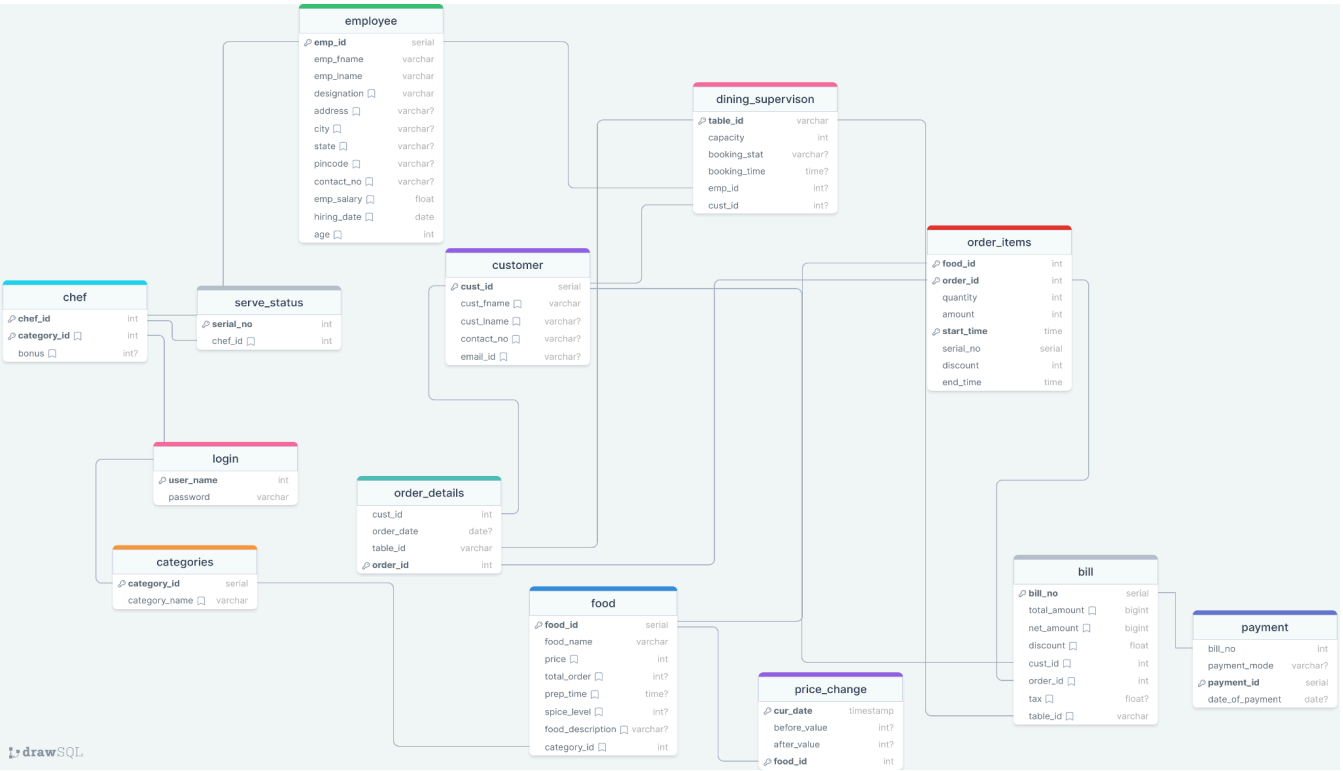
Index

- **About the project:** The restaurant management system provides many features, including table allocations, order item notation, and automated bill calculations. It effectively handles table turnover by automatically freeing up tables after bill settlements, ensuring a smooth dining experience. The system caters to various roles, including manager, server, dining supervisor, chef, and customer, enabling them to fulfil their responsibilities seamlessly. Furthermore, it tracks menu updates and price changes based on demand, aiding in informed decision-making for the business.

- **Workflow of the project:**



- **Relational model of database:**



Tables in the Database

login_details
Primary key: user_name **Foreign Key:** None **General Constraints:** **NOTNULL:** user_name, password

employee
Primary key: emp_id **Foreign Key:** None **General Constraints:** **NOTNULL:** emp_fname, emp_lname, designation, emp_salary, hiring_date, age

categories
Primary key:category_id **Foreign Key:** None **General Constraints:** **NOTNULL:** category_name

chef
Primary key: chef_id, category_id **Foreign Key:** category_id refers categories.category_id chef_id refers employee.emp_id **General Constraints:** **NOTNULL:** chef_id, category_id **CHECK:** bonus>=0

customer
Primary key: cust_id **Foreign Key:** None **General Constraints:** **NOTNULL:** cust_fname

food
Primary key: food_id **Foreign Key:** category_id refers categories.category_id **General Constraints:** **NOTNULL:** food_name, price, category_id **CHECK:** price>0, prep_time>’00:00:00’

dining_supervison
Primary key: table_id **Foreign Key:** emp_id refers to employee.emp_id **General Constraints:** **NOTNULL:** table_id, capacity **CHECK:** capacity>0

order_details
Primary key: order_id **Foreign Key:** table_id refers dining_supervison.table_id cust_id refers customer.cust_id **General Constraints:** **NOTNULL:** cust_id, table_id

order_items

Primary key: serial_no
Foreign Key: food_id refers food.food_id
order_id refers order_details.order_id
General Constraints:
NOTNULL: order_id, food_id, quantity
CHECK: quantity>0

serve_status

Primary key: serial_no
Foreign Key: chef_id refers employee.emp_id
General Constraints:
NOTNULL: serial_no

bill

Primary key: bill_no
Foreign Key: table_id refers dining_supervison.table_id
cust_id refers customer.cust_id
General Constraints:
NOTNULL: net_amount, table_id, cust_id, order_id
CHECK: total_amount>=0, discount>=0

payment

Primary key: cur_date, food_id
Foreign Key: food_id refers food.food_id
General Constraints:
NOTNULL: food_id

price_change

Primary key: cur_date, food_id
Foreign Key: food_id refers food.food_id
General Constraints:
NOTNULL: food_id,cur_date

