

CSCE 5350 004

FUNDAMENTALS OF DATABASE SYSTEMS

PROJECT - PART 4

GROUP-8

Name	ID
Nitin Chakravarthy Chilukuri	11601099
Surya Vamsi Chintapalli	11645442
Prathyusharani Dumpala	11656342
Yamini Gollamudi	11642723
Lohitha Sai Bonthu	11611601

ATTRIBUTES:

Stores: Store Id, Address, Manager, Assigned Pharmacist, Assigned Doctor, Region Code

Warehouses: Warehouse Id, Address, Warehouse Manager, Current Stock, Capacity, Region Code

Region: Region Name, Region Code, Region Manager

Employees: EID, Name, SSN, Age, Gender, Address, Ph NO, Wage, Type, Location, Bank Account Number

Payroll: EID, Date, Hours Worked

Patients: PID, Name, Age, Gender, Ph NO, Address, SSN

Insurance: INM, Name of Insurance, PID, Amount, Date Claimed, Status

Prescription: PrescriptionID, DoctorID, PatientID, Date_Prescribed

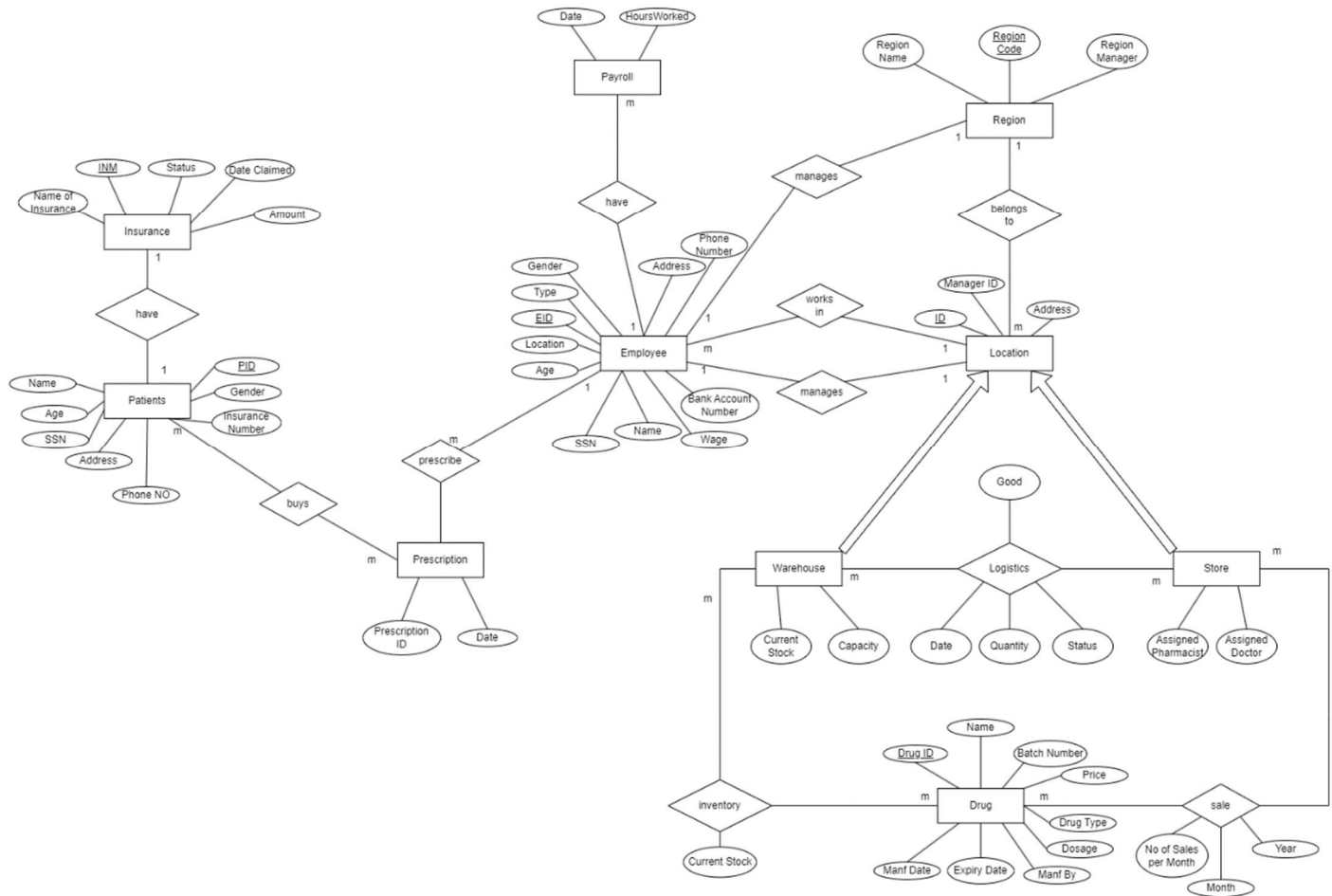
Drug: Drug ID, Name, Price, Drug Type, Dosage, Manf By, Manf Date, Batch NO, Expiry Date

