**University of Dhaka**

**Department of Computer Science and Engineering**

**Project : *A Database management system of A Departmental Store***

**Course No:**

***CSE – 2201***

**Date**

**29-11-2016**

**Submitted By**

**Redwan Ahmed Rizvee**

**Roll – 09**

**Submitted to**

**Mr. Abu Ahmed Ferdaus**

**a)Introduction:**

Here, basically I wanted to build a database management system on Departmental store. It’s not actually a schema which complete describes a departmental store database, but here I wanted to show, with simplicity how a management system works and the total organizing system of these stores and how they operate. Besides main idea was to implement the knowledge of keys, schema relations, functional dependency preservations and entity relations and to make queries in a word to implement what we learnt.

My departmental store has 7 tables. They are following –

1. Branch
2. Customer
3. Employee
4. Product
5. Dailysale
6. Bank\_Income
7. Delivery\_ service

From these tables there are one independent table Branch. Other tables are related with others through foreign keys or with dependency.

My **Bank** table is totally independent. **Customer** table and **Employee** table are related to this **branch** table because each **customer** must belong to a branch and same case applies for **employee** too. These two tables generally hold the attributes of these two entities of customer and employee, their personal information and that information through which they are related to the store.

**Product** table, is dependent on branch because each product must belong to a branch. **Dailysale** table is dependent on both **customer** table, **branch** table and **product** table because this table actually deals with the sale system which includes a branch, customer and product to complete a deal. Besides this table also helps to compute how much sell are done each day, which products from which branch is bought in one day, the new upgraded **product, customer**  and **branch\_income** relation are totally related with the tuples of this table.

My **Delivery\_service** table actually works with **Customer, Employee** and **Product** relation to indicate a relation which includes who is buying, what is being delivered and which employee is associated with it. Besides, it’s variation or upgrade also effects the **Dailyservice** table, because each delivery service also contributes in **Dailyservice** table and at a whole to **Branch\_income** table.

And last of all my **Bank\_Income** table is related with **Branch** table mainly. And also dependent with **Dailyservice** table. Because the function of this table is to calculate monthly profit of the different branches of this departmental store and save them, for comparing them.

**b) Schemas With Attributes**

In my previous topic I discussed which are my tables and their functions. Now here I will give the attributes of these schemas. They are given following –

1. Branch (**ID**, location, phone\_num, employee\_tot, customer\_tot)
2. Customer (**C\_ID**, C\_NAME, C\_ADDRESS, C\_PHONE,C \_TOTAL\_BOUGHT, C\_SPECIAL\_OFFER, C\_BRANCH\_ID)
3. Employee (**E\_ID**,E\_NAME,E\_BRANCH,E\_PHONE,E\_POST,E\_SALARY,E\_ADDRESS)
4. Product(**P\_ID,** **B\_ID**, P\_NAME,P\_QUANTITY, EXP\_DATE, UNIT\_PRICE, DISCOUNT\_KG)
5. DAILYSALE (**BR\_ID,PR\_ID,CUS\_ID, DAY**,D\_QUANTITY, VAT,TOTAL)
6. BRANCH\_INCOME(**ID, MONTH, YEAR**, TAX,RENT,SALE,OTHER\_COSTS, TOTAL\_PROFIT\_MONTH)
7. DELIVERY\_SERVICE(**SERIAL\_NUM**,CUS\_ID,CUS\_PROD,DIN,BR\_ID,QUANTITY,DELIVERY\_CHARGE,EMPLOYEE\_ID)

**d) Entity Relationship Diagram ( E R Diagram)**

Our Departmental\_store schema is shown on the previous paragraph. Now I will

Discuss about the Entity relationship diagram of my database management system. For clarification and simplicity I will discuss every relationship individually rather merging them.

Here, I will discuss how main relationships were formed in my table or entities from the basic entities and the co relation between entities.

1. Relationship from **Branch** entity and **People\_info** entity to **customer** entity and **employee** entity.



Figure – Customer Entity

Here left to right exists one to many relation. And right side and left side both are not total. It’s a strong entity relation.

To clarify we will explain, our people\_info entity. This entity is not actually in our database, but we can say this as a general entity which every person share and from which we get our derived entity customer and employee associated with branch. Moreover this extra entity is bought to explain how from general entity people\_info we can divide into two different entity customer and employee.

Now, here we see , employee entity.



Figure – Employee entity.