- **FUNCTIONS AND PROCEDURES USED :**

- **FUNCTION GetLastcustomer():** To retrieve the latest `customer_id` inserted into the database, a custom function named GetLastcustomer has been implemented. This function utilizes the PostgreSQL currval function to fetch the current value of the customer_cust_id_seq sequence, providing real-time access to the latest customer_id for front-end applications.

```
---function to get latest customer_id---

CREATE OR REPLACE FUNCTION GetLastcustomer()
RETURNS INTEGER AS $$
DECLARE
    last_pk INTEGER;
BEGIN
    SELECT currval('customer_cust_id_seq') INTO last_pk;
    RETURN last_pk;
END;
$$ LANGUAGE plpgsql;

select * from GetLastcustomer();

-----------------------------------------
```

- **PROCEDURE Table_allocation(custid INT,members INT):** This procedure has been created to allocate tables based on customer requirements. This procedure iterates through the `dining_supervision` table to find an available table with sufficient capacity. If a suitable table is found, it updates the booking status to 'booked' along with the booking time and customer ID. Additionally, it inserts a record into the `order_details` table to track the booking. If no suitable table is available, a notification is raised indicating the unavailability of tables.

```
---function for table allocation---

CREATE OR REPLACE PROCEDURE Table_allocation(custid INT,members INT)
AS $$
DECLARE
    f record;
    t int;
BEGIN
    t := 0;
    FOR f IN SELECT capacity, table_id, booking_stat FROM dining_supervison ORDER BY capacity LOOP
        IF (f.booking_stat = 'not_booked' AND f.capacity >= members) THEN
            UPDATE dining_supervison SET booking_stat = 'booked', booking_time = CURRENT_TIME, cust_id = custid WHERE table_id = f.table_id;
            t := 1;
            RAISE NOTICE 'Table allocated = %', f.table_id;
            RAISE NOTICE 'Booked Successfully';
            INSERT INTO order_details(cust_id, order_date, table_id) VALUES (custid, CURRENT_DATE, f.table_id);
            EXIT;
        END IF;
    END LOOP;

    IF (t = 0) THEN
        RAISE NOTICE 'No suitable table found';
    END IF;
END;
$$ LANGUAGE plpgsql;

-----------------------------------------
```

- **FUNCTION GetLasttableid():**This has been implemented to retrieve the latest table_id from the dining_supervison table based on the most recent booking time. This function facilitates real-time access to the latest table allocation in the restaurant

```
---function to get latest table_id---

CREATE or replace FUNCTION GetLasttableid()
RETURNS varchar AS
$$
DECLARE
    last_pk varchar;
BEGIN
    SELECT table_id INTO last_pk
    FROM dining_supervison
    ORDER BY booking_time DESC
    LIMIT 1;
    RETURN last_pk;
END;
$$ LANGUAGE plpgsql;
select * from dining_supervison;
select * from GetLasttableid();
-----------------------------------------
```

- **FUNCTION GetLastorderid():** This has been implemented to retrieve the latest order_id from the order_details table based on the most recent booking time.

```
---function to get latest order_id---

CREATE FUNCTION GetLastorderid()
RETURNS INTEGER AS
$$
DECLARE
    last_pk INTEGER;
BEGIN
    SELECT order_id INTO last_pk
    FROM order_details
    ORDER BY order_id DESC
    LIMIT 1;

    RETURN last_pk;
END;
$$ LANGUAGE plpgsql;

select * from GetLastorderid();

-----------------------------------
```

- **PROCEDURE insert_oi(orderid int,foodid int , quant int):** This procedure has been created to insert records into the order_items table, capturing details such as order_id, food_id, quantity, and the start time of the order. This procedure facilitates the recording of individual food items ordered by customers for a particular order_id.

```
---function to insert order_items by server---

CREATE OR REPLACE PROCEDURE insert_oi(orderid int,foodid int , quant int )
 AS $$
BEGIN
    insert into order_items (order_id, food_id, quantity,start_time) values (orderid,foodid,quant,CURRENT_time);

END;
$$ LANGUAGE plpgsql;

call insert_oi(4,1,2);
call insert_oi(4,2,2);
call insert_oi(4,3,2);
-----------------------------------
```

- **PROCEDURE chef_serve(sno int):** This procedure updates the serve_status table, marking the completion of food preparation by setting the status to 'Yes' for the specified serial_no. This process allows the chef to efficiently manage the preparing and serving process.

```
--serve status getting updated Done by chef

CREATE OR REPLACE PROCEDURE chef_serve(sno int) as $$
DECLARE
sid int;
BEGIN
update serve_status set status = 'Yes' where serial_no = sno;
END;
$$ LANGUAGE plpgsql;

call chef_serve(5);
call chef_serve(6);
call chef_serve(7);

-----------------------------------------------
```

