CSE 4126

DISTRIBUTED DATABASE MANAGEMENT SYSTEM LAB

FINAL PROJECT REPORT

Restaurant Management System

SUBMITTED BY

AMIR HAMZA | 15-01-04-118

Executive Summary of Project:

With the flow of technology it is now become essential that the management system should be autonomous and computing should be fluent.
Restaurants are increasing rapidly in Bangladesh. New management systems are introduced every day. But efficiently managing the database is much important. Our main goal is to implement a project which can manage several database entity of a distributed database system. Managing optimal query for a distributed system is also a challenge. Managing optimality is also focused here. Therefore, our project focuses on reliable and easy management of all the branches by merging new branches data with existing branches data. Manage overall system of the restaurant with food information, food category, name of the food, prize, offers, ingredient information, creating cart etc. Also the management of the employee such as employee information's, salary, employee type etc. Here each or this necessity we have designed this distributed database management system for Restaurant Company.

Contribution into the project:

I have done the fragmentation, query optimization, data insert and function, trigger and procedure, which are related to employee table. I have added screenshot of my work under every procedure and function which are done by me.

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:

Employee:

- **delete_employee_type_checks_fk:**

  It deletes an employee type from 'Type' table if there is no employee of that given type. If there remains at least one employee of given type it will not delete the employee type. Here foreign key is checked manually.

  - Input: Employee type id.
  - Output: Confirmation of deletion.
  - Transparency Level: 1.

  A screen shot of delte_employee_type_check_fk procedure is given below:

```
SQL> select * from type@site_link;

   TYPE_ID TYPE_NAME                SALARY
---------- -------------------- ----------
         1 Manager                   30000
         2 Cook                      25000
         3 server                    18000
         4 cleaner                   15000
         5 guard                     10000

SQL> @ "C:\New folder\procedure_function\query_link_on_employee\delete_employee_type_checks_fk.sql"
enter type_id to delete : 4
old  14: b:='&type_id';
new  14: b:='         4';
the type is reffered in employee_table you can not delete it

PL/SQL procedure successfully completed.

SQL> @ "C:\New folder\procedure_function\query_link_on_employee\delete_employee_type_checks_fk.sql"
enter type_id to delete : 5
old  14: b:='&type_id';
new  14: b:='         5';
successful deletation

PL/SQL procedure successfully completed.

SQL> select * from type@site_link;

   TYPE_ID TYPE_NAME                SALARY
---------- -------------------- ----------
         1 Manager                   30000
         2 Cook                      25000
         3 server                    18000
         4 cleaner                   15000
```

Fig1: delte_employee_type_check_fk

- **delete_employee1_site1:**

  It will delete an employee of branch 'Dhaka' from 'Employee$_1$' fragment table from site_link which shows a level 3 transparency.

  - Input: Employee id
  - Output: Confirmation of delete.
  - Transparency Level: 3.

Another screenshot of delete_employee1_site1 is given below:

```
SQL> select * from employee1@site_link;

EMPLOYEE_ID    TYPE_ID  BRANCH_ID EMPLOYEE_NAME              AGE
---------- ---------- ---------- -------------------- ----------
ADDRESS              SEX   EMAIL                PHONE
-------------------- ----- -------------------- ------------
         1          2          1 Hamza                        22
9-south Mugdha       M     ah226038@gmail.com   01851152343

         2          2          1 Masud                        25
27/B kakrail         M     Masud22@gmail.com    01738028170

         3          1          1 Sujan                        25
27/B polton          M     Sujan23@gmail.com    017323983370


SQL> @ "C:\New folder\procedure_function\query_link_on_employee\delete_employee1.sql"
enter employee_id of employee1 fragment at site1 : 3
old   8: a:='&e_id';
new   8: a:='         3';

PL/SQL procedure successfully completed.

SQL> select * from employee1@site_link;

EMPLOYEE_ID    TYPE_ID  BRANCH_ID EMPLOYEE_NAME              AGE
---------- ---------- ---------- -------------------- ----------
ADDRESS              SEX   EMAIL                PHONE
-------------------- ----- -------------------- ------------
         1          2          1 Hamza                        22
9-south Mugdha       M     ah226038@gmail.com   01851152343

         2          2          1 Masud                        25
27/B kakrail         M     Masud22@gmail.com    01738028170


SQL>
```

Fig2: delete_employee1_site1

- **insert_employee:**

  It will insert an employee data to its suitable fragment table to all its allocated sites.

  - Input: Employee name, type id, branch name, age, address, sex, email, phone.
  - Output: Confirmation of Insert.
  - Transparency Level: 1.

Another screenshot of insert_employee is given below: Here user interface via console is used.

