

RESTAURENT MANAGEMENT SYSTEM

AUST CSE 4126 Spring 2018 - Project Report

Roll: 15.01.04.066(Khandakar Mahmudur Rahman)

15.01.04.070, 15.01.04.080

Introduction:

“Restaurant management system” is a management system for managing the branches of the restaurant such as daily sales, working schedule, coordinating the employees etc. It is very common to use a centralized database in a restaurant. But for a successful restaurant it is common to have many branches in different cities or locations or in different country. So, in this case, it is badly in need of a distributed database management system to manage and look over the whole database management system. Distributed database management system not only makes the system easier but also makes operation faster to operate in a particular location. Searching for a particular data in the whole database is unnecessary and complicated for the operating system as well. So, we can avoid such inessential operations just by making the database system distributed.

Overview of the System:

Restaurant Management System is an automation system for a restaurant which performs different functionalities such as storing customer info, branch info, sale info, employee info etc. Depending on this management system many activities can be performed in future like notifying customers about new items or offers on food items or new branch opening through email or phone etc.

The flow of this system is that, the librarian logs into the system. Now, different branches have different managers. They store daily sale info, employee info and customer info. According to 'ID', customers and employees are distributed in different sites.

Features and Functionalities –

Storing information about customers, employees, menu items, categories etc. Operations like searching particular food items, inserting in branches, updating and delete information,

creating fragments of 9 global relations for different sites and assigning data in corresponding sites.

User –

right now the user is only the manager of the brunch and admin.

Sites –

There are two sites in this management system. They are distributed location wise to operate the system fast and with less complexity.

Relations and Sites:

Global Relations:

employee(EID, EName, Address, Phone, Position, Salary)

catagories(CID, CName)

menuitems(ItemID, ItemName, Price, Description, CID)

customer(CustomerID, CustomerName, Address)

reservation(RID, duration, RDate, RTime, CustomerID,)

diningTables(DID, WID, ChairCount)

orders(OID, ODate, Quantity, CustomerID, ItemID)

diningTableTrack(Serial,OID, DID, TableServesDate, TableServesTime)

reserves(RID, DID)

bills(BID, OID, CustomerID, Discount, Amount, BDate, BTime)

Fragmentation Schema:

employee1=SL_{EID<3000}(employee)

employee2=SL_{EID>3000}(employee)

orders1=SL_{OID<=2298}(orders)

orders2=SL_{OID}>=2298(orders)

catagories1=SL_{CID}<3000(catagories)

catagories2=SL_{CID}>3000(catagories)

menuitems1=SL_{ItemID}<3000(menuitems)

menuitems2=SL_{ItemID}>3000(menuitems)

customer1=SL_{CustomerID}<3000(customer)

customer2=SL_{CustomerID}>3000(customer)

reservation1=SL_{RID}<3000(reservation)

reservation2=SL_{RID}>3000(reservation)

diningTables1=SL_{DID}<3000(diningTables)

diningTables2=SL_{DID}>3000(diningTables)

diningTableTrack1=SL_{Serial}<3000(diningTableTrack)

diningTableTrack2=SL_{Serial}>3000(diningTableTrack)

bills1=SL_{BID}<3000(bills)

bills2=SL_{BID}>3000(bills)

Allocation Schema:

There are two sites.

Site 1(Dhanmondi): employee1, orders1, catagories1, menuitems1, customer1, reservation1, diningTables1, diningTableTrack1, bills1, reserves.