- **PROCEDURE bill(oid INT, pm varchar):** This procedure generates and records the bill for a specific order by

calculating the total amount, tax, discounts, and net amount. It then assigns a bill number and updates the payment mode for the order, facilitating seamless billing and payment processing.

```
---generating bill by supervisor---

CREATE OR REPLACE PROCEDURE bill(oid INT, pm varchar)  AS $$
DECLARE
  na INT;
  tot_amount INT;
  tax float;
  disc float;
  tid VARCHAR(8);
  cid int;
  billno int;
BEGIN
  SELECT cust_id INTO cid FROM order_details WHERE order_id = oid;
  select table_id into tid from order_details WHERE order_id = oid;
  SELECT SUM(amount) into tot_amount FROM order_items WHERE order_id = oid GROUP BY order_id;
  SELECT SUM(discount) into disc FROM order_items WHERE order_id = oid GROUP BY order_id;
  tax := tot_amount* 0.18;
  na := tot_amount + tax - disc;
  insert into bill (total_amount,tax,discount,net_amount,table_id,cust_id,order_id) values (tot_amount,tax,disc,na,tid,cid,oid);
  select bill_no into billno from bill where order_id = oid;
  update payment set payment_mode = pm;

END;
$$ LANGUAGE plpgsql;

call bill(4,'gpay');
select * from bill;

------------------------------
```

- **FUNCTION adjust_price(p_food_id INT, p_quantity INT):**This function dynamically changes the price of a food item based on its current demand. If the total ordered quantity exceeds a specified threshold, the price is increased by 5%; otherwise, it is decreased by 5%. This function helps manage pricing strategies to optimize revenue and customer satisfaction.

```
---function for price change---

CREATE OR REPLACE FUNCTION adjust_price(p_food_id INT, p_quantity INT) RETURNS VOID AS $$
DECLARE
    total_ordered_quantity INT;
    price_change_factor NUMERIC;
    old_price INT;
    new_price INT;
BEGIN
    SELECT price INTO old_price FROM food WHERE food_id = p_food_id;
    SELECT COALESCE(SUM(oi.quantity), 0) INTO total_ordered_quantity FROM order_items oi
    JOIN order_details od ON oi.order_id = od.order_id WHERE oi.food_id = p_food_id
    AND date_trunc('month', od.order_date) = date_trunc('month', CURRENT_DATE);

    IF total_ordered_quantity > p_quantity THEN
        price_change_factor := 1.05; -- Increase price by 5%
        RAISE NOTICE 'Price for food_id % is being incremented.', p_food_id;
    ELSE
        price_change_factor := 0.95; -- Decrease price by 5%
        RAISE NOTICE 'Price for food_id % is being decremented.', p_food_id;
    END IF;

    new_price := ROUND(old_price * price_change_factor);

    UPDATE food SET price = new_price WHERE food_id = p_food_id;
    RAISE NOTICE 'Old price: %, New price: %', old_price, new_price;

    INSERT INTO price_change (food_id, before_value, after_value)
    VALUES (p_food_id, old_price, new_price);
END;
$$ LANGUAGE plpgsql;

SELECT * from adjust_price(1, 10);

-------------------------------------------
```

- **TRIGGERS AND THEIR FUNCTION CALLS:**

- **trigger generate_custid:**It is set to execute the `get_custid` function after a new row is inserted into the `customer` table. This trigger generates a notice containing the `cust_id` of the newly added customer, providing real-time feedback for monitoring new customer additions.

```
---To get notice for customer_id when new cusomer added---

CREATE OR REPLACE FUNCTION get_custid() RETURNS TRIGGER AS $$
BEGIN
raise notice 'Customer id = %',New.cust_id;
return NULL;
END;
$$ LANGUAGE plpgsql;

create or replace trigger generate_custid
after insert on customer
for each row
execute function get_custid();

-------------------------------------------------------
```

- **trigger generate_oid:** This is designed to execute the `get_oid` function after a new row is inserted into the `order_details` table. This trigger generates a notice containing the `order_id` and the corresponding `cust_id` of the newly added order, providing real-time feedback for tracking new orders associated with customers.

```
---To get notice for order_id when new cusomer added---

CREATE OR REPLACE FUNCTION get_oid() RETURNS TRIGGER AS $$
BEGIN
raise notice 'order id = % for customer = % ',new.order_id,New.cust_id;
return NULL;
END;
$$ LANGUAGE plpgsql;

create or replace trigger generate_oid
after insert on order_details
for each row
execute function get_oid();

-------------------------------------------------------
```

- **TRIGGER calc_amount_trigger:** This trigger is set to execute the calc_amount function after a new row is inserted into the order_items table. This trigger calculates the amount for the ordered item based on its price and quantity, updating the amount column for the newly inserted row. This ensures that the total amount for each ordered item is accurately calculated and stored in the database.

```
---To calculate amount for ordered item in order_items---

CREATE OR REPLACE PROCEDURE cal_amount(foodid INT, quant INT,sno int) AS $$
DECLARE
    p INT;
BEGIN
    SELECT price INTO p FROM food WHERE food_id = foodid;
    -- Update only the newly inserted row in order_items
    UPDATE order_items SET amount = p * quant WHERE serial_no = sno;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION calc_amount() RETURNS TRIGGER AS $$
BEGIN
    CALL cal_amount(NEW.food_id, NEW.quantity,NEW.serial_no);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER calc_amount_trigger
AFTER INSERT ON order_items
FOR EACH ROW
EXECUTE FUNCTION calc_amount();

-------------------------------------------------------------
```