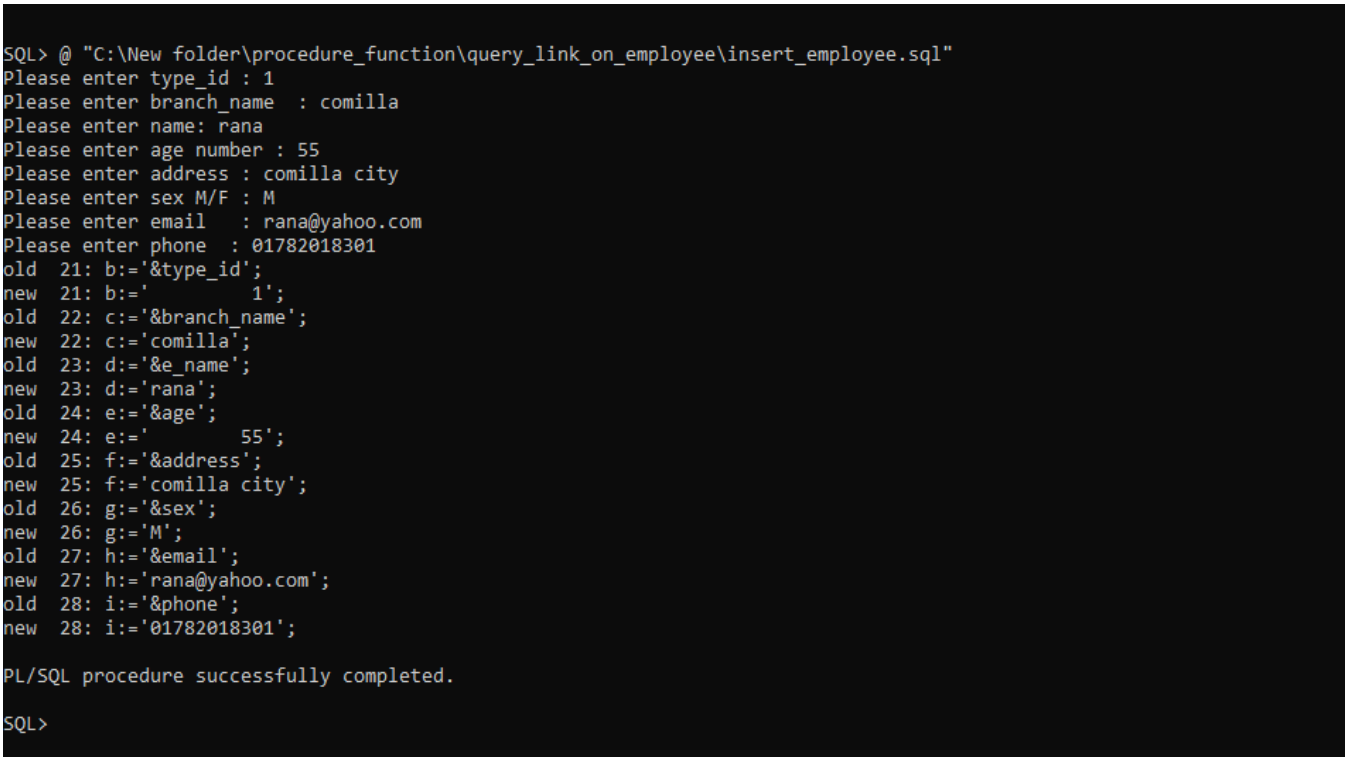


Fig3: insert_employee

- **transfer_employee:**

  It will transfer an employee from one branch to another. It actually inserts that employee to its suitable fragment table to all its allocated sites from one of its previous fragment table allocated site and then delete that from previous fragment all its allocated sites.

  - Input: Employee id and branch name(branch where that employee will be transferred)
  - Output: Confirmation of Transfer completion.
  - Transparency Level: 1 (Effect of Update).

Another screenshot of transfer_employee is given below: Here user input from console is used.Here employee with id=2 is transferred to comilla branch.

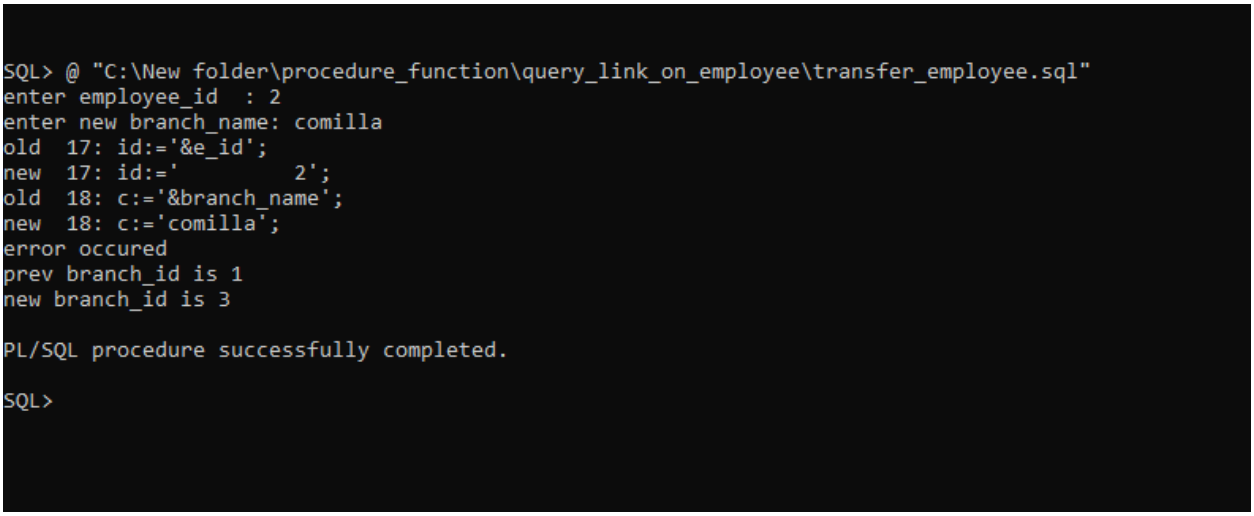


Fig4: transfer_employee

- **trigger_employee2_update_type_id:**

  It's a trigger function which will be called after every update query in 'empoyee$_2$' fragment. 'empoyee$_2$' fragment has been allocated both sites(host and site_link). Therefore, update on this fragment will have effect of both the sites. This trigger will make sure that both the sites have same data.