Inventory: Drug ID, Building ID, Current Stock

Logistics: Good, Date, Warehouse ID, Store ID, Quantity, Status

Sales: No of sales per month, Month, Year, Drug ID, Store ID

ER DIAGRAM OF ENTITIES:



CREATION & BODY OF PACKAGE:

create or replace package fdb_proj as

 procedure empsal_update(eid in employee.EID%type, multiplier in number);

 procedure total_hours(y in number, m in number, eid in payroll.EID%type);

 function get_insamt(d in varchar2) return insurance.amount%type;

 function get_swl(sid in logistics.storeid%type, wid in logistics.warehouseid%type, y in number, m in number) return logistics.quantity%type;

end fdb_proj;

```
SQL Plus
SQL> create or replace package fdb_proj as
2   procedure empsal_update(eid in employee.EID%type, multiplier in number);
3   procedure total_hours(y in number, m in number, eid in payroll.EID%type);
4   function get_insamt(d in varchar2) return insurance.amount%type;
5   function get_swl(sid in logistics.storeid%type, wid in logistics.warehouseid%type, y in number, m in number) return logistics.quantity%type;
6 end fdb_proj;
7 /

Package created.

SQL> create or replace package body fdb_proj as
2   procedure empsal_update(eid in employee.EID%type, multiplier in number)
3   as
4   begin
5       update employee
6       set employee.wage = employee.wage*multiplier
7       where employee.EID = eid;
8       dbms_output.put_line('Wage of Employee '|| eid ||' changed');
9   end empsal_update;
10
11  procedure total_hours(y in number, m in number, eid in payroll.EID%type)
12  as
13  s payroll.HOURS_WORKED%type;
14  begin
15      select sum(HOURS_WORKED) into s
16      from payroll
17      where EXTRACT(YEAR FROM WORK_DATE)=y AND EXTRACT(MONTH FROM WORK_DATE)=m AND payroll.EID = eid;
18      dbms_output.put_line('Number of Hours worked by Employee '|| eid ||' is: '|| s);
19  end total_hours;
20
21  function get_insamt(d in varchar2)
22  return insurance.amount%type
23  is
24  begin
25      s insurance.amount%type;
26      select SUM(amount) into s
27      from insurance
28      where dateclaimed = TO_DATE(d,'yyyy/mm/dd');
29      return s;
30  end get_insamt;
31
32  function get_swl(sid in logistics.storeid%type, wid in logistics.warehouseid%type, y in number, m in number)
33  return logistics.quantity%type
34  is
35  s logistics.quantity%type;
36  begin
37      select SUM(quantity) into s
38      from logistics l
39      where l.warehouseid = wid and l.storeid = sid and EXTRACT(YEAR FROM l.DATEOFORDER)=y AND EXTRACT(MONTH FROM l.DATEOFORDER)=m;
40      return s;
41  end get_swl;
42 end fdb_proj;
43 /

Package body created.
```

```
SQL Plus
8       dbms_output.put_line('Wage of Employee '|| eid ||' changed');
9   end empsal_update;
10
11  procedure total_hours(y in number, m in number, eid in payroll.EID%type)
12  as
13  s payroll.HOURS_WORKED%type;
14  begin
15      select sum(HOURS_WORKED) into s
16      from payroll
17      where EXTRACT(YEAR FROM WORK_DATE)=y AND EXTRACT(MONTH FROM WORK_DATE)=m AND payroll.EID = eid;
18      dbms_output.put_line('Number of Hours worked by Employee '|| eid ||' is: '|| s);
19  end total_hours;
20
21  function get_insamt(d in varchar2)
22  return insurance.amount%type
23  is
24  begin
25      s insurance.amount%type;
26      select SUM(amount) into s
27      from insurance
28      where dateclaimed = TO_DATE(d,'yyyy/mm/dd');
29      return s;
30  end get_insamt;
31
32  function get_swl(sid in logistics.storeid%type, wid in logistics.warehouseid%type, y in number, m in number)
33  return logistics.quantity%type
34  is
35  s logistics.quantity%type;
36  begin
37      select SUM(quantity) into s
38      from logistics l
39      where l.warehouseid = wid and l.storeid = sid and EXTRACT(YEAR FROM l.DATEOFORDER)=y AND EXTRACT(MONTH FROM l.DATEOFORDER)=m;
40      return s;
41  end get_swl;
42 end fdb_proj;
43 /

Package body created.
```