- **trigger total_quant_trigger:**This is set to execute the `total_quant` function after a new row is inserted into the

`order_items` table. This trigger updates the `total_order` column in the `food` table, incrementing it by the quantity of the newly inserted order item. This helps track the total quantity of each food item ordered across all orders.

```
---This updates the food table in total orders till now--------

CREATE OR REPLACE PROCEDURE tot_quant(foodid INT, quant INT) AS $$
BEGIN
    UPDATE food SET total_order = total_order + quant WHERE food_id = foodid;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION total_quant() RETURNS TRIGGER AS $$
BEGIN
    CALL tot_quant(NEW.food_id, NEW.quantity);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER total_quant_trigger
AFTER INSERT ON order_items
FOR EACH ROW
EXECUTE FUNCTION total_quant();

-------------------------------------------------
```

- **trigger `into_serve_stat_trigger`**: This is set to execute the `into_serve_stat` function after a new row is inserted into the `order_items` table. This trigger allocates a chef to the newly inserted order item based on the category of the food item. It retrieves the `chef_id` associated with the `category_id` of the food item and inserts an entry into the `serve_status` table to track the serving status of the food item by the chef.

```
---Allocating chef for each food_item---

CREATE OR REPLACE PROCEDURE allocate_chef(sno INT, foodid INT) AS $$
DECLARE
    cid INT;
BEGIN
    -- Retrieve the chef_id based on the category_id of the food
    SELECT chef_id INTO cid
    FROM chef
    INNER JOIN food ON chef.category_id = food.category_id
    WHERE food.food_id = foodid;
    -- Insert the allocation into serve_status
    INSERT INTO serve_status (serial_no, chef_id, status)
    VALUES (sno, cid, 'No');
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION into_serve_stat() RETURNS TRIGGER AS $$
BEGIN
    CALL allocate_chef(NEW.serial_no, NEW.food_id);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE or replace TRIGGER into_serve_stat_trigger
AFTER INSERT ON order_items
FOR EACH ROW
EXECUTE PROCEDURE into_serve_stat();

-----------------------------------------------------------
```

- **trigger `update_serve_stat_trigger:** This is designed to execute the `update_serve_stat` function before an update

operation is performed on the `serve_status` table. This trigger calculates discounts or updates chef bonuses based on the difference between the start and end times of food preparation. If the difference exceeds the preparation time plus 20 minutes, a 5% discount is applied to the order item; otherwise, a bonus of 2 is added to the chef's bonus. This mechanism incentivizes timely food preparation and enhances customer satisfaction.

```
---creating trigger for discount calculation by checking serve time---

CREATE OR REPLACE PROCEDURE end_time_disc(sno INT, cid INT) AS $$
DECLARE
    start_time1 TIME;
    end_time1 TIME;
    prep_time1 INTERVAL;
    foodid int;
    catid int;
BEGIN
    -- Update the end time in the order_items table
    UPDATE order_items SET end_time = CURRENT_TIME
    WHERE serial_no = sno;

    -- Retrieve start_time, end_time, and prep_time,foodid
    SELECT food_id INTO foodid FROM order_items WHERE serial_no = sno;
    SELECT start_time INTO start_time1 FROM order_items WHERE serial_no = sno;
    SELECT end_time INTO end_time1 FROM order_items WHERE serial_no = sno;
    SELECT prep_time INTO prep_time1 FROM food WHERE food_id = foodid;
    SELECT category_id INTO catid FROM food where food_id = foodid;

    -- Check if the difference between end_time and start_time is greater than prep_time plus 20 minutes
    IF (end_time1 - start_time1 > prep_time1 + INTERVAL '20 minutes') THEN
        -- Apply discount if the condition is met
        UPDATE order_items SET discount = 0.05 * amount WHERE serial_no = sno;
    ELSE
        -- Otherwise, update chef bonus
        UPDATE chef SET bonus = bonus + 2 WHERE chef_id = cid and category_id = catid;
    END IF;
END;
$$ LANGUAGE plpgsql;

CREATE or replace TRIGGER update_serve_stat_trigger
before UPDATE ON serve_status
FOR EACH ROW
EXECUTE FUNCTION update_serve_stat();

-----------------------------------------------------------------
```

- **trigger insert_into_payment_trigger:** This inserts a record into the payment table after a new row is inserted into the bill table, linking the payment to its corresponding bill using the bill number and recording the date of payment.

```
---Trigger to insert payment record after an insertion into the bill table---

CREATE OR REPLACE FUNCTION insert_into_payment() RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO payment (date_of_payment, bill_no) VALUES (CURRENT_DATE, NEW.bill_no);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE or replace TRIGGER insert_into_payment_trigger
AFTER INSERT ON bill
FOR EACH ROW
EXECUTE FUNCTION insert_into_payment();
```

