CSE

4126

DISTRIBUTED DATABASE MANAGEMENT SYSTEM LAB

FINAL PROJECT REPORT

Restaurant Management System

SUBMITTED BY

MD. MASUD RANA | 15-01-04-134

Summary:

Restaurants are all over the world. A popular restaurant has more than one branches. These branches depend on many processes that work in sync with each other and also can be a lot far from each other (like in different states). Managing these branches data in one site is very difficult. This problem can be solved using distributed database management system. It is reliable and makes it easy to manage data of all the branches. Therefore, we have designed this distributed database management system for multi branches Restaurant Company.

We have tried to fit in all the things, which we have learned about DDB from class and lab. We have implemented Horizontal and vertical fragmentation, allocation schema, distributed transparency, functions, procedures triggers, effects of update, query transformation and some basic function of restaurant management.

In this group project, we divided our project in 3 different part for our three members.

- Employee.
- Customer and Order.
- Ingredient.

I have done **customer and order** part. For this part I have made some function, procedure, query and trigger namely:

- amount_sale_in_a_date.
- find_customers.
- find_customers_of_a_date.
- get_order_info.
- insert_order.
- make_bill.

All of my function, procedure, query, trigger details and also the difference types of fragmentation, query transformation and distributed transparency are given blew.

Entity set:

- branch - Keeps information of the branches of the restaurant.
- customer - Keeps information of the customers.
- employee- Keeps information of the employees.
- type- Keeps employee types and their information
- foodCategory – Keeps information of the food categories.
- foodItem- Keeps information of the food items.
- makeOrder- Keeps information of the customer orders.
- orderDetails- Keeps all the ordered items details of a customer order.
- ingredients- Keeps information of the ingredients.
- Ing_details- Keeps details ingredients purchased and remained.
- Ing_daily_used- Keeps details of the ingredients, which have been used.

Global Schema:

- branch(branch_id, branch_name, branch_address, phone)
- customer(customer_id, branch_id, name , phone)
- employee(employee_id, type_id, branch_id, employee_name, age, address, sex, email, phone)
- type(type_id, type_name, salary)
- foodCategory(c_id, c_name)
- foodItem(food_id, category_id, f_name, price, discount, f_details)
- makeOrder(order_id, customer_id, amount, order_date)
- orderDetail(order_id, food_id, quantity)
- ingredients(i_id, name)
- ing_details(entry_id, i_id, branch_id, total_cost, quantity_remained, entry_date, expiry_date, quantity_brought)
- ing_daily_use(use_id, entry_id, used_date, quantity_used)

Fragmentation Schema:

- $employee_1$ = employee $SJ_{employee.branch\_id=branch.branch\_id}$ $SL_{branch\_name='Dhaka'}$ branch
- $employee_2$ = employee $SJ_{employee.branch\_id=branch.branch\_id}$ $SL_{branch\_name='Comilla'}$ branch
- $employee_3$ = employee $SJ_{employee.branch\_id=branch.branch\_id}$ $SL_{branch\_name='Chittagong'}$ branch

- $ing\_details_1$ = PJ $_{entry\_id, i\_id, branch\_id, total\_cost, quantity\_remained}$ ing_details
- $ing\_details_2$ = PJ $_{entry\_id, entry\_date, expiry\_date, quantity\_brought}$ ing_details

Allocation Schema:

There are two sites in our project. (Named as host and site_link)

- $employee_1$ @ site_link
- $employee_2$ @ host, site_link
- $employee_3$ @ host

- $ing\_details_1$ @ host
- $ing\_details_2$ @ site_link

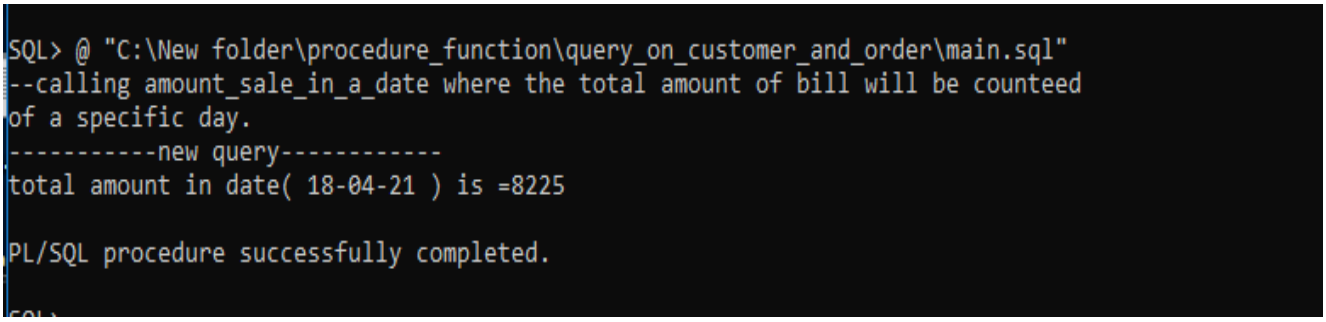
Function/Procedure/Trigger:

Customer and Order:

- **amount_sale_in_a_date:**

  This function will give you the total amount of sale of all branches of the restaurant of a specific date.

  - Input: Date.
  - Output: Total amount of sale for the restaurant on that day.
  - Transparency Level: 1.



- **find_customers:**

  This function will give you all the names of the customer who have ordered the more than a specific amount.

  - Input: Bill amount.
  - Output: Customer Names.
  - Transparency Level: 1.

```
SQL> @ "C:\New folder\procedure_function\query_on_customer_and_order\main.sql"
--calling fund_customers returns name of customers if the bill amount >1000.
-----------new query------------
ratul khan
Md. Mithun Ali
Bahadur Ali
Ozil
lukaku

PL/SQL procedure successfully completed.

SQL>
```

- **find_customers_of_a_date:**

  This function will give you all the names of the customer who have ordered in a specific date.

  - Input: Date.
  - Output: Customer Names.
  - Transparency Level: 1.

```
SQL> @ "C:\New folder\procedure_function\query_on_customer_and_order\main.sql"
--calling find_customers_of_a_date which returns all customer of a specific day.
-----------new query------------
ratul khan
Bahadur Ali
Ozil
lukaku

PL/SQL procedure successfully completed.

SQL>
```

- **get_order_info:**

  This function will give you all the ordered food items, quantity and total cost of the specific order. It can be considered as a cart.

  - Input: Order id.
  - Output: Food item id, name, ordered quantity, unit price and total price.
  - Transparency Level: 1.

```
SQL> @ "C:\New folder\procedure_function\query_on_customer_and_order\main.sql"
------calling get_order_info---------------
5 4 THAI TUNA SALAD 3 520
5 6 ITALIAN ICE CREAM 3 195
5 7 450G BABY CHICKEN 1 575
total cost of order 5 =2720

PL/SQL procedure successfully completed.
```

- **insert_order:**

  This function inserts two ordered food items with their quantity for a customer of a branch.

  - Input: Branch id, customer id, first food item id, first items quantity, second food item and second items quantity.
  - Output: Confirmation of Insert.
  - Transparency Level: 1.

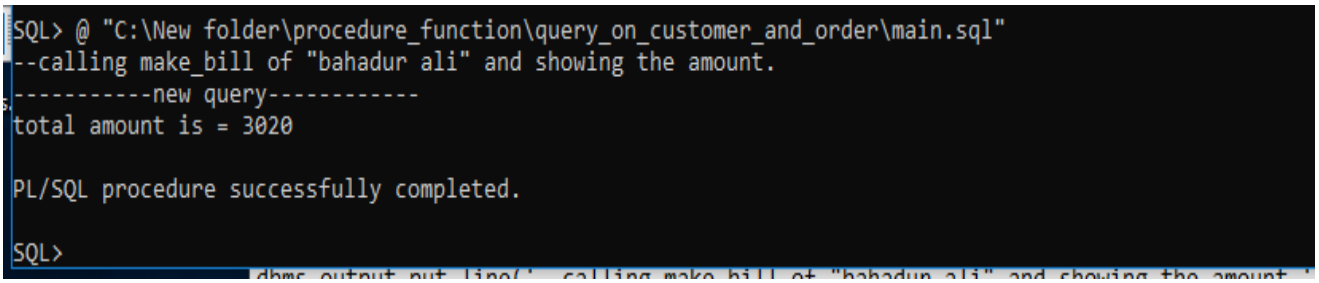
```
SQL> @ "C:\New folder\procedure_function\query_on_customer_and_order\main.sql"
--calling insert order procedure to insert the order into databas
-----------new query------------
10 3 BIG CHICK BURGER 3 575
10 1 JUICY LUCY BURGER 1 495
total cost of order 5 =2220
successfull insertion
succesful

PL/SQL procedure successfully completed.
```

- **make_bill:**

  This function gives you total amount of foods ordered by a customer.