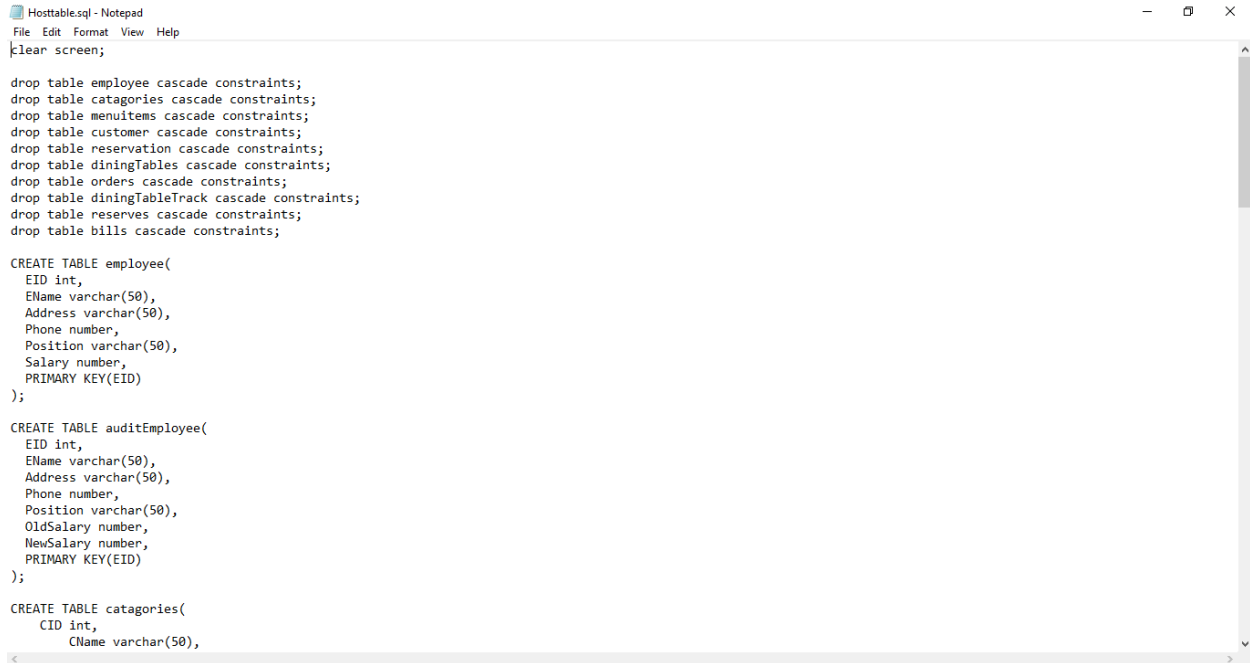
Site 2(Beliroad):): employee2, orders2, catagories2, menuitems2, customer2, reservation2, diningTables2, diningTableTrack2, bills2, reserves.

Contribution:

My contributions in this project are to implement 'creating database', 'Procedure', 'Fragmentation', 'Trigger'.

1. Creating database:

This part of my work was creating database and tables and also inserting values.



```
HostTable.sql - Notepad
File Edit Format View Help
clear screen;

drop table employee cascade constraints;
drop table categories cascade constraints;
drop table menuitems cascade constraints;
drop table customer cascade constraints;
drop table reservation cascade constraints;
drop table diningTables cascade constraints;
drop table orders cascade constraints;
drop table diningTableTrack cascade constraints;
drop table reserves cascade constraints;
drop table bills cascade constraints;

CREATE TABLE employee(
    EID int,
    EName varchar(50),
    Address varchar(50),
    Phone number,
    Position varchar(50),
    Salary number,
    PRIMARY KEY(EID)
);

CREATE TABLE auditEmployee(
    EID int,
    EName varchar(50),
    Address varchar(50),
    Phone number,
    Position varchar(50),
    OldSalary number,
    NewSalary number,
    PRIMARY KEY(EID)
);

CREATE TABLE categories(
    CID int,
    CName varchar(50),
```

Fig:1

2. Procedure:

Pro1.sql defines which item had sold the most between two dates.