Here ,exists, one to many relation from left to right side. And left and right side both are not total. Because, it can happen a new branch is established which didn’t have recruited new employees till then, besides all people need not to become an employee of this store.

1. Relationship from **branch\_info** and element to **product** entity.



Figure – Product

Here,both sides are many to many relation and each side are no total relation. It’s also a strong entity relation.

Same explanation can be given for element entity, which is not directly associated with product entity rather exclaimed as a general element entity which is shared by all the elements of the world from some of them are associated with our branch which we have given name as product entity.

1. Relationship between derived entity **customer** and derived entity **product** to new entity relation **DAILYSALE**.



**Figure – DAILYSALE**

It’s actually a many to many relation but there is some extra attributes which alongside form the relation **dailysale**. It’s left side is not total but right side is total, where of course a customer buy’s at least a product.

1. Relationship between derived entity **product**, derived entity **employee** and customer by making a relationship entity **deliveryservice** along with an extra primary key attribute **serial\_num.**



**Figure – Delivery\_Service**

Here three entities form the final entity Delivery\_service. And so it’s little bit complicated to explain the whole relation here. Here from employee there exists one to many relations and it’s not total, from customer it’s also an one to many relation because a single customer can buy or order many things and it’s not total and from the product side it’s also one to many relation and it’s not also total.

Here in the final relation, serial\_num is the primary key which is totally an extra attribute and doesn’t belong to any other forming entities. It should have been better if (C\_ID,E\_ID,P\_ID,B\_ID,Data,Time) could been a primary key. But to maintain simplicity it’s not performed here.

1. Relationship **Branch\_income**



Figure – Branch\_income

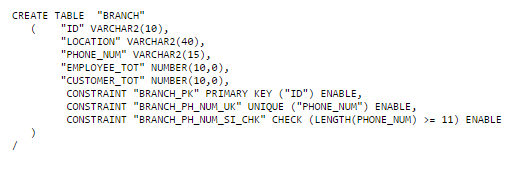
A relationship formed from dailysale, branch and branch\_extra\_info and to a final relationship entity branch\_income. Here all attributes are total and all are one to one relation.

But if we look carefully, it’s actually just a derived relation which was not very necessary. This table is prepared only to make an easy access to the profits of the branches at different times, at to save them in the same table, But each of them can be showed this table can be derived alone from dailysale table if we not erase the data of this table after each day.

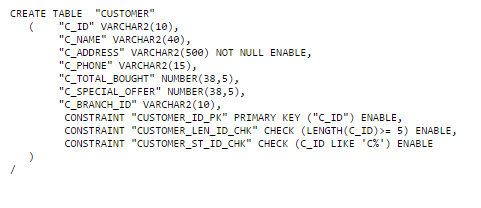
My idea was, Dailysale table is made only to calculate and save daily services along with help of **delivery\_service** and the final result will be saved in our **branch\_income** entity.