This Package contains 2 Procedures and 2 Functions.

These are:

- **empsal_update**: This is a procedure used to update the wage of an employee. It takes EID and multiplier as inputs. $\text{New_Wage} = \text{Old_Wage} * \text{Multiplier}$.

create or replace procedure empsal_update(eid in employee.EID%type, multiplier in number)

as

begin

 update employee

 set employee.wage = employee.wage*multiplier

 where employee.EID = eid;

 dbms_output.put_line('Wage of Employee '|| eid ||' changed');

end;

- **total_hours**: This is a procedure to get the total number of hours an employee worked in a particular month in a particular year. This takes year, month and EID as input.

create or replace procedure total_hours(y in number, m in number, eid in payroll.EID%type)

as

s payroll.HOURS_WORKED%type;

begin

 select sum(HOURS_WORKED) into s

 from payroll

 where EXTRACT(YEAR FROM WORK_DATE)=y AND EXTRACT(MONTH FROM WORK_DATE)=m AND payroll.EID = eid;

 dbms_output.put_line('Number of Hours worked by Employee '|| eid ||' is: '|| s);

end;

- **get_insamt**: This is a function to get the total amount of insurance claimed on a particular date. It takes a date as input.

create or replace function get_insamt(d in varchar2)

return insurance.amount%type

is

s insurance.amount%type;

```

begin

    select SUM(amount) into s

    from insurance

    where dateclaimed = TO_DATE(d,'yyyy/mm/dd');

    return s;

end;

```

- **get_swl**: This is a function that gives us how much stock is moved from a warehouse to a store in a month. It takes StoreId, WarehouseID, month and year as inputs.

create or replace function get_swl(sid in logistics.storeid%type, wid in logistics.warehouseid%type, y in number, m in number)

return logistics.quantity%type

is

s logistics.quantity%type;