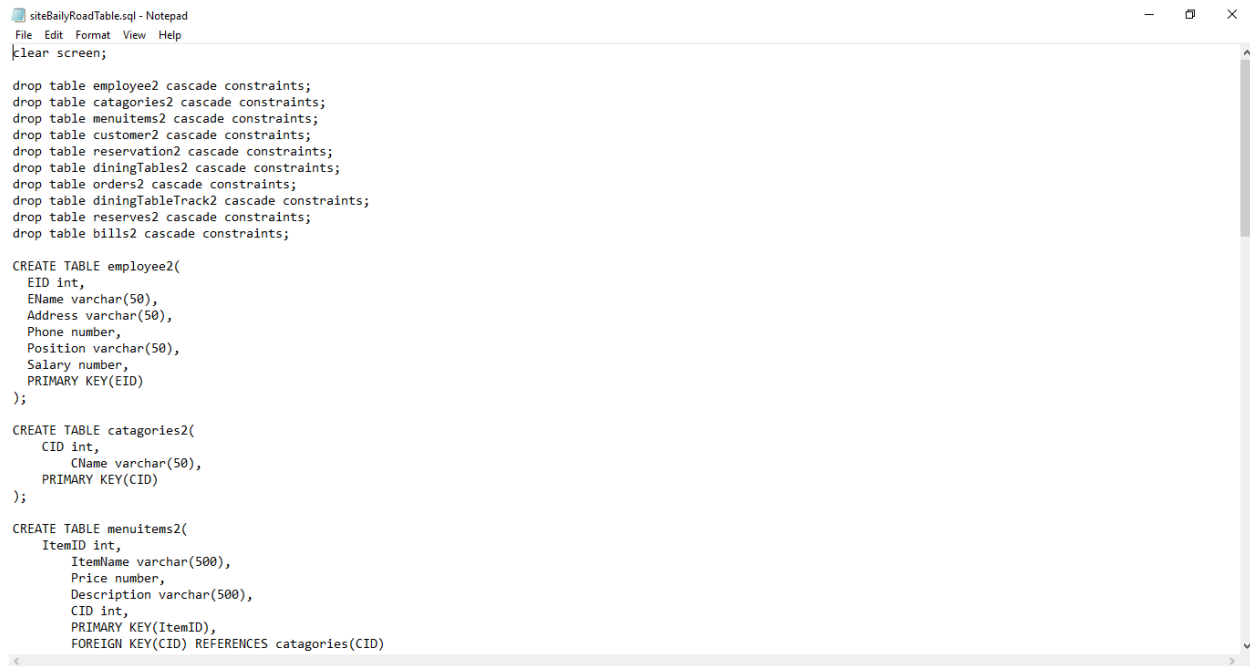


```
pro1.sql - Notepad
File Edit Format View Help
create or replace procedure top_ordering(fromDate in date,toDate in date)
is
    itemName varchar(200);
begin
    SELECT MI.ItemName into itemName FROM menuitems MI INNER JOIN orders O ON
    MI.ItemID = O.ItemID WHERE ODate between fromDate AND toDate AND ROWNUM <= 1
    GROUP BY ItemName ORDER BY sum(O.Quantity) DESC;
    DBMS_OUTPUT.PUT_LINE(itemName);
end;
/
```