**e) Snapshots of the SQL DDL of Schemas and Tables**

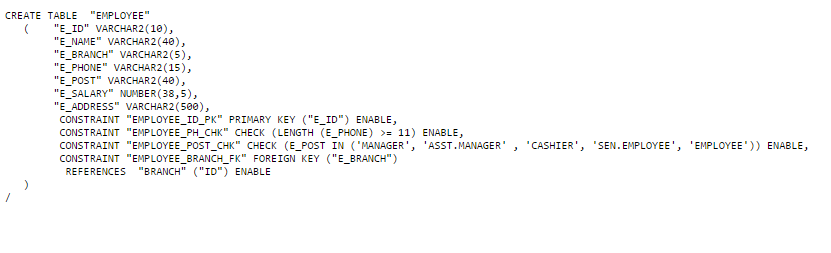
#**Branch Relation:**

****

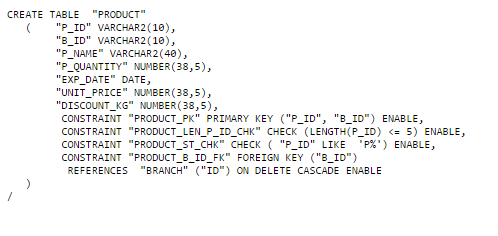
**#Customer Relation:**

****

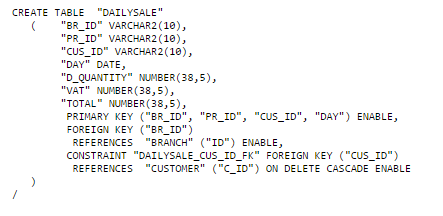
#**Employee Relation:**

****

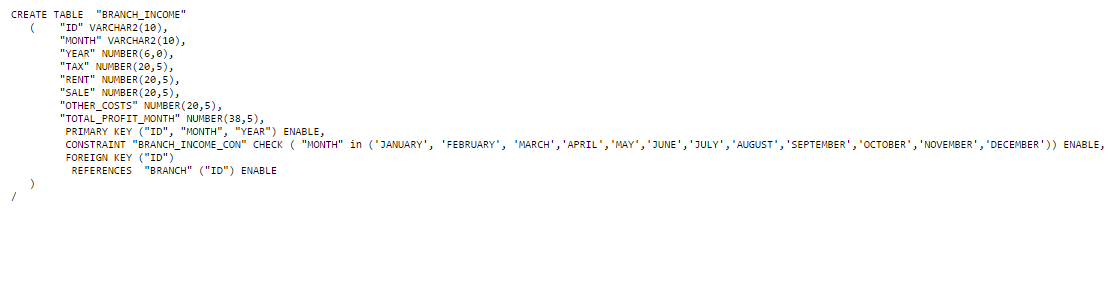
**#Product Relation:**

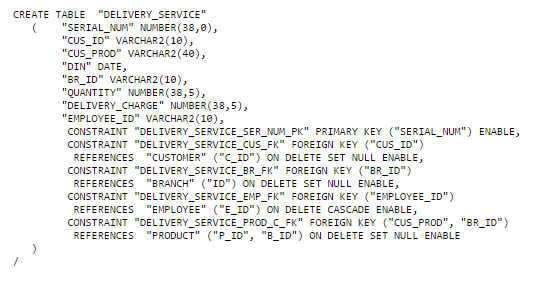
****

**#Dailysale Relation:**

****

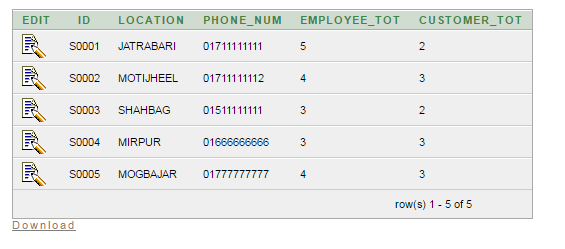
**#Branch\_income Relation:**

****

**#Delivery\_service relation:**

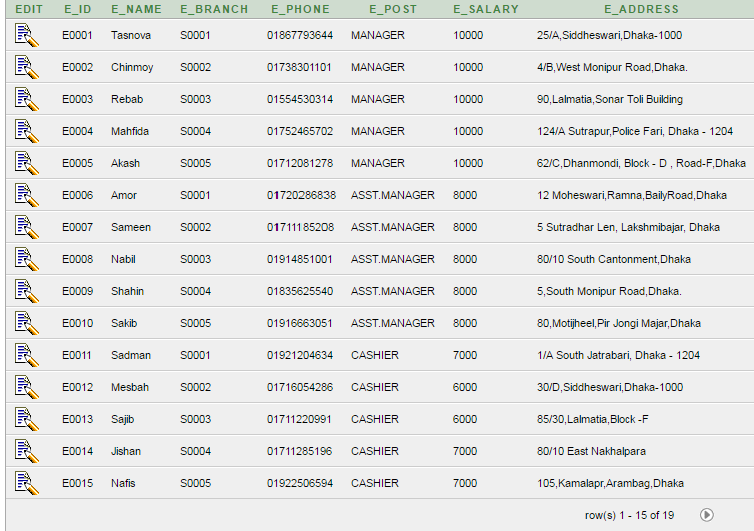
**e) Snapshots of the Instances**

**#Table Branch**

****

**#Table Customer**

**#Table Employee**

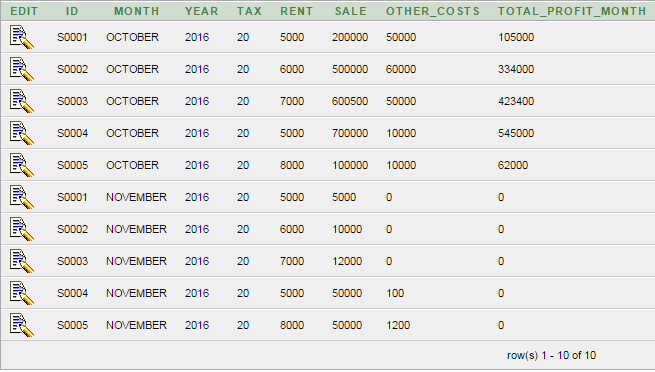
****

**#Table Product**

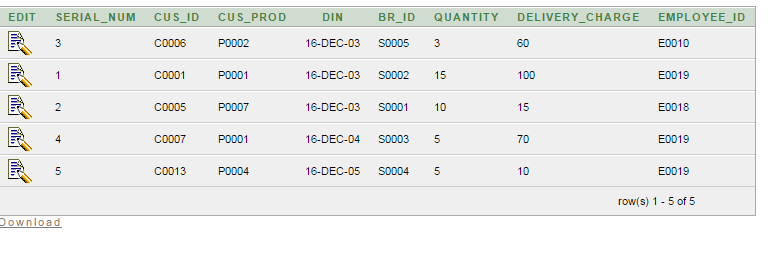
****

****

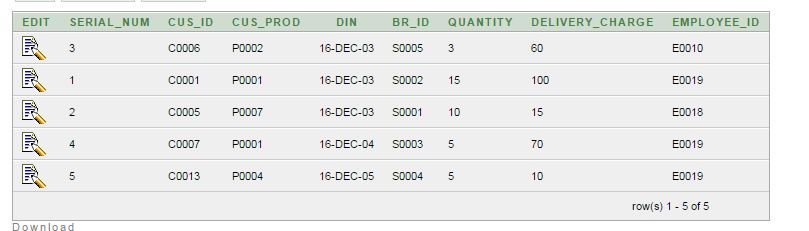
**#Table Branch\_Income**

****

**#Table Delivery\_service**

****

**#Table Dailysale**

****

**g) Query Expression**

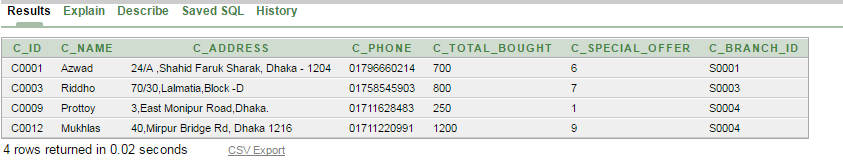