begin

select SUM(quantity) into s

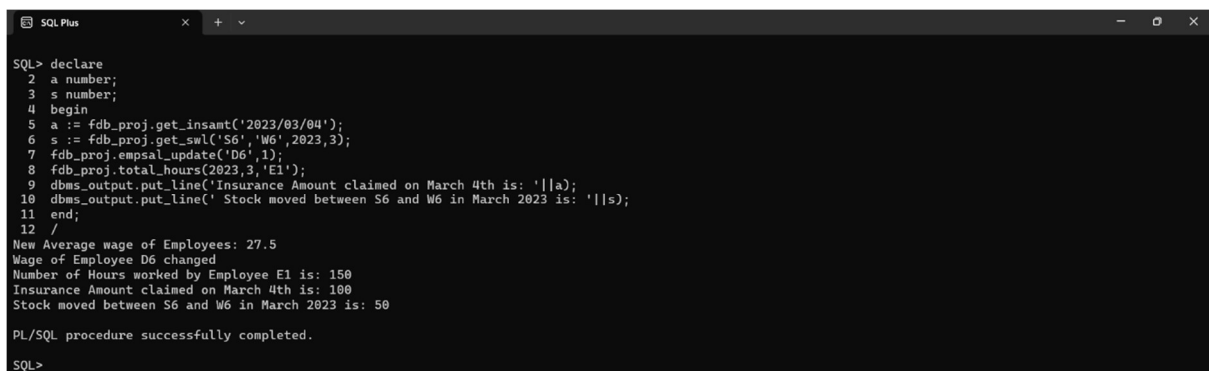
from logistics l

where l.warehouseid = wid and l.storeid = sid and EXTRACT(YEAR FROM l.DATEOFORDER)=y AND EXTRACT(MONTH FROM l.DATEOFORDER)=m;

return s;

end;

When executed, the above four results in the following:



```

SQL> declare
  2 a number;
  3 s number;
  4 begin
  5 a := fdb_proj.get_insamt('2023/03/04');
  6 s := fdb_proj.get_swl('S6','W6',2023,3);
  7 fdb_proj.empsal_update('D6',1);
  8 fdb_proj.total_hours(2023,3,'E1');
  9 dbms_output.put_line('Insurance Amount claimed on March 4th is: '||a);
 10 dbms_output.put_line(' Stock moved between S6 and W6 in March 2023 is: '||s);
 11 end;
 12 /
New Average wage of Employees: 27.5
Wage of Employee D6 changed
Number of Hours worked by Employee E1 is: 150
Insurance Amount claimed on March 4th is: 100
Stock moved between S6 and W6 in March 2023 is: 50

PL/SQL procedure successfully completed.

SQL>

```

We can observe a line 'New Average wage of Employees: 27.5' in the output. This is the result of a trigger which we will see later.

PROCEDURES & FUNCTIONS:

1) avg_hw: This is a procedure that displays the average working hours and average wage per hour for an employee in a month.

create or replace procedure avg_hw

as

c1 number;

s1 number;

avg_h number;

c2 number;

s2 number;

avg_w number;

begin

select SUM(count(DISTINCT EID)) into c1

from payroll

group by payroll.EID;

select SUM(HOURS_WORKED) into s1 from payroll;

avg_h := s1/c1;

select SUM(count(DISTINCT EID)) into c2

from employee

group by employee.EID;

select SUM(wage) into s2 from employee;

avg_w := s2/c2;

dbms_output.put_line('Average Number of Hours worked by an Employee is: '||
avg_h);

dbms_output.put_line('Average Wage of an Employee is: '|| avg_w);

end;

```

create or replace procedure avg_hw
2 as
3 c1 number;
4 s1 number;
5 avg_h number;
6 c2 number;
7 s2 number;
8 avg_w number;
9 begin
10 select SUM(count(DISTINCT EID)) into c1
11 from payroll
12 group by payroll.EID;
13 select SUM(HOURS_WORKED) into s1 from payroll;
14 avg_h := s1/c1;
15
16 select SUM(count(DISTINCT EID)) into c2
17 from employee
18 group by employee.EID;
19 select SUM(wage) into s2 from employee;
20 avg_w := s2/c2;
21
22 dbms_output.put_line('Average Number of Hours worked by an Employee is: '|| avg_h);
23 dbms_output.put_line('Average Wage of an Employee is: '|| avg_w);
24 end;
25 /
Procedure created.