  - Input: Customer id.
  - Output: Total amount of ordered.
  - Transparency Level: 1.



Employee:

- delete_employee_type_checks_fk.
- delete_employee1_site1.
- insert_employee.
- transfer_employee.
- trigger_employee2_update_type_id.
- update_employee2_type_id.
- employees_name_given_type.
- employees_given_type_and_branch.

Ingredient:

- insert_ingredient
- insert_ingredient_detail
- insert_used_info
- notify_expiry_date
- trigger_before_update_ing_details_1

Effect of update:

Transfer an employee to Comilla branch who's employee id= 2.

Now employee id= 2 has branch_id=1 and it is fragmented as $employee_1$ which is allocated on site_link.

Step-1: Store the data of employee_id=2 from any site of $employee_1$ fragment.

$employee_1$ @ site_link.

employee_id	type_id	branch_id	employee_name	Age	address	Sex	email	phone
2	2	1	Masud	25	27/B kakrail	M	Masud22@gmail.com	01738028170

Now, Comilla has branch_id=3 and it is fragmented as $employee_2$ which is allocated on both host and site_link.

Step-2: Insert the stored data to all sites of $employee_2$ fragment.

$employee_2$ @ host

employee_id	type_id	branch_id	employee_name	Age	address	Sex	email	phone
2	2	3	Masud	25	27/B kakrail	M	Masud22@gmail.com	01738028170

$employee_2$ @ site_link

employee_id	type_id	branch_id	employee_name	Age	address	Sex	email	phone
2	2	3	Masud	25	27/B kakrail	M	Masud22@gmail.com	01738028170

Step-3: Delete the data of employee_id=2 from $employee_1$ fragment.

$employee_1$ @ site_link.

employee_id	type_id	branch_id	employee_name	Age	address	Sex	email	phone
~~2~~	~~2~~	~~1~~	~~Masud~~	~~25~~	~~27/B kakrail~~	~~M~~	~~Masud22@gmail.com~~	~~01738028170~~

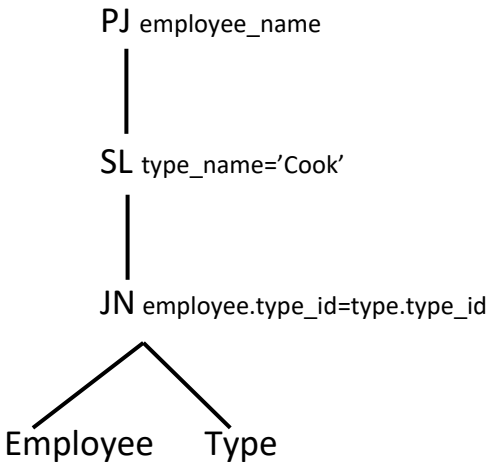
Query Transformation:

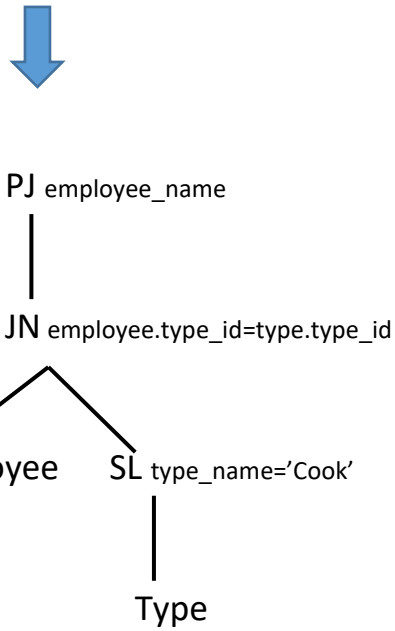
Here is a query which we have transformed into its fragments. It is used in the 'employees_name_given_type' function.