**Query – string operation**

Show all the consumers, who use grameen phone or prefix of phone number ‘017’

**SQL: select \***

**from CUSTOMER**

**where C\_PHONE like '017%'**



**Figure – String Operation used**

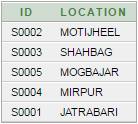
**Query – Cross Product**

Show the branches which have at least one customer using cross product.

SQL: **select distinct(branch.id),branch.location**

**from customer,branch**

**where customer.c\_branch\_id = branch.id**

****

**Figure – Cross Product used**

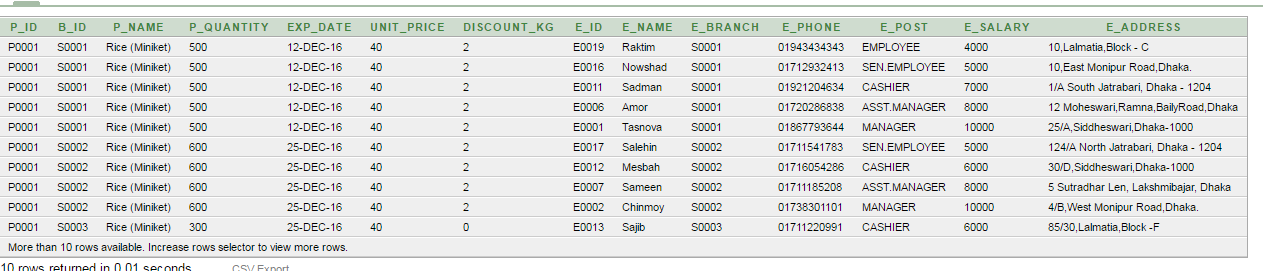
**Query – Cross Product**

Show the employee who can sell a product along with the product , because employee can sell only those products which are found in his branch and that product is not of ‘P0002’ id

SQL: **select \***

**from product,employee**

**where employee.e\_branch = product.b\_id and product.p\_id <> ‘P0002’**

****

**Figure – Cross Product used**

**Query – Set Operation**

Show the information of the employees who are not associated with some delivery service

SQL:

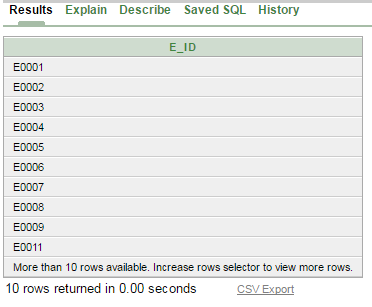
**select Employee.E\_ID**

**from Employee**

**minus**

**select DELIVERY\_SERVICE.EMPLOYEE\_ID**

**from DELIVERY\_SERVICE**



**Figure – Set Operation used**

**Query – Nested Subquery**

Show the information of the employees who are not associated with some delivery service (Same query but using nested sub query not set operation)

SQL:

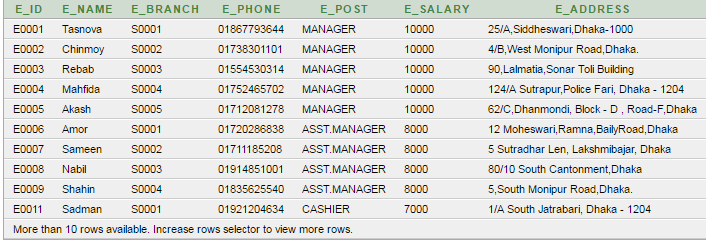
**select \***

**from employee**

**where employee.E\_id not in (**

**select distinct(Employee\_id)**

**from delivery\_service)**



**Figure – Nested Subquery used**

**Query – With Clause, nested subquery, aggregate function**

Show the information of the customer who have purchased maximum valued elements.

SQL:

**select \***

**from customer**

**where customer.c\_id in (**

**with max\_total as (**

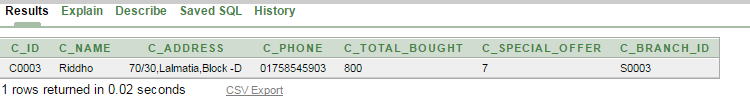
**select max(total) as value**

**from dailysale)**

**select CUS\_ID**

**from dailysale,max\_total**

**where dailysale.total = max\_total.value)**



**Figure – Nested, with, aggregate function used**

**Query – Order by**

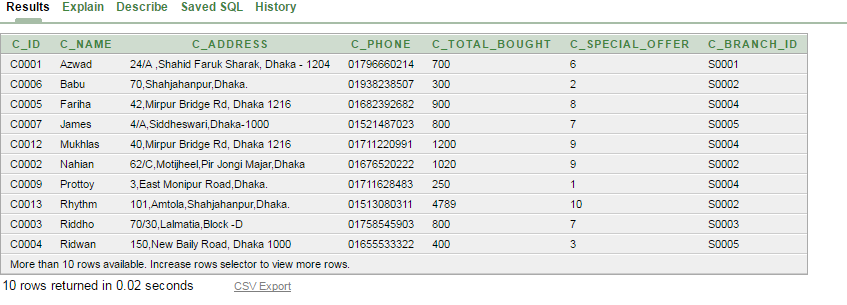
Show all the customers in ascending order by their name

SQL:

**select \***

**from customer**

**order by c\_name asc**



**Figure** – **Order by used**

**Query – Group by, having, aggregate function**

Find the maximum total profit in month october from the existing branches

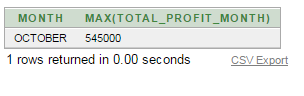
SQL:

**select month,max(TOTAL\_PROFIT\_MONTH)**

**from branch\_income**

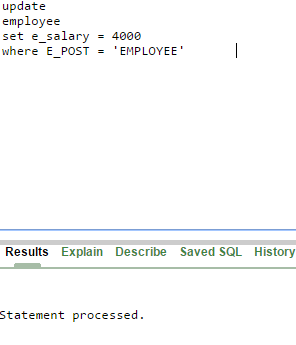
**group by month**

**having month = 'OCTOBER'**



**Figure – Group by, having, aggregate function used**

**Query – Update (modification)**



**Figure – Modification (update)**

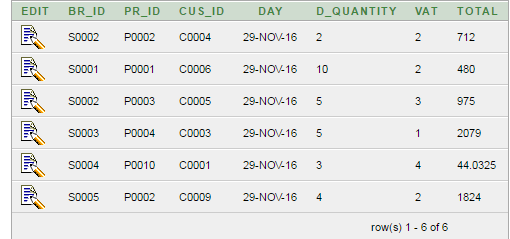
**Query – Insertion(Modification)**

If customer ‘C0004’ buys ‘P0002’ from ‘S0002’ on ’29-NOV-2016’

Then update the dailysale table .