- **trigger update_dining_supervison_trigger:** This updates the dining_supervison table, marking the associated table as 'not_booked' after a new bill is inserted, ensuring that the table is available for the next booking.

```
---Trigger to update the dining_supervison table after an bill is done---

CREATE OR REPLACE FUNCTION update_dining_supervison() RETURNS TRIGGER AS $$
BEGIN
    UPDATE dining_supervison
    SET booking_stat = 'not_booked'
    WHERE table_id = NEW.table_id;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE or replace TRIGGER update_dining_supervison_trigger
AFTER INSERT ON bill
FOR EACH ROW
EXECUTE FUNCTION update_dining_supervison();

-----------------------------------------------------------------------
```

- **ROLES**

Manager

The manager operates the entire restaurant so he has access to the complete database.

```
create role manager
with superuser
login
password 'manager123';
```

Granted Privileges for the Manager

```
---granting all privileges to manager---
grant all
on all tables
in schema "public"
to manager;
```

Chef

The chef looks at whether his category food has been ordered or not. If ordered, he starts preparing the food when it is done he updates that the food is ready.

```
create role chef
login
password 'chef123';
```

Granted Privileges for the Chef

```
---granting privileges to chef---
grant update on serve_status to chef;
grant select on chef,food to chef;
grant select on serve_status to chef;
grant update,select on serve_status to chef;
grant update,select on order_items to chef;
grant update ,select on chef to chef;
```

Dining supervisor

The dining supervisor will allow the table to the customer i.e. table allocation, he also generates a bill for a particular order and has access to the payment table where the restaurant keeps the records of all payments done.

```
create role dining_supervisor
login
password 'dining_supervisor123';
```

Granted Privileges for the Dining supervisor

```
-- granting privileges to dining_supervisor

grant all on dining_supervison , bill , payment to dining_supervisor;
grant select on order_details,order_items to dining_supervisor;
grant insert on order_details to dining_supervisor;
grant all on customer to dining_supervisor;
GRANT USAGE, SELECT ON SEQUENCE customer_cust_id_seq TO dining_supervisor;
GRANT USAGE, SELECT ON SEQUENCE order_details_order_id_seq TO dining_supervisor;
GRANT USAGE, SELECT ON SEQUENCE bill_bill_no_seq TO dining_supervisor;
GRANT USAGE, SELECT ON SEQUENCE payment_payment_id_seq TO dining_supervisor;
```

Server

The server takes customer's orders after the dining supervisor has assigned them a table, shows the menu to customers, and brings the food prepared by the chef after the chef updates that the food is ready.

```
create role dining_supervisor
login
password 'dining_supervisor123';
```

Granted Privileges for the Server

```
-- grantong privileges to server
grant select on food , order_details, dining_supervison,serve_status to server;
grant all on order_items to server;
grant usage, select on sequence order_items_serial_no_seq TO server;
grant select on chef to server;
grant insert on serve_status to server;
grant insert on food to server;
grant update on food to server;
```

Customer

The customer comes and orders the food.He/She has access to the menu that comes from the food table.

```
create role customer
login
password 'customer123';
```

Granted Privileges for the Customer

```
-- granting privileges to customer
grant select on food to customer;
```

- ## VIEWS
  - **Display_food:** Whenever we go to a restaurant and ask the server how the food is, he tells us the description. So this view helps the server check the food table's description and tell that to the customer

```
create view display_food as (select food_id,food_name,food_description from food order by food_id);
```

  - **notserved_food:**When food orders are placed, chefs are required to review the list of food items assigned to their specific IDs that are pending service, displaying the food names instead of the food IDs.

```
create view notserved_food as ( SELECT serve_status.chef_id,
food.food_name,order_items.quantity,serve_status.serial_no
FROM serve_status
INNER JOIN order_items ON serve_status.serial_no = order_items.serial_no
INNER JOIN food ON order_items.food_id = food.food_id
INNER JOIN order_details ON order_items.order_id = order_details.order_id
WHERE serve_status.status LIKE 'No'
AND order_details.order_date = CURRENT_DATE
ORDER BY order_items.start_time ASC
);
```

  - **chef_food:**When a customer raises a complaint about their food not being served, the server consults the view displaying all not-served food items along with their respective chef allocations to address the issue promptly. The server goes to the particular chef and enquiries.

```
create view chef_food as ( SELECT order_items.order_id,food.food_name,
CONCAT(employee.emp_fname, ' ', employee.emp_lname) AS chef_name
FROM serve_status
INNER JOIN order_items ON serve_status.serial_no = order_items.serial_no
INNER JOIN food ON order_items.food_id = food.food_id
INNER JOIN employee ON employee.emp_id = serve_status.chef_id
INNER JOIN order_details ON order_items.order_id = order_details.order_id
Where order_details.order_date = CURRENT_DATE
ORDER BY order_items.start_time ASC );
```

  - **orders_revenue_by_date:**This view provides the manager with the total number of orders placed and the corresponding total amount received for a specific day, aiding in the oversight of the restaurant's daily operations and financial performance.The below **view is for the total revenue per day**:

```
CREATE VIEW orders_revenue_by_date AS(
SELECT order_date, COUNT(order_details.order_id) AS total_orders,
SUM(net_amount) AS total_revenue
FROM order_details
JOIN bill ON order_details.order_id = bill.order_id
GROUP BY order_date );
```

The below **view is for total revenue by month:**

```
CREATE VIEW orders_revenue_by_month AS (
    SELECT DATE_TRUNC('month', order_date) AS month,
           COUNT(order_details.order_id) AS total_orders,
           SUM(net_amount) AS total_revenue
    FROM order_details
    JOIN bill ON order_details.order_id = bill.order_id
    GROUP BY month
);
```