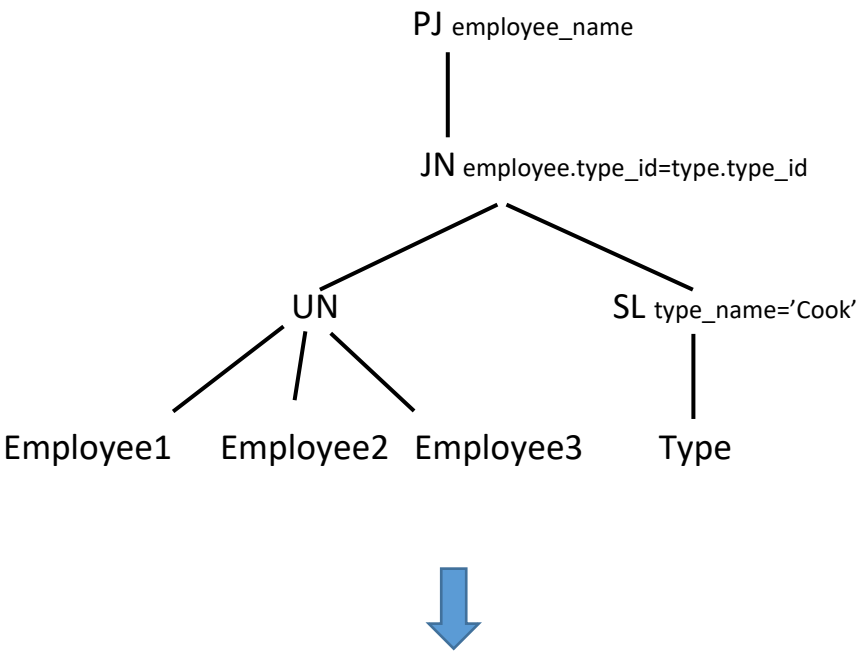
Query:

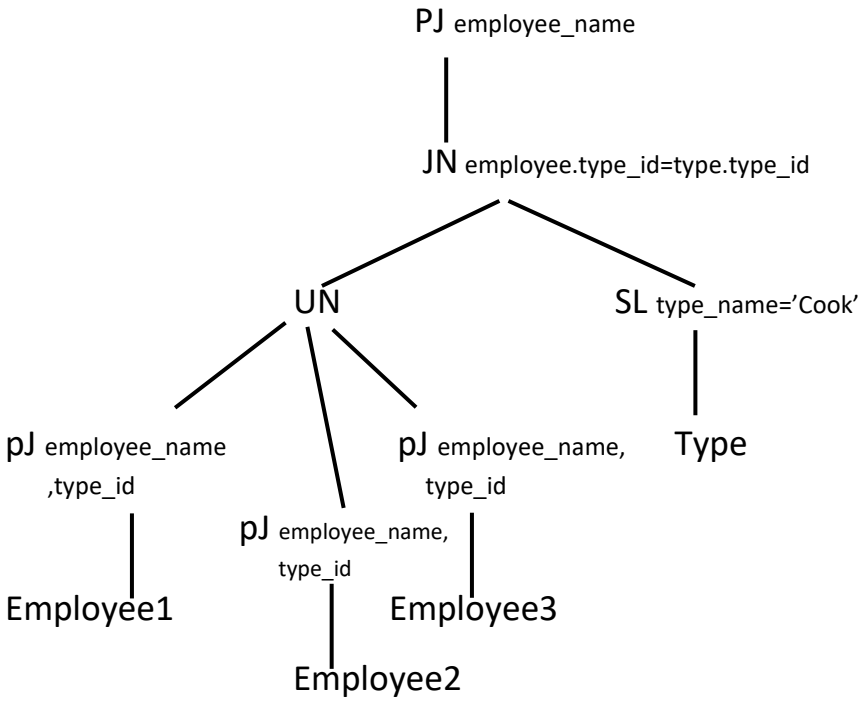
PJ employee_name SL type_name='cook' (Employee NJemployee.type_id=type.type_id Type)

