DBMS LAB 9

GROUP: 4
SECTION:2.15
GRP MEMBERS:
RISHABH NANDANIA -202001209
KETAN KHUNTI - 202001213

Schemas:

1. **Admin-**
(admin_id(**pk**),name,age,gender,address) Primary key : admin_id
Functional Dependencies: admin_id -> (name,age,gender,address)
Composite attribute : address,name

=> here address is composite attribute . so schema is not in 1NF;
=> we will decompose address into (city,street).
=> we will decompose name into (first name,last name)

After 1 NF :
Admin-(admin_id(**pk**),first name,last name,age,gender,city,street)

=> here candidate key = admin_id
=> for, 2NF any proper subset of candidate key should not define any non prime attribute.
=> here, we have only admin_id which defines all the attributes.
=> So, schema is in 2NF

After 2NF :
Admin-(admin_id(**pk**),first name,last name,age,gender,city,street)

=> here super key = admin_id;

=> for 3NF . left side of functional dependency should be any super key or right side should be any prime attribute

=> Here we have admin_id in the left side for all FDs.

=> So, schema is in 3NF.

After 3NF :

Admin-(admin_id(**pk**),first name,last name,age,gender,city,street)

=>there is no case where any attribute which is not super key is determining any attributes.

=> here, admin_id is defining all the attributes, So schema is in BCNF.

After BCNF :

Admin-(admin_id(**pk**),first name,last name,age,gender,city,street)

2. **Beer-** ((beer_id,beer_name)**(pk**),alcohol%,cost,availability)

Primary key : (beer_id,beer_name)

Functional Dependencies : (beer_id,beer_name) -> (alcohol% ,cost , availability)

=> there is no composite or multivalued attribute nor any partial dependency.

=> So, schema is in 2NF.

=> there is no transitive dependency. So, schema is in 3NF.

=> there is no case where any attribute which is not super key is determining any attributes.So, schema is in BCNF.

Final Schema : **Beer-** ((beer_id,beer_name)**(pk**),alcohol%,cost,availability)

3. **Raw materials-**((beer_id,beer_name)(**pk)**,barley,naked rice,oats,syrup,sugars,enzymes,glucose,yeast,calcium,magnesium)

Primary key : (beer_id,beer_name)
Functional Dependencies : (beer_id,beer_name) -> (barley,naked rice,oats,syrup,sugars,enzymes,glucose,yiest,calcium,magnesium)

=> there is no composite or multivalued attribute nor any partial dependency.
=> So, schema is in 2NF.
=> there is no transitive dependency. So, schema is in 3NF.
=> there is no case where any attribute which is not super key is determining any attributes.So, schema is in BCNF.

Final schema : ((beer_id,beer_name)(**pk)**,barley,naked rice,oats,syrup,sugars,enzymes,glucose,yeast,calcium,magnesium)

4. **Buyer-**
   (buyer_id(**pk**),(beer_id,beer_name)(**fk**),name,address,quantity,id proof(**business constraint**),contact_no)

   Primary key : buyer_id
   Foreign key : (beer_id,beer_name)
   Multivalued attribute : contact_no
   Composite attribute : name(first name,last name) , address(city,street)
   Functional Dependencies : buter_id -> (name,address,quantity,id proof(**business constraint**),contact_no),

=> Here , we have composite and multivalued attributes. So, schema is not in 1NF.
=> decompose name and address and make separate table for multivalued attribute.

After 1NF :
Buyer- (buyer_id(**pk**),(beer_id,beer_name)(**fk**),first name,last name,city street,quantity,id proof(**business constraint**))
Buyer_contact-(buyer_id,contact_no);

=> there is no partial dependency . so schema is in 2 NF.
=> there is no transitive dependency. So, schema is in 3NF.
=> there is no case where any attribute which is not super key is determining any attributes.So, schema is in BCNF.

Final tables:

Buyer- (buyer_id(**pk**),(beer_id,beer_name)(**fk**),first name,last name,city street,quantity,id proof(**business constraint**))
Buyer_contact-(buyer_id,contact_no);

5. **Employee-**(emp_id(**pk**),name,age,gender,address,department,salary) Primary key : emp_id

Functional Dependencies :
emp_id->(name,age,gender,address,department,salary)

Composite attribute : address,name

=> here address is composite attribute . so schema is not in 1NF;
=> we will decompose address into (city,street).
=> we will decompose name into (first name,last name)

After 1 NF :
Employee-(emp_id(**pk**),first name,last name,,age,gender,city,street,department,salary)

=> there is no composite or multivalued attribute nor any partial dependency.
=> So, schema is in 2NF.
=> there is no transitive dependency. So, schema is in 3NF.
=> there is no case where any attribute which is not super key is determining any attributes.So, schema is in BCNF.

Final schema : **Employee-**(emp_id(**pk**),first name,last name,,age,gender,city,street,department,salary)

6. **Production department-**
(production_id(**pk**),(beer_id,beer_name)**(fk**),req_amount,prod_time,cost)

Primary key : production_id

Foreign key : (beer_id,beer_name)
Functional Dependencies : (production_id)-> (req_amount,prod_time,cost)

=> there is no composite or multivalued attribute nor any partial dependency.
=> So, schema is in 2NF.
=> there is no transitive dependency. So, schema is in 3NF.
=> there is no case where any attribute which is not super key is determining any attributes.So, schema is in BCNF.

7. **ingredients-**(ingre_id(**pk**),ingre_name,availability,req_amount,amount_to_ b uy)

Primary key : ingre_id
Functional Dependencies : (ingre_id) -> (ingre_name,availability,req_amount,amount_to_buy)

=> there is no composite or multivalued attribute nor any partial dependency.
=> So, schema is in 2NF.
=> there is no transitive dependency. So, schema is in 3NF.
=> there is no case where any attribute which is not super key is determining any attributes.So, schema is in BCNF.

Final schema : (ingre_id) -> (ingre_name,availability,req_amount,amount_to_buy)

8. **machines**-(mach_id(**pk**),mach_name,type,working,mach_cost,power_eff,m a intenance)

Primary key :mach_id
Functional dependencies : mach_id -> (mach_name,type,working,mach_cost,power_eff,maintenance)

mach_name -> type

=> there is no composite or multivalued attribute nor any partial dependency.
=> So, schema is in 2NF.
=> Here , mach_name is defining machine type, which is not super key. So, schema is not in 3NF.
=> we will decompose above schema into 2 different schemas

8.1