The below **view is the <u>total revenue received for each food category</u>**:

```
CREATE VIEW orders_revenue_by_month AS (
    SELECT DATE_TRUNC('month', order_date) AS month,
           COUNT(order_details.order_id) AS total_orders,
           SUM(net_amount) AS total_revenue
    FROM order_details
    JOIN bill ON order_details.order_id = bill.order_id
    GROUP BY month
);
```

- **<u>top_selling_food_items:</u>** This view facilitates the retrieval of food items sorted by the total number of orders placed, making it easy to recommend the most popular dishes when asked about the best options available at our restaurant.

```
▷ Run | New Tab | Copy
CREATE VIEW top_selling_food_items AS(
SELECT f.food_id, f.food_name, COALESCE(SUM(oi.quantity), 0) AS total_orders
FROM food f
LEFT JOIN order_items oi ON f.food_id = oi.food_id
GROUP BY f.food_id, f.food_name
ORDER BY total_orders DESC
LIMIT 5 );
```

- **<u>revenue_each_category:</u>**To ensure the accuracy of the cash provided by the dining supervisor and the total amount received, the manager utilises a view to review the total revenue generated in each category, such as PhonePe and Google Pay transactions.

```
▷ Run | New Tab | Copy
CREATE VIEW total_revenue_per_payment_mode AS
SELECT payment_mode, SUM(net_amount) AS total_revenue
FROM payment
JOIN bill ON payment.bill_no = bill.bill_no
GROUP BY payment_mode;
```

- **INDEXING:**

Indexing on food prices

The index used: B+ Tree: Utilising a B+ tree index on the price field enables efficient retrieval of food items within a specified price range. This is particularly beneficial for users seeking food options within specific budget constraints.

```
Run | New Tab
CREATE INDEX idx_food_price ON food  USING BTREE (price);

--q1
Run | New Tab | Copy
CREATE OR REPLACE FUNCTION get_food_between_prices(min_price INT, max_price INT)
RETURNS TABLE (food_id INT, food_name VARCHAR(100), price INT, food_description VARCHAR(100))
AS $$
BEGIN
    RETURN QUERY
    SELECT f.food_id, f.food_name, f.price, f.food_description
    FROM food f
    WHERE f.price BETWEEN min_price AND max_price;
END;
$$ LANGUAGE plpgsql;
```

Takes the minimum and maximum price and returns the food items in the given range.

Indexing on the description of the food

The index used: GIN index : When customers inquire about food items containing rice or those prepared with vegetables, we can utilise a GIN index in the food description field. This enables us to efficiently retrieve relevant food items using queries structured as "like '%rice%'" and "like '%vegetables%'".

```
Run | New Tab
CREATE EXTENSION IF NOT EXISTS pg_trgm;
Run | New Tab
CREATE INDEX idx_food_description_gin ON food USING gin (food_description gin_trgm_ops);

--q1--
Run | New Tab | JSON
SELECT * FROM food WHERE food_description ILIKE '%rice%';
--q2--
Run | New Tab | JSON
SELECT * FROM food WHERE food_description ILIKE '%chicken%'
```

Indexing on the date of payment and amount

The index used: Hash index for date and B+tree for amount: Retrieve the total bills for a specific date where the payment amount exceeds or equals a given value, considering the payment method.
For example, a query like retrieve all orders made on November 19, 2023, where the net amount exceeds 2000 and payment was made using either a credit or debit card?

```
Run | New Tab
CREATE INDEX idx_payment_date_of_payment_hash ON payment USING HASH (date_of_payment);
-- B-tree index on net_amount column in bill table
Run | New Tab
CREATE INDEX idx_bill_net_amount_btree ON bill USING BTREE (net_amount);
```

```
---q1--

Run | New Tab | Copy
CREATE OR REPLACE FUNCTION
get_orders_by_price_and_paymentmode(price_value NUMERIC, payment_mode_param VARCHAR(10))
RETURNS TABLE (order_id INT, bill_id INT, net_amount NUMERIC) AS $$
BEGIN
    RETURN QUERY
    SELECT b.order_id, b.bill_no, b.net_amount::NUMERIC FROM bill b
    JOIN payment p ON b.bill_no = p.bill_no
    WHERE p.date_of_payment = CURRENT_DATE AND
    p.payment_mode = payment_mode_param AND b.net_amount > price_value;
END;
$$ LANGUAGE plpgsql;
```

Indexing on category name

The index used: Hash index for category name:Retrieve all the details of food that are matched for the given category name.As we are using a complete string match of the category name it is better to use a hash index For queries like getting food details that are categorised as 'Desserts'.

```
---creating HASH for categories on category_name---
 Run | New Tab
CREATE INDEX idx_categories_category_name_hash
 ON categories USING HASH (category_name);
--q1
 Run | New Tab | Copy
CREATE OR REPLACE FUNCTION get_foods_by_category(p_category_name VARCHAR(20))
RETURNS TABLE (food_id INT, food_name VARCHAR(100), price INT, total_order INT,
prep_time TIME, spice_level INT, food_description VARCHAR(100))
AS $$
BEGIN
    RETURN QUERY
    SELECT f.food_id, f.food_name, f.price, f.total_order,
    f.prep_time, f.spice_level, f.food_description
    FROM food f
    JOIN categories c ON f.category_id = c.category_id
    WHERE c.category_name = p_category_name;
END;
$$ LANGUAGE plpgsql;

 Run | New Tab | JSON
SELECT * FROM get_foods_by_category('Soup');
```

- **Queries that are used for various roles in the front end:(We are planning to show during presentation)**

Manager:

- view_all_employees
- view_menu
- revenue_by_month
- top_selling_foods
- change_price

Chef:

- check_pending_orders
- update_serve_status

Server:

- enter_order
- food_and_chef

Dining_supervisor:

- table_allocation
- bill_calculation
- display_bill

Customer:

- view_menu

- **Workflow and output from customer entering to restaurant till bill calculation :**

1. **Entering customer details (Done by: the dining supervisor)**

   Enter the customer's first name, last name, contact number, and email id.