  - Input: No input.
  - Output: Old and new values.

- **update_employee2_type_id:**

  This procedure helps to give promotion or demotions to the 'employee$_2$' fragment table.

  - Input: Employee id and employee type (In which type you want to update) of an employee who's data is in 'employee$_2$' fragment.
  - Output: Confirmation of Transfer completion.
  - Transparency Level: 2.

  Another screen shot of update_employee2_type_id is given below: Here the trigger describe above trigger_employee2_update_type_id is automatically called and it shows the previous information and updated information.

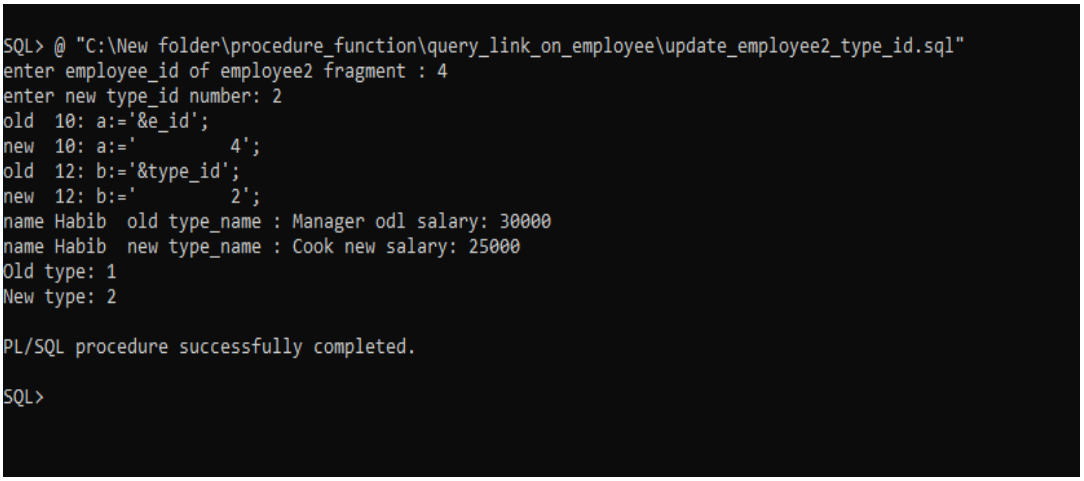


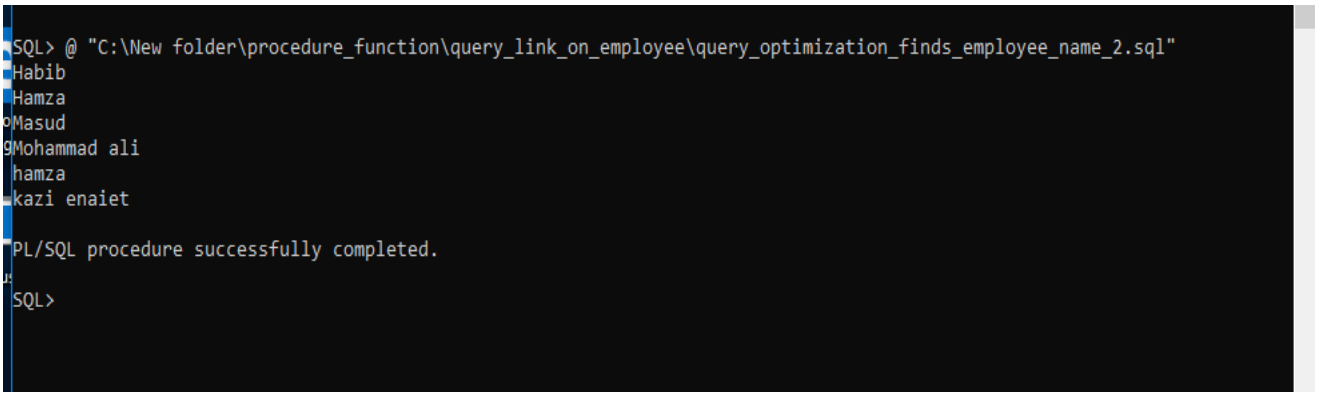
Fig4: update_employee2_type_id

- **Employee_name_given_type:**

This function is used to find an employee who works who's type is 'Manager'.

  - Input: Employee_ type
  - Output: name of the employee who is 'Manager'.
  - Transparency Level: 1.