Fig:2

3. Fragmentation:

siteBailyRoadTable.sql also called site 1 which has all the fragmented tables from main database.



```
siteBailyRoadTable.sql - Notepad
File Edit Format View Help
|clear screen;

drop table employee2 cascade constraints;
drop table catagories2 cascade constraints;
drop table menuitems2 cascade constraints;
drop table customer2 cascade constraints;
drop table reservation2 cascade constraints;
drop table diningTables2 cascade constraints;
drop table orders2 cascade constraints;
drop table diningTableTrack2 cascade constraints;
drop table reserves2 cascade constraints;
drop table bills2 cascade constraints;

CREATE TABLE employee2(
  EID int,
  EName varchar(50),
  Address varchar(50),
  Phone number,
  Position varchar(50),
  Salary number,
  PRIMARY KEY(EID)
);

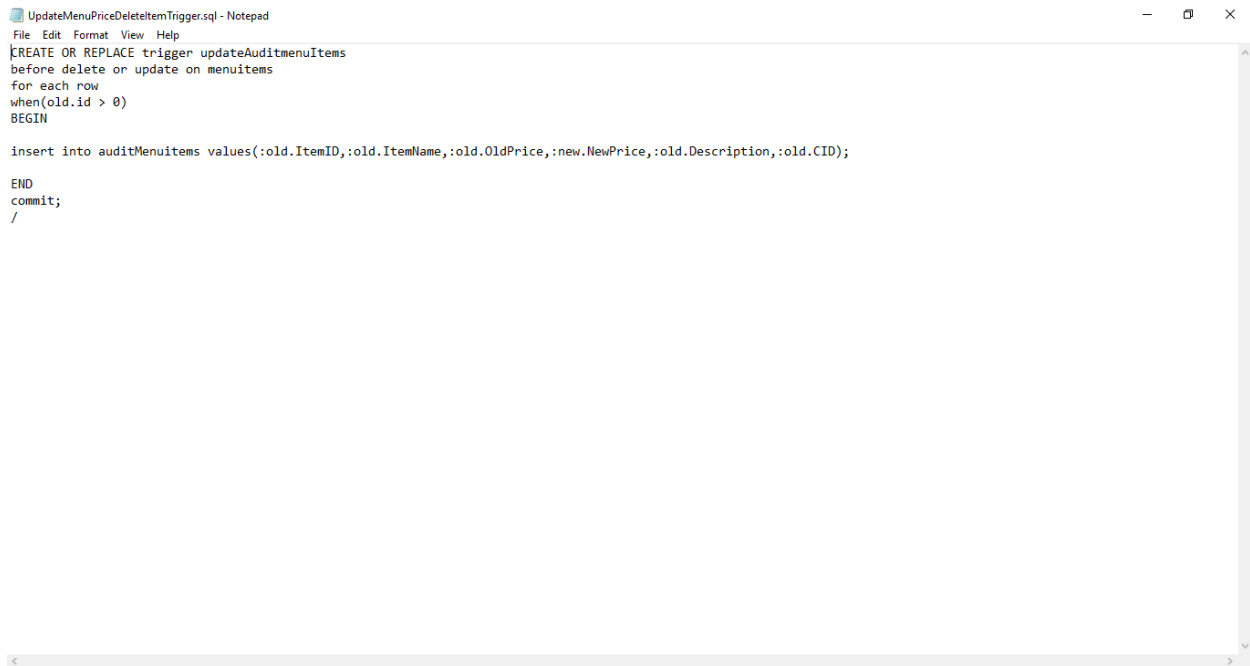
CREATE TABLE catagories2(
  CID int,
  CName varchar(50),
  PRIMARY KEY(CID)
);

CREATE TABLE menuitems2(
  ItemID int,
  ItemName varchar(500),
  Price number,
  Description varchar(500),
  CID int,
  PRIMARY KEY(ItemID),
  FOREIGN KEY(CID) REFERENCES catagories(CID)
```

Fig:3

4. Trigger:

UpdateMenuPriceDeleteItemTrigger.sql which has all the data of previous price and current price and also has previous item and current item.



The image shows a Notepad window titled "UpdateMenuPriceDeleteItemTrigger.sql - Notepad". The window contains the following SQL code:

```
File Edit Format View Help
CREATE OR REPLACE trigger updateAuditmenuItems
before delete or update on menuitems
for each row
when(old.id > 0)
BEGIN

insert into auditMenuItems values(:old.ItemID,:old.ItemName,:old.OldPrice,:new.NewPrice,:old.Description,:old.CID);

END
commit;
/
```

The code is a PL/SQL trigger named "updateAuditmenuItems" that fires before a delete or update operation on the "menuitems" table. It inserts a record into the "auditMenuItems" table with the old and new values of the item. The trigger is created or replaced, and it ends with a commit statement.

Fig:4