   This inserts the customer data into the customer table.

   Let the customer be: Emily Cooper

```
1004  insert into customer(cust_fname,cust_lname,contact_no,email_id)
1005  values ('Emily','Cooper','123456789','emily123@gmail.com')
1006  select * from customer
```

Data Output | Notifications | Messages

	cust_id [PK] integer	cust_fname character varying (15)	cust_lname character varying (15)	contact_no character varying (13)	email_id character varying (30)
1	1	Emily	Cooper	123456789	emily123@gmail.com

   The cust_id of Emily is 1.

```
1008  select GetLastcustomer()
```

Data Output | Notifications | Messages

	getlastcustomer integer
1	1

   There is also a raise notice feature to view customer_id as soon as it gets inserted into the customer:

   To check the raise notice i have inserted another customer.

```
1007  insert into customer(cust_fname,cust_lname,contact_no,email_id)
1008  values ('Mary','Jane','123546789','Jane123@gmail.com')
```

Data Output | Notifications | Messages

```
NOTICE:  Customer id = 2
INSERT 0 1
```

2. **Ask the customer the total number of people who are joining him/her (Done by: the dining supervisor)**

   So now after the customer response, the dining supervisor enters the details of the total no.of people that are with that customer, customer_id.

   Now the Table_allocation function returns the Table allocated.

   Let the total number of people along with Emily be 5:

```
1013  call Table_allocation(1,5)
1014
```

Data Output | Notifications | Messages

```
NOTICE:  Table allocated = T7
NOTICE:  Booked Successfully
NOTICE:  order id = 1 for customer = 1
CALL

Query returned successfully in 99 msec.
```

   Table T7 is allocated to Emily and her family and her order_id is told to her as 1.

```
1015  select * from dining_supervison
1016
```

Data Output | Notifications | Messages

	table_id [PK] character varying (8)	capacity integer	booking_stat character varying (10)	booking_time time without time zone	emp_id integer	cust_id integer
12	T13	2	not_booked	00:00:00	11	[null]
13	T14	4	not_booked	00:00:00	12	[null]
14	T7	6	booked	21:20:08.0999	12	1

   As capacity is 5 Allocate tables of capacity 6. The booking time of the table is also updated.

3. **Taking the order from the customers. (Done by: Server)**

This inserts the order_id, food_id, and quantity into the order_items table.

The start time of food is entered into the order_items table.

Now at this point, the serving status of that food item is 'not yet served', and also when the food is ordered the total_quantity column in the food table increments by the quantity that a customer ordered.

Let Emily ordered the following items:

->Tomato Soup (food id = 1) - price 110 of quantity 3

->Lasagna (food id = 4) - price 280 of quantity 2

->Hyderabadi Biryani (food id = 7) - price 250 of quantity 4

```
call insert_oi(1,1,3);
call insert_oi(1,4,2);
call insert_oi(1,7,4);

select * from order_items
```

ıta Output | Notifications | Messages

serial_no [PK] integer	order_id integer	food_id integer	discount integer	quantity integer	amount integer	start_time time without time zone	end_time time without time zone
1	1	1	0	3	330	21:27:47.95869	[null]
2	1	4	0	2	560	21:27:47.95869	[null]
3	1	7	0	4	1000	21:27:47.95869	[null]

We can see the end time is not updated.

```
select * from serve_status
```

a Output | Notifications | Messages

serial_no [PK] integer	chef_id integer	status character varying (10)
1	1	No
2	4	No
3	3	No

Immediately a chef is allocated to each of these food_items and the details are entered into serve_status

	food_id [PK] integer	food_name character varying (100)	price integer	total_order integer
28	1	Tomato Soup	110	3
29	4	Lasagna	280	2
30	7	Hyderabadi Biryani	250	4

According to the trigger created as the order is placed the total quantity in food_table gets updated.

4. **Next, update that the food was prepared. (Done by: chef)**

This changed the serve_status of that food item to "served".

When the serve_status changed, the end time of preparation of food was entered into order_items.

If end_time - start_time <= prep_time then the chef gets a bonus.

Chef gets a bonus of 2 points for every item that he prepares is less than end_time - start_time.

```
call chef_serve(1);
call chef_serve(2);
```

Output | Notifications | Messages

serial_no [PK] integer	chef_id integer	status character varying (10)
3	3	No
1	1	Yes
2	4	Yes

After the chef has served 1 and 2 its status is updated as Yes.

```
call chef_serve(3);

select * from order_items
```

Output | Notifications | Messages

serial_no [PK] integer	order_id integer	food_id integer	discount integer	quantity integer	amount integer	start_time time without time zone	end_time time without time zone
1	1	1	0	3	330	21:27:47.95869	21:32:22.715807
2	1	4	0	2	560	21:27:47.95869	21:32:22.715807
3	1	7	0	4	1000	21:27:47.95869	21:38:27.514731

Now the chef has served the 3rd item also. We can see all the end_times are updated and serial_no 3 has different end_time as it is served later.