```

```

SQL> execute avg_hw;
Average Number of Hours worked by an Employee is: 37.4
Average Wage of an Employee is: 27.5

PL/SQL procedure successfully completed.

SQL>

```

2) age_patients: This procedure tells us which age group contains the greatest number of patients.

create or replace procedure age_patients

as

c0 number;

c1 number;

c2 number;

c3 number;

c4 number;

c5 number;

c6 number;

m number;

s varchar2(20);

begin

select count(pid) into c0 from patients where patients.age>0 and patients.age<10;

select count(pid) into c1 from patients where patients.age>=10 and patients.age<20;

select count(pid) into c2 from patients where patients.age>=20 and patients.age<30;

select count(pid) into c3 from patients where patients.age>=30 and patients.age<40;

```
select count(pid) into c4 from patients where patients.age>=40 and patients.age<50;  
select count(pid) into c5 from patients where patients.age>=50 and patients.age<60;  
select count(pid) into c6 from patients where patients.age>=60 and patients.age<70;
```

```
if c0 > c1 then  
    m := c0;  
    s := 'Between 0 & 10';  
else  
    m := c1;  
    s := 'Between 10 & 20';  
end if;  
if m < c2 then  
    m := c2;  
    s := 'Between 20 & 30';  
end if;  
if m < c3 then  
    m := c3;  
    s := 'Between 30 & 40';  
end if;  
if m < c4 then  
    m := c4;  
    s := 'Between 40 & 50';  
end if;  
if m < c5 then  
    m := c5;  
    s := 'Between 50 & 60';  
end if;  
if m < c6 then  
    m := c6;  
    s := 'Between 60 & 70';
```



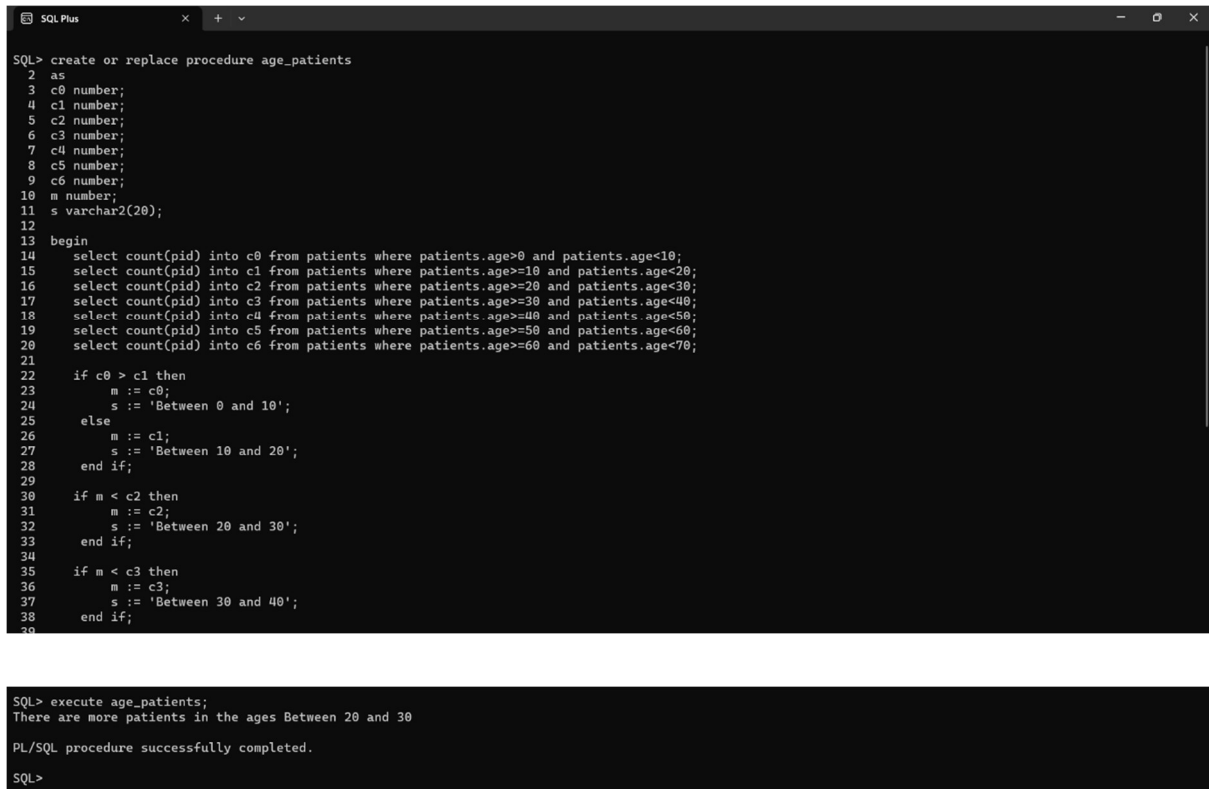
```

        end if;

        dbms_output.put_line('There are more patients in the ages ' || s);

end;