SQL:

**insert into dailysale values ('S0002', 'P0002','C0004','29-NOV-2016',2,2,712)**

****

**Figure – Insertion**

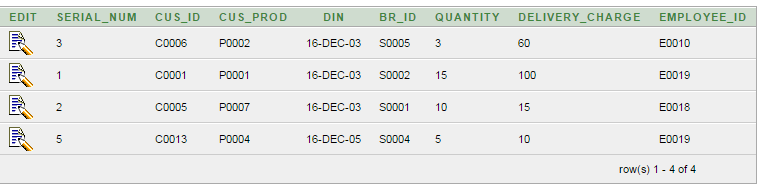
**Query - Deletion**

Delete the 4th order from the delivery service

SQL:

**delete from delivery\_service**

**where serial\_num = 4**

****

**Figure - Deletion**

**Query – With, nested sub query and any**

Find the product which is maximum in quantity in any branch.

SQL:

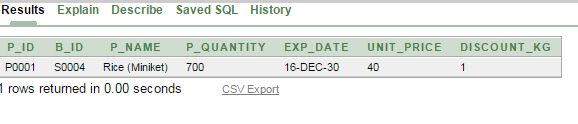
**select \***

**from product**

**where p\_quantity >= all (**

**select P\_QUANTITY**

**from product)**

**Figure - With, nested sub query and any**

**Query – With, nested subquery and some**

Show all the products which are less than in quantity compairing to any other products.

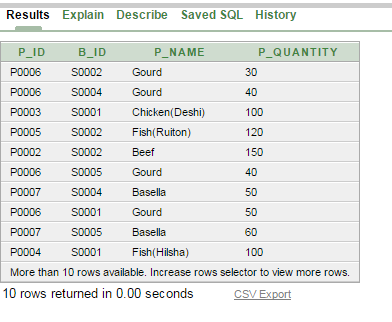
**select distinct(P\_ID),B\_ID, P\_NAME, P\_QUANTITY**

**from product**

**where p\_quantity > some (**

**select P\_QUANTITY**

**from product)**



**Figure - With, nested subquery and some**

**Query – With, nested subquery and exists**

Show the employees who hava salary of at least 4000 and not associated with delivery\_service

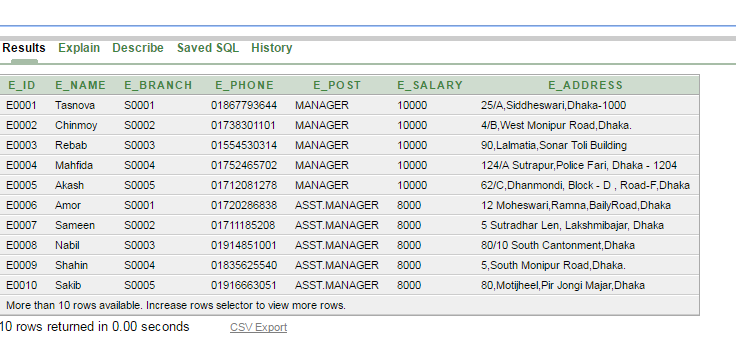
**select \***

**from employee**

**where e\_salary>= 4000 and exists(**

**select employee\_id**

**from delivery\_service**

**)** 

**Figure** - **With, nested subquery and exists**

**Query – Natural join,with**

Show the products which are sold and their remaining quantity.

SQL:

**with delivery\_service\_new as(**

**select br\_id as b\_id,cus\_prod as prod ,employee\_id as e\_id,din as d, quantity as bought\_quantity**

**from delivery\_service), product\_new as (**

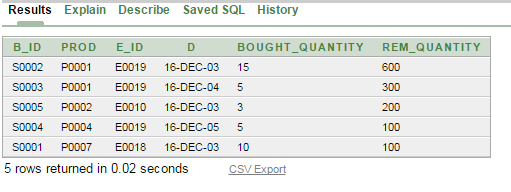
**select p\_id as prod, b\_id as b\_id, p\_quantity as rem\_quantity**

**from product**

**)**

**select \***

**from delivery\_service\_new natural join product\_new**



**Figure - Natural join,with**

**Query – left outer join (using)**

**Same work using left outer join**

SQL:

**with delivery\_service\_new as(**

**select br\_id as b\_id,cus\_prod as prod ,employee\_id as e\_id,din as d, quantity as bought\_quantity**

**from delivery\_service), product\_new as (**

**select p\_id as prod, b\_id as b\_id, p\_quantity as rem\_quantity**

**from product**

**)**

**select \***

**from delivery\_service\_new left outer join product\_new using (prod,b\_id)**

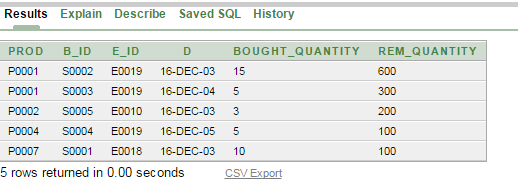


Figure - **Natural Join, left outer join (using)**

**Query – Left outer join , and on**

**Same work been done using one**

SQL:

**with delivery\_service\_new as(**

**select br\_id as b\_id,cus\_prod as prod ,employee\_id as e\_id,din as d, quantity as bought\_quantity**

**from delivery\_service), product\_new as (**

**select p\_id as prod, b\_id as b\_id, p\_quantity as rem\_quantity**

**from product**

**)**

**select \***

**from delivery\_service\_new left outer join product\_new on (prod,b\_id)**

**h) Functional Dependency,BCNF,Canonical Cover:**

**#Represention of functional dependencies -**

1. Relation – Branch (ID, location,phone\_num,employee\_tot,customer\_tot)

We assume,

ID = A

location =B

Phone\_num = C

employee\_tot = D

customer\_tot = E

So, our functional dependency(FD) stands,

A->B C D E

1. Relation – Customer (C\_ID,C\_NAME,C\_ADDRESS,C\_PHONE,C\_TOTAL\_BOUGHT,C\_SPECIAL\_OFFER,C\_BRANCH\_ID)

We assume,

C\_ID = F

C\_NAME = G

C\_ADDRESS = H

C\_PHONE = I

C\_TOTAL\_BOUGHT = J

C\_SPECIAL\_OFFER = K

C\_BRANCH\_ID = Branch.id = A

So FD stands, F->G H I J K A

1. Relation – Employee (E\_ID,E\_NAME,E\_BRANCH,E\_PHONE,E\_POST,E\_SALARY,E\_ADDRESS)

We assume,

E\_ID = L

E\_NAME = M

E\_BRANCH = Branch.id = A

E\_PHONE = N

E\_POST = O

E\_SALARY = P

E\_ADDRESS = Q

So our FD stands, L -> M A N O P Q

1. Relation – Product(P\_ID, B\_ID, P\_NAME,P\_QUANTITY, EXP\_DATE, UNIT\_PRICE, DISCOUNT\_KG)

We assume

P\_ID = R

B\_ID = Branch.id = A

P\_NAME = S

P\_QUANTITY = T

EXP\_DATE = U

UNIT\_PRICE = V

DISCOUNT\_KG = W

So, FD stands, R A -> S T U V W

1. Relation – DAILYSALE (BR\_ID,PR\_ID,CUS\_ID, DAY,D\_QUANTITY, VAT,TOTAL)

We assume

BR\_ID = Branch.id = A

PR\_ID = Product.P\_ID = R

CUS\_ID = Customer.C\_ID = F

DAY = X

D\_QUANTITY = Y

VAT = Z

TOTAL = alpha

So, our FD stands , A R F X -> Y Z alpha

1. Relation – BRANCH\_INCOME(ID, MONTH, YEAR, TAX,RENT,SALE,OTHER\_COSTS, TOTAL\_PROFIT\_MONTH)

We assume.

ID = Branch.id = A

Month = beta

Year = gamma

Tax = delta

Rent = epsilon

SALE = zeta

OTHER\_COSTS = eta

TOTAL\_PROFIT\_MONTH = theta

So our FD stands, A beta gamma -> delta epsilon zeta eta theta

1. Relation DELIVERY\_SERVICE(SERIAL\_NUM,CUS\_ID,CUS\_PROD,DIN,BR\_ID,QUANTITY,DELIVERY\_CHARGE,EMPLOYEE\_ID)

We assume

SERIAL\_NUM = phi

CUS\_ID = F

CUS\_PROD = R

DIN = X

BR\_ID = A

QUANTITY = Y

DELIVERY\_CHARGE = psi

EMPLOYEE\_ID = L

So our FD, stands, phi -> F R X A Y psi L

So our main Functional Dependencies for our schema Departmental\_Store are,

1. A->B C D E
2. F->G H I J K A
3. L -> M A N O P Q
4. R A -> S T U V W
5. A R F X -> Y Z alpha
6. A beta gamma -> delta epsilon zeta eta theta
7. phi -> F R X A Y psi L