After the end_time calculation discount or bonus is calculated using triggers.

In this case all the items are served before time so the discount is 0.

And we can see that bonus is added to chef below

```
1035  select * from chef
```

Data Output | Notifications | Messages

	chef_id [PK] integer	category_id [PK] integer	bonus integer
7	1	1	2
8	4	4	2
9	2	5	0
10	3	7	2

5. **Bill generation and Payment. (Done by: dining supervisor)**

The final bill for the total order done by the particular customer is generated.
In bill calculation,
total cost of food items = (quantity*cost - 0.05(quantity*cost)(only if prep_time<end_time-start_time)) + tax (18% of total cost).As payment is done the supervisor enters the bill_id, and payment mode then it gets uploaded into the payment table the payment_id, bill_id, payment_method, and date_of_payment.Now Emily has finished her dinner and she goes back to the supervisor for bill.
Let Emily's payment mode be phonepay. She tells her order_id which is 1 to the supervisor.

```
1031  call bill(1,'phonepay');
1032
1033  select * from bill
```

Data Output | Notifications | Messages

	bill_no [PK] integer	total_amount bigint	tax double precision	discount double precision	net_amount bigint	table_id character varying (8)	cust_id integer	order_id integer
1	1	1890	340.2	0	2230	T7	1	1

This bill is generated and given to emily.
Also the bill no and mode are inserted into payment using triggers for manager records.

```
1040  select * from payment
```

Data Output | Notifications | Messages

	date_of_payment date	payment_id [PK] integer	payment_mode character varying (10)	bill_no integer
1	2024-04-30	1	phonepay	1

6. Deallocating of the table (Done by: Due to trigger after bill generation)

After paying the bill. Emily leaves so as soon as the bill is generated the table gets deallocated and can be reassigned later.

```
1042  select * from dining_supervison;
```

Data Output | Notifications | Messages

	table_id [PK] character varying (8)	capacity integer	booking_stat character varying (10)
11	T12	8	not_booked
12	T13	2	not_booked
13	T14	4	not_booked
14	T7	6	not_booked

The T7 table is back to not_booked.

—> **Done by manager:**

The manager inputs the food id, quantity

If quantity given > total_order in the food table then price is decremented by 5% as this means demand is less else incremented by 5%.

This increase log is updated in the price_change table.

```
1044
1045  select * from price_change
```

Data Output | Notifications | Messages

	food_id [PK] integer	cur_date [PK] timestamp without time zone	before_value integer	after_value integer
1	4	2024-04-30 22:03:26.390671	280	266

Also, it is updated in the food table

```
1042  select adjust_price(4,6);
1043  select * from food where food_id = 4
```

Data Output | Notifications | Messages

	food_id [PK] integer	food_name character varying (100)	price integer	total_order integer
1	4	Lasagna	266	2

- ## Working of Roles:

```
048  set role customer;
049  call Table_allocation(1,3);
```

Data Output | Notifications | Messages

```
ERROR:  permission denied for table dining_supervison
CONTEXT:  SQL statement "SELECT capacity, table_id, booking_stat FROM dining_supervison order by capacity"
PL/pgSQL function table_allocation(integer,integer) line 7 at FOR over SELECT rows

SQL state: 42501
```

When assigning the customer role, access is limited to the food table. However, when attempting to call the Table_allocation function, which is accessible only to the dining supervisor, an error occurs due to no grant access to the Table_allocation for the customer.

```
1048  set role chef;
1049  update employee set emp_salary = emp_salary+10000;
```

Data Output | Notifications | Messages

```
ERROR:  permission denied for table employee

SQL state: 42501
```

When assigning the chef role, access is not given to the employee table for the chef. So when attempting to update the employee table, an error occurs due to no grant access to the employee table for the chef.

```
1048  set role manager;
1049  select * from employee;
```

Data Output | Notifications | Messages

	emp_id [PK] integer	emp_fname character varying (10)	emp_lname character varying (10)	designation character varying (10)	address character varying (50)	city
1	1	John	Smith	chef	St101	Mumbai
2	2	Emily	Taylor	chef	St105	Kolkata
3	3	Michael	Clark	chef	St106	Hyderab

When assigning the manager role, access is given to all tables in the database, and all privileges are granted to him. So when attempting to select the data from the employee table, no error occurs because all privileges and access are given to the manager.

```
1051  set role server;
1052  call bill(2,'creditcard');
```

Data Output | Notifications | Messages

```
ERROR:  permission denied for table bill
CONTEXT:  SQL statement "insert into bill (total_amount,tax,discount,net_amount,table_id,cus
PL/pgSQL function bill(integer,character varying) line 17 at SQL statement
```

When assigning the server role, access is not given to the billing procedure. So when attempting to call the bill procedure, we get an error because no permission is given to server on billing procedure.

```
1054  set role dining_supervisor;
1055  call bill(1,'creditcard');
1056
```

Data Output | Notifications | Messages

```
CALL

Query returned successfully in 88 msec.
```

Here we have set the role as dining supervisor, as we know that the dining supervisor has access to the bill table, the function executes without any issue.