```



```

SQL> create or replace procedure age_patients
2  as
3  c0 number;
4  c1 number;
5  c2 number;
6  c3 number;
7  c4 number;
8  c5 number;
9  c6 number;
10 m number;
11 s varchar2(20);
12
13 begin
14     select count(pid) into c0 from patients where patients.age>0 and patients.age<10;
15     select count(pid) into c1 from patients where patients.age>=10 and patients.age<20;
16     select count(pid) into c2 from patients where patients.age>=20 and patients.age<30;
17     select count(pid) into c3 from patients where patients.age>=30 and patients.age<40;
18     select count(pid) into c4 from patients where patients.age>=40 and patients.age<50;
19     select count(pid) into c5 from patients where patients.age>=50 and patients.age<60;
20     select count(pid) into c6 from patients where patients.age>=60 and patients.age<70;
21
22     if c0 > c1 then
23         m := c0;
24         s := 'Between 0 and 10';
25     else
26         m := c1;
27         s := 'Between 10 and 20';
28     end if;
29
30     if m < c2 then
31         m := c2;
32         s := 'Between 20 and 30';
33     end if;
34
35     if m < c3 then
36         m := c3;
37         s := 'Between 30 and 40';
38     end if;
39
SQL> execute age_patients;
There are more patients in the ages Between 20 and 30

PL/SQL procedure successfully completed.

SQL>

```

3) logistics_status: This is a procedure that tells us how many logistics orders are in pending state and how many orders are In Transit.

```
create or replace procedure logistics_status
```

```
as
```

```
p number;
```

```
t number;
```

```
begin
```

```
    select count(status) into p
```

```
    from logistics
```

```
    where status = 'Pending';
```

```
    select count(status) into t
```

```
    from logistics
```

```
    where status = 'In Transit';
```

```
dbms_output.put_line('Total Number of Orders in Pending Status is: '|| p);
```

```
dbms_output.put_line('Total Number of Orders in Transit are: '|| t);
```

```
end;
```

```
SQL> create or replace procedure logistics_status
2 as
3 p number;
4 t number;
5 begin
6   select count(status) into p
7   from logistics
8   where status = 'Pending';
9
10  select count(status) into t
11  from logistics
12  where status = 'In Transit';
13
14  dbms_output.put_line('Total Number of Orders in Pending Status is: '|| p);
15  dbms_output.put_line('Total Number of Orders in Transit are: '|| t);
16 end;
17 /
Procedure created.
```

```
SQL> execute logistics_status;
Total Number of Orders in Pending Status is: 5
Total Number of Orders in Transit are: 3
PL/SQL procedure successfully completed.
```

4) get_stock: This is a function that gives us the total amount of stock present for a certain drug. It takes DrugId as the input.

```
create or replace function get_stock(did in drug.DRUGID%type)
```

```
return inventory.CURRENTSTOCK%type
```

```
is
```

```
s inventory.CURRENTSTOCK%type;
```

```
begin
```

```
select SUM(currentstock) into s
```

```
from inventory
```

```
where inventory.DRUGID = did;
```

```
return s;
```

```
end;
```

```
SQL> create or replace function get_stock(did in drug.DRUGID%type)
2 return inventory.CURRENTSTOCK%type
3 is
4 s inventory.CURRENTSTOCK%type;
5 begin
6   select SUM(currentstock) into s
7   from inventory
8   where inventory.DRUGID = did;
9
10  return s;
11 end;
12 /
Function created.
SQL>
```