A screen shot of the function is given below:

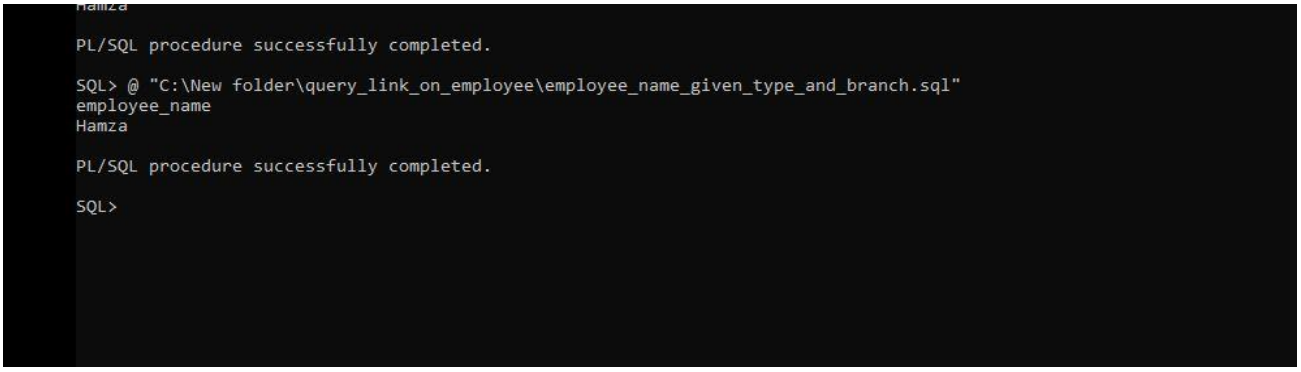


- **Employee_name_given_type_and_branch:**

This function is used to find an employee who works in Dhaka branch and who's type is 'Manager'.

- Input: branch_id and employee_ type
- Output: name of the employee who is in dhaka branch and type is 'Manager'.
- Transparency Level: 1.

A screen shot of the function is given below:



Here the optimization for employee_name_given_branch_and_type is given below:

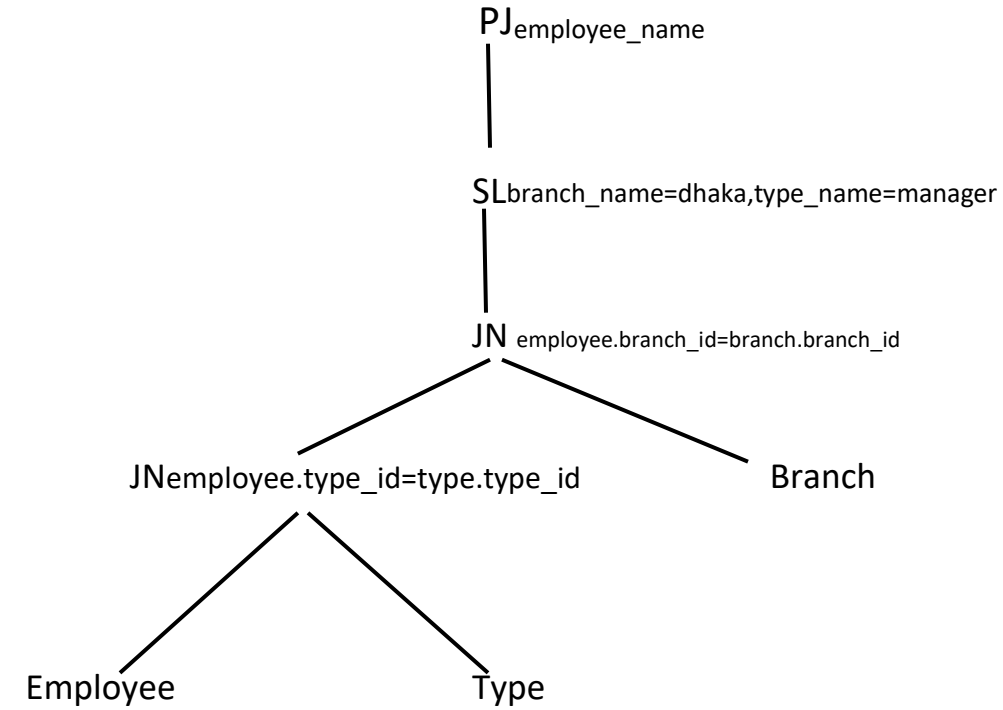
- **Query Transformation:**

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

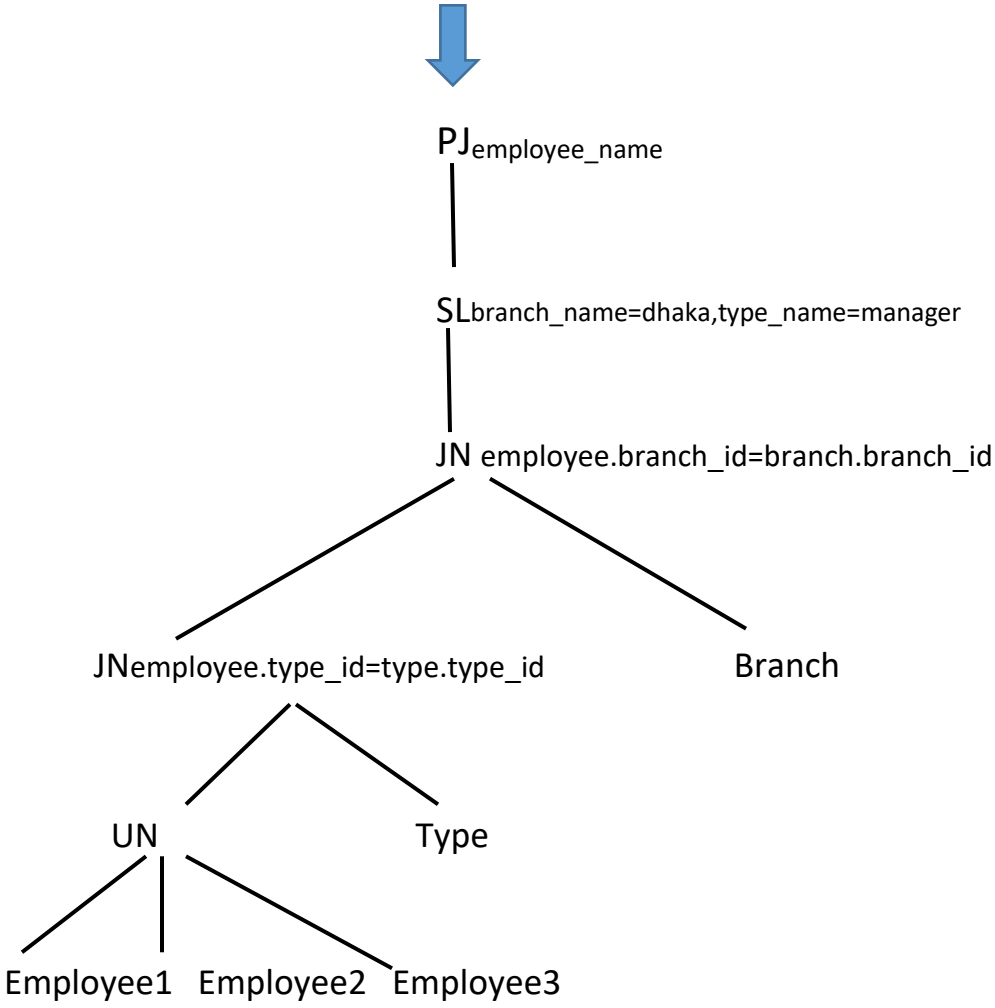
Query:

PJ$_{employee_name}$SL branch_name='dhaka' and type_name='manager'((Employee JN$_{employee.type_id=type.type_id}$ Type) JN$_{employee.branch_id=branch.branch_id}$ Branch)

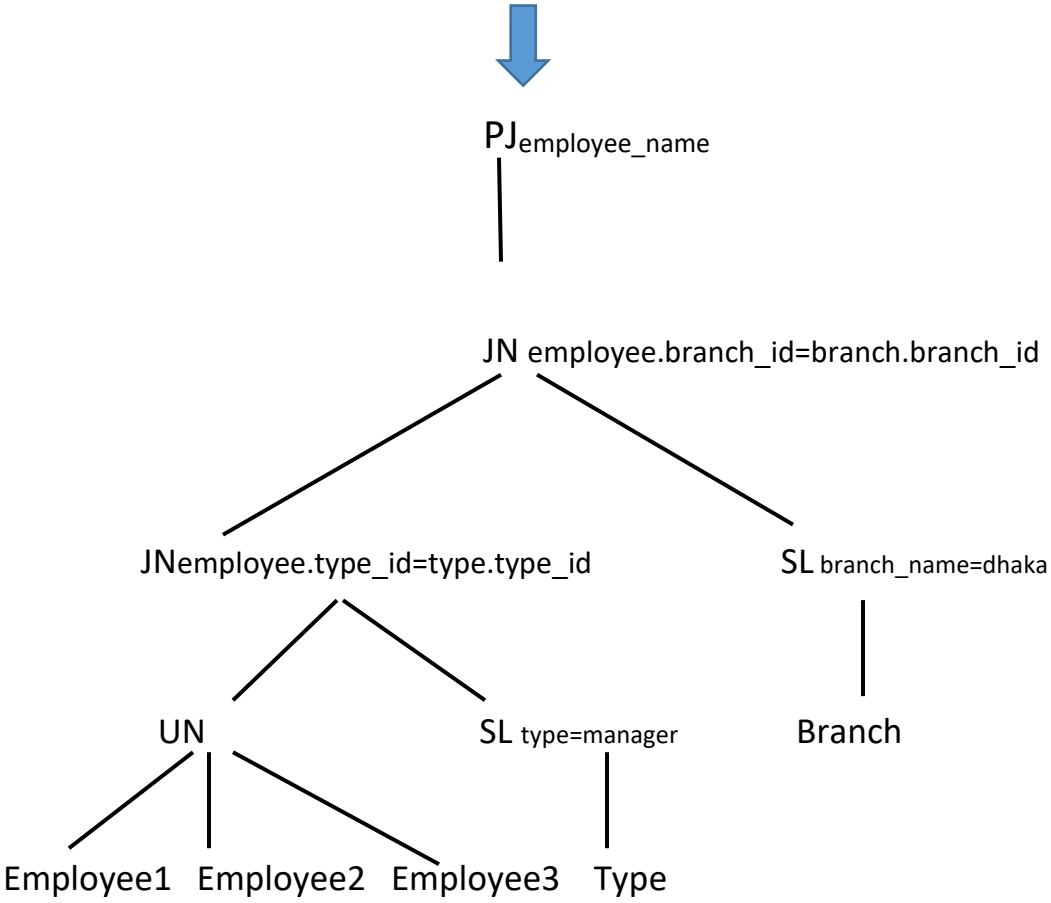
Operator Tree:



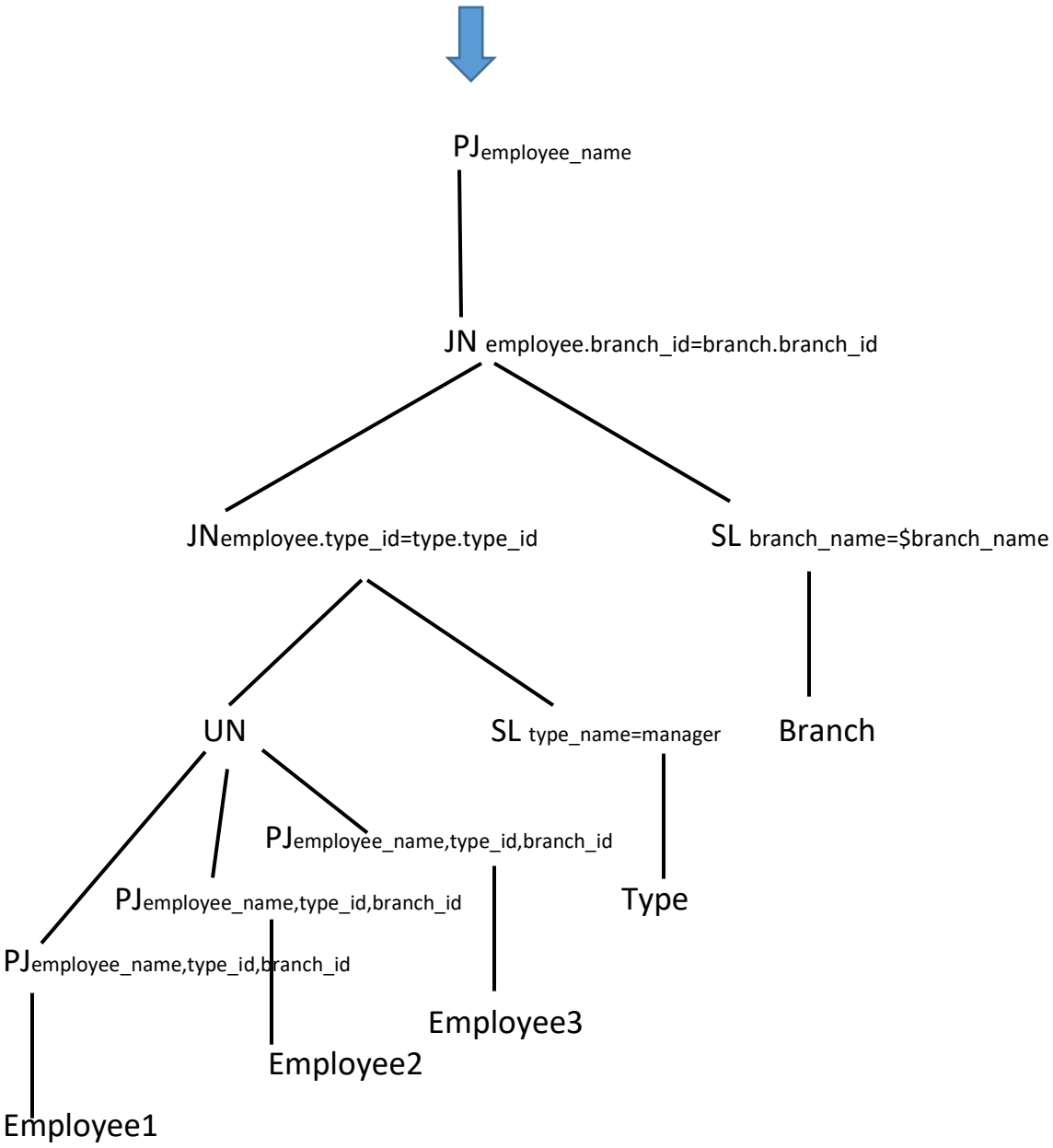
($T_{fragments}$):


PJemployee_name
SLbranch_name=dhaka,type_name=manager
JN employee.branch_id=branch.branch_id
JNemployee.type_id=type.type_id
Branch
UN
Type
Employee1
Employee2
Employee3