#**Proof of BCNF**

these are our main functional dependencies which all support BCNF, because these dependencies are all candidate key’s for that relation. As two condtion satisfies BCNF.

i)either the relations are trivial

ii) or they are super key

As oure every functional dependency form relation with super keys they can be said they are in BCNF form.

#**Proof of 3NF Format**

Functional dependencies which support BCNF definitely support 3NF so. Our functional dependencies support 3NF.

#**Canonical Cover Finding:**

If we look at the functional dependencies carefully we will notice this –

From,

R A -> S T U V W…………………….(4)

A R F X -> Y Z alpha…………………(5)

From (4) A R F X -> F X S T U V W……………….(6)

Union (5) and (6),

A R F X -> F X S T U V W Y Z alpha………………(7)

Removing F X from both sides,

A R -> S T U V W Y Z alpha………………………..(8)

Which means from (BRANCH.ID,PRODUCT.P\_ID)

we can find uniquely

(Product.P\_NAME, Product. P\_QUANTITY, Product. EXP\_DATE, Product. UNIT\_PRICE, Product. DISCOUNT\_KG, DAILYSALE.D\_QUANTITY,DAILYSALE.VAT,DAILYSALE.TOTAL)

Which will be a very huge table but functional dependency will reduce.

Again, if we look carefully to this relation,

Phi -> F R X A Y psi……………………….(9)

This relation gives almost all the attributes like this

Phi -> F R X A Y

* phi -> F X Y (A R)
* phi -> F X Y S T U V W Y Z alpha

But this is generally wrong because phi is a special attribute which is serial number in DELIVERY\_SERVICE table, if we use this attribute and reduce our functional dependency, most of the main relation will diminish. So this will not be done.

Besides, we already explained, if we use serial\_num from this as equivalent to,

Phi = A R F X

Serial\_num = Branch.id, Product.P\_ID, Customer.C\_ID,DAILYSALE.DAY

So,

A R F X -> F R X A Y psi L

Removing trivial dependencies,

A R F X -> Y psi L …………………………………(10)

So, we already say,

ARFX-> F X S T U V W Y Z alpha

Now union,

A R F X -> F X S T U V W Y Z alpha psi L

Removing trivial dependencies,

A R -> S T U V W Y Z alpha psi L.

So , the canonical cover of our functional dependencies are,

A -> B C D E

F -> G H I J K

L -> M A N O P Q

A R -> S T U V W Y Z alpha psi L

A beta gamma -> delta epsilon zeta eta theta

So, the new established relations are

Relation1.

(**Branch.ID**, Branch.location, Branch.phone\_num, Branch.employee\_tot, Branch.customer\_tot)

Relation2.

(**Customer.C\_ID**, Customer.C\_NAME, Customer.C\_ADDRESS, Customer.C\_PHONE, Customer.C\_TOTAL\_BOUGHT, Customer.C\_SPECIAL\_OFFER, Customer.C\_BRANCH\_ID)

Relation3.

(**Employee.E\_ID**, Employee.E\_NAME, Employee.E\_BRANCH, Employee.E\_PHONE, Employee.E\_POST, Employee.E\_SALARY Employee.,E\_ADDRESS)

Relation4.

(**Branch.id, Product.P\_ID**, Product.P\_QUANTITY, Product.EXP\_DATE, Product.UNIT\_PRICE,Product. DISCOUNT\_KG, DAILYSALE.D\_QUANTITY, DAILYSALE.VAT, DAILYSALE.TOTAL, DELIVERY\_SERVICE.DELIVERY\_CHARGE, DELIVERY\_SERVICE.EMPLOYEE\_ID)

Relation5.

(**BRANCH\_INCOME.ID, BRANCH\_INCOME.MONTH, BRANCH\_INCOME.YEAR**, BRANCH\_INCOME.TAX, BRANCH\_INCOME.RENT, BRANCH\_INCOME.SALE, BRANCH\_INCOME.OTHER\_COSTS, BRANCH\_INCOME.TOTAL\_PROFIT\_MONTH).