5) get_stockr: This is a function that gives us the total stock available of a drug in all the warehouses in a particular region. It takes DrugId and RegionCode as inputs.

create or replace function get_stockr(did in drug.drugid%type, region in region.regioncode%type)

return inventory.currentstock%type

is

```
s inventory.currentstock%type;

begin

select SUM(currentstock) into s

from inventory

where buildingid in

(

    select warehouseid

    from warehouse

    where warehouse.region = region

)

and inventory.drugid = did;

return s;
```

end;

```
SQL> create or replace function get_stockr(did in drug.drugid%type, region in region.regioncode%type)
2  return inventory.currentstock%type
3  is
4      s inventory.currentstock%type;
5      begin
6          select SUM(currentstock) into s
7          from inventory
8          where buildingid in
9          (
10             select warehouseid
11             from warehouse
12             where warehouse.region = region
13         )
14         and inventory.drugid = did;
15         return s;
16     end;
17 /

Function created.

SQL>
```

6) get_totalsalesd: This is a function that that gives us how many units a particular drug has been sold across all the stores in a particular year.

create or replace function get_totalsalesd(did in drug.drugid%type, y in sales.year%type)

return sales.number_of_sales%type

is

```

s sales.number_of_sales%type;

begin

    select sum(number_of_sales) into s

    from sales

    where drugid = did and sales.year=y;

    return s;

end;

```

```

SQL> create or replace function get_totalsalesd(did in drug.drugid%type, y in sales.year%type)
2  return sales.number_of_sales%type
3  is
4  s sales.number_of_sales%type;
5  begin
6      select sum(number_of_sales) into s
7      from sales
8      where drugid = did and sales.year=y;
9
10     return s;
11 end;
12 /
Function created.

```

When the above three functions get_stock, get_stockr and get_totalsalesd are executed we get the following result.

```

SQL> declare
2  a number;
3  b number;
4  c number;
5  begin
6  a := get_stock('Dg6');
7  b := get_stockr('Dg6', 'R1');
8  c := get_totalsalesd('Dg6', 2022);
9  dbms_output.put_line('Total Stock of Dg6 is: ' || a);
10 dbms_output.put_line('Total Stock of Dg6 in Region R1 is: ' || b);
11 dbms_output.put_line('Total Sales of Dg6 is: ' || c);
12 end;
13 /
Total Stock of Dg6 is: 250
Total Stock of Dg6 in Region R1 is: 250
Total Sales of Dg6 is: 25
PL/SQL procedure successfully completed.
SQL> |

```

TRIGGERS:

1) num_of_emp: This is a trigger on employee table that gets triggered after delete or insert operations on the employee table. This displays the number of employees present after delete or insert operations are performed.

```
CREATE OR REPLACE TRIGGER num_of_emp
```

```
AFTER DELETE OR INSERT ON employee
```

```
DECLARE
```

```
    c number;
```

```
BEGIN
```

```
    select count(*) into c
```

```

        from employee;

        dbms_output.put_line('Total Number of Employees: '|| c);

END;

```

```

SQL> CREATE OR REPLACE TRIGGER num_of_emp
2 AFTER DELETE OR INSERT ON employee
3 DECLARE
4     c number;
5 BEGIN
6     select count(*) into c
7     from employee;
8     dbms_output.put_line('Total Number of Employees: '|| c);
9 END;
10 /

Trigger created.

```

```

SQL> insert into employee values ('M31','NA61','SSN61',37,'Female','AD61','PH61','30','Manager','R10','BAC61');
Total Number of Employees: 61

1 row created.

SQL> delete from employee where eid = 'M31';
Total Number of Employees: 60

1 row deleted.

```

2) avg_wage: This trigger gives us the average hourly wage of all employees after there has been an update operation on the wage column of employee table.