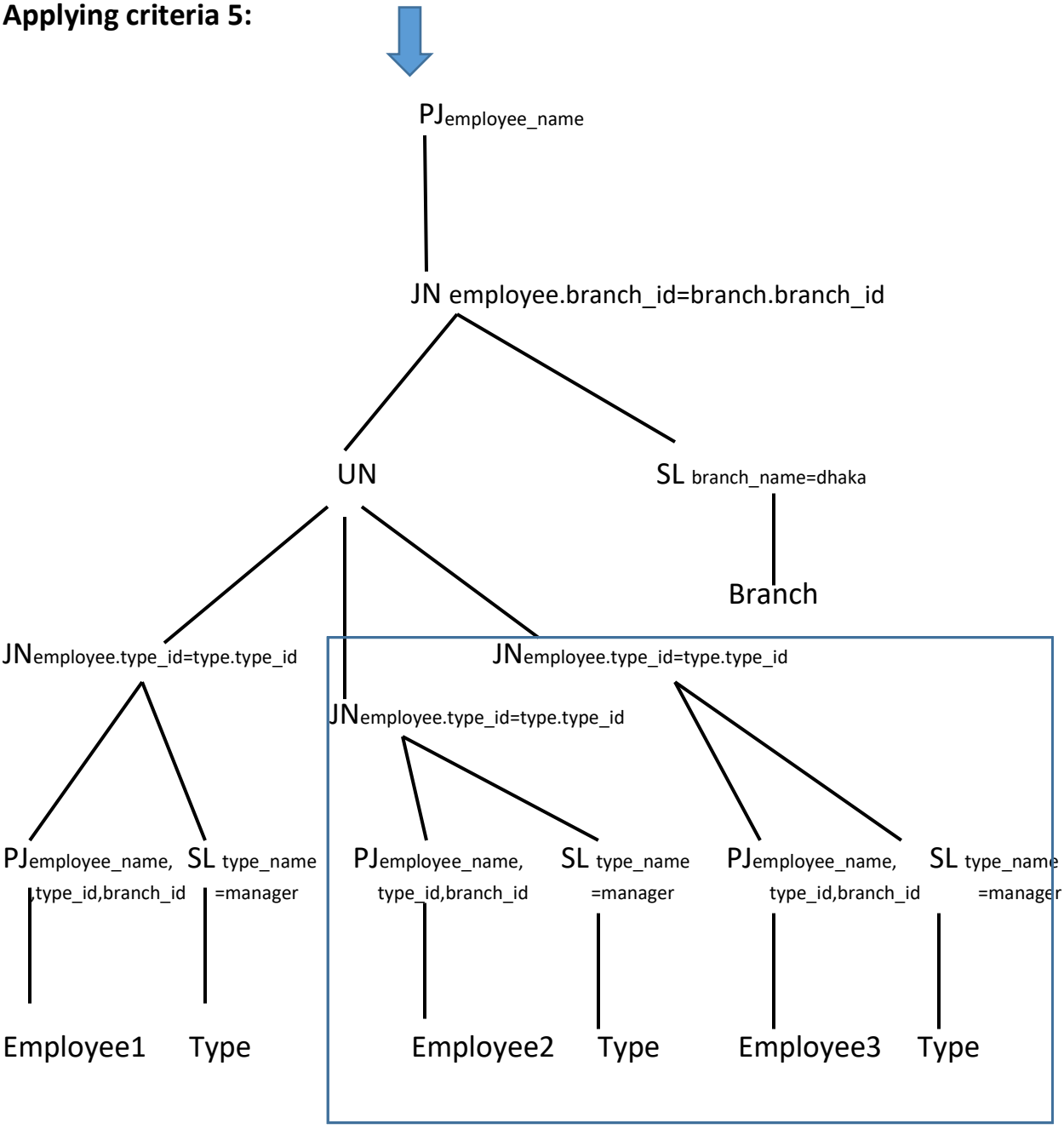
Applying criteria 2:



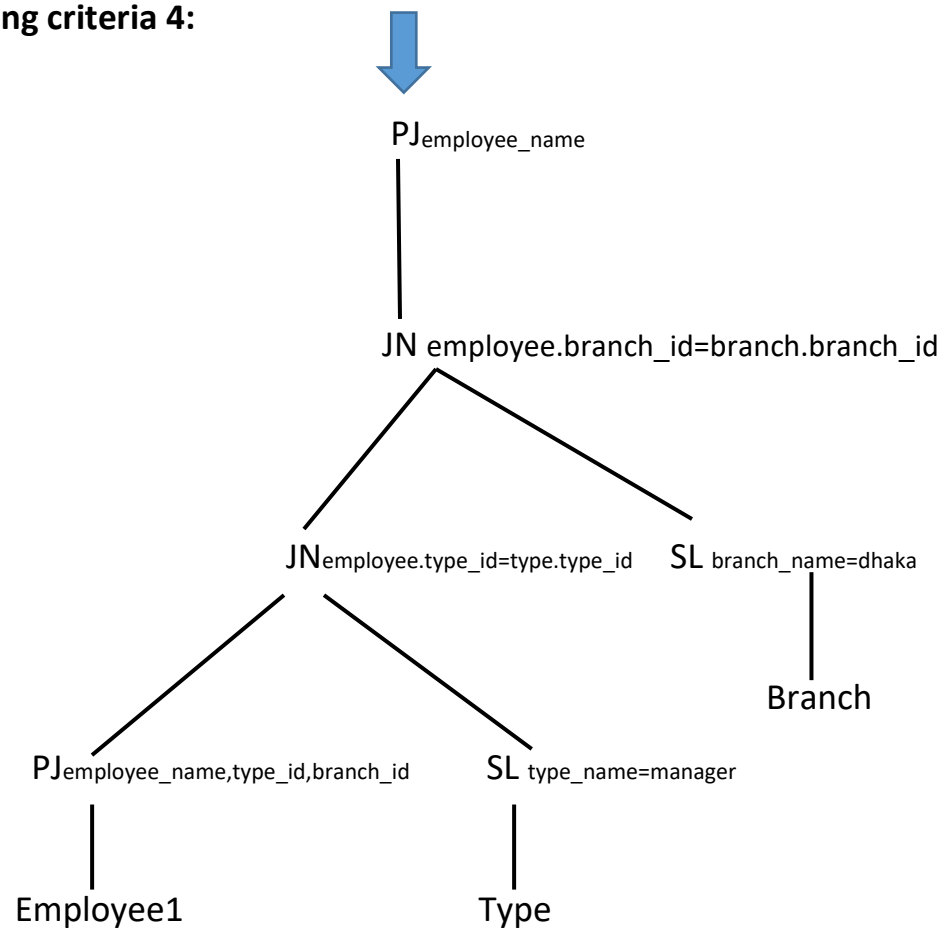
Applying criteria 1:



Applying criteria 5:



Applying criteria 4:



Transformed Query ($Q_{transformed}$):

PJemployee_name(((PJ employee_name,type_id,branch_id Employee1) JNemployee.type_id=type.type_id (SL type_name=manager Type)) JN employee.branch_id=branch.branch_id (SL branch_name=dhaka Branch))

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 host.

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

$employee_3$ @ host.

employee_id	type_id	branch_id	employee_name	Age	address	Sex	email	phone
2	4	1	redwan ahmed	24	bashkhali uttorpara	M	ridwan@gmail.com	01633229923

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	4	3	redwan ahmed	24	bashkhali uttorpara	M	ridwan@gmail.com	01633229923

$employee_2$ @ site_link

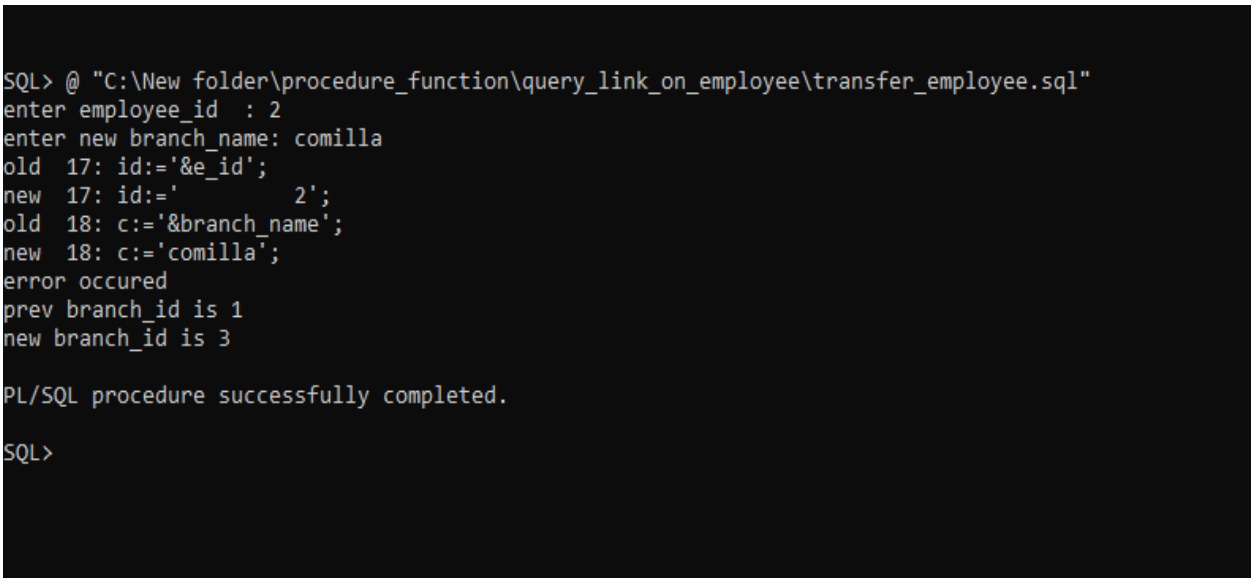
employee_id	type_id	branch_id	employee_name	Age	address	Sex	email	phone
2	4	3	redwan ahmed	24	bashkhali uttorpara	M	ridwan@gmail.com	01633229923

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~~	~~4~~	~~1~~	~~redwan ahmed~~	~~24~~	~~bashkhali uttorpara~~	~~M~~	~~ridwan@gmail.com~~	~~01633229923~~

A screen shot of the update is given below.



Effect of update on Delete_Employee_type_checks_fk

Type_id	Type_name	Salary
5	Guard	10000

Step 1:

Delete the data from employe_type@site_link where type_id=5 if no employee is assigned on this type. Other wise it would not delete the type. Here foreign key constraint is checked.

Type_id	Type_name	Salary
~~5~~	~~Guard~~	~~10000~~

Conclusion:

This project is basically a demonstration of how to handle distributed databases and their transparency over the users. This Restaurant Management System is providing these advances to the users where the query of insert, update, delete and modify has the interfaces where level 1, level 2 and level 3 transparency are shown. We have managed the operations with raw codes. Here we have managed employee into three horizontal fragments of employee1, employee2, employee3 and ingredient_details into two vertical fragments which are ingredient_details1 and ingredient_details_2. It is now a completely working project. Hope that it will help us to do work fluently in future.