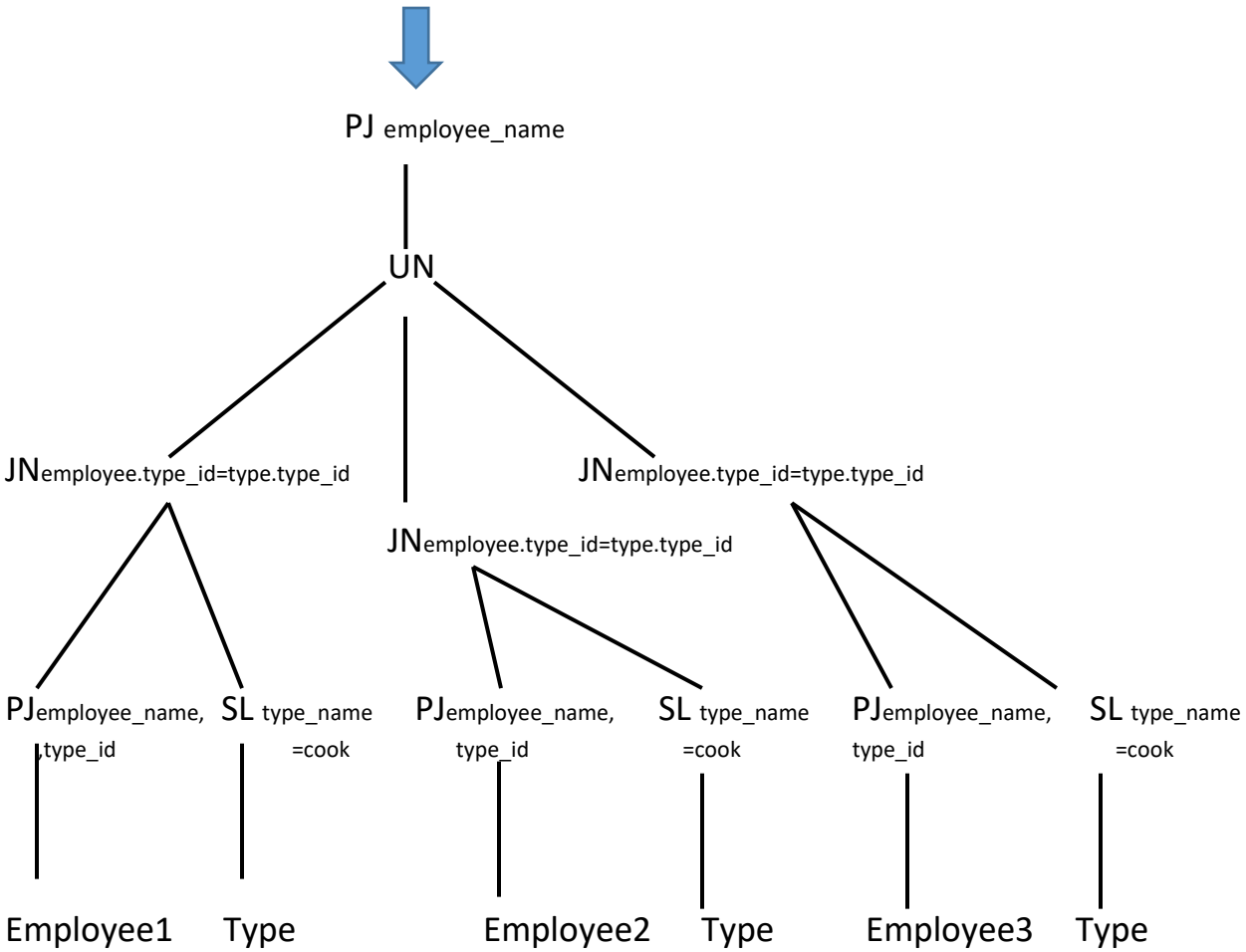
Operator Tree:




PJ employee_name
JN employee.type_id=type.type_id
Employee
SL type_name='Cook'
Type


PJ employee_name
JN employee.type_id=type.type_id
UN
SL type_name='Cook'
Employee1
Employee2
Employee3
Type


PJ employee_name
JN employee.type_id=type.type_id
UN
SL type_name='Cook'
pJ employee_name ,type_id
pJ employee_name, type_id
pJ employee_name, type_id
Type
Employee1
Employee3
Employee2


PJ employee_name
UN
JNemployee.type_id=type.type_id
JNemployee.type_id=type.type_id
JNemployee.type_id=type.type_id
PJemployee_name, ,type_id
SL type_name =cook
PJemployee_name, type_id
SL type_name =cook
PJemployee_name, type_id
SL type_name =cook
Employee1
Type
Employee2
Type
Employee3
Type

OPTIMIZED QUERY:

$PJ_{employee\_name}$(

((PJemployee_name,type_idEmployee1) $NJ_{employee.type\_id=type.type\_id}$ (SL type_name=cook Type))

UN

((PJemployee_name,type_idEmployee2) $NJ_{employee.type\_id=type.type\_id}$ (SL type_name=cook Type))

UN

((PJemployee_name,type_idEmployee3) $NJ_{employee.type\_id=type.type\_id}$ (SL type_name=cook Type))

)

Conclusion:

We have tried to include all the things we learned in this course. This project helped us to understand how exactly how the DDB works and how to make it efficient for users. We also learned that which things should be given priority. We will work on it further to give it completeness.