CREATE OR REPLACE TRIGGER avg_wage

AFTER UPDATE of wage ON employee

DECLARE

 c number;

BEGIN

 select AVG(wage) into c

 from employee;

 dbms_output.put_line('New Average wage of Employees: '|| c);

END;

```

SQL> CREATE OR REPLACE TRIGGER avg_wage
2 AFTER UPDATE of wage ON employee
3 DECLARE
4     c number;
5 BEGIN
6     select AVG(wage) into c
7     from employee;
8     dbms_output.put_line('New Average wage of Employees: '|| c);
9 END;
10 /

Trigger created.

```

```
SQL> declare
2  a number;
3  s number;
4  begin
5  a := fdb_proj.get_insant('2023/03/04');
6  s := fdb_proj.get_sml('S6', 'W6', 2023, 3);
7  fdb_proj.empsal_update('D6', 1);
8  fdb_proj.total_hours(2023, 3, 'E1');
9  dbms_output.put_line('Insurance Amount claimed on March 4th is: '||a);
10 dbms_output.put_line(' Stock moved between S6 and W6 in March 2023 is: '||s);
11 end;
12 /
New Average wage of Employees: 27.5
Wage of Employee D6 changed
Number of Hours worked by Employee E1 is: 150
Insurance Amount claimed on March 4th is: 100
Stock moved between S6 and W6 in March 2023 is: 50
PL/SQL procedure successfully completed.
SQL>
```

In this we can observe the line 'New Average wage of Employees: 27.5' in the output.

This is the result of the above trigger which was triggered due to the empsal_update procedure in the fdb_proj package.

3) accidental_log_order: This is a trigger on logistics table. This gets triggered before an insert operation on logistics table. When we want to record a new logistics order, this trigger checks whether the warehouse have enough stock of the drug in question to satisfy the order and warns us to get more stock if not present.

CREATE OR REPLACE TRIGGER accidental_log_order

BEFORE INSERT ON LOGISTICS

FOR EACH ROW

DECLARE

q number;

oq number;

w warehouse.warehouseid%type;

d drug.drugid%type;

BEGIN

oq := :new.quantity;

w := :new.warehouseid;

d := :new.good;

select SUM(currentstock) into q

from inventory

where buildingid = w and drugid = d;

```

if q >= oq then

    dbms_output.put_line('Order Accepted.');
```

else

```

    dbms_output.put_line('Order Cannot be Fulfilled ! Get More Stock.');
```

end if;

END;

```

SQL> CREATE OR REPLACE TRIGGER accidental_log_order
2 BEFORE INSERT ON LOGISTICS
3 FOR EACH ROW
4 DECLARE
5     q number;
6     oq number;
7     w warehouse.warehouseid%type;
8     d drug.drugid%type;
9 BEGIN
10     oq := :new.quantity;
11     w := :new.warehouseid;
12     d := :new.good;
13
14     select SUM(currentstock) into q
15     from inventory
16     where buildingid = w and drugid = d;
17
18     if q >= oq then
19         dbms_output.put_line('Order Accepted.');
```

20 else

```

21         dbms_output.put_line('Order Cannot be Fulfilled ! Get More Stock.');
```

22 end if;

```

23 END;
24 /

Trigger created.

SQL>
```

```

SQL> insert into logistics values('Dg5','W5','S5',TO_DATE('2023-03-06','yyyy/mm/dd'),'5000','Pending');
Order Cannot be Fulfilled ! Get More Stock.

1 row created.

SQL> |
```

SUMMARY:

In this part of the project, we have created

5 Procedures:

- empsal_update
- total_hours
- avg_hw
- age_patients
- logistics_status

5 Functions:

- get_insamt()
- get_swl()
- get_stock()
- get_stockr()
- get_totalsalesd()

3 Triggers:

- num_of_emp
- avg_wage
- accidental_log_order

1 Package: Which contains two procedures emsal_update, total_hours and the two functions get_insamt(), get_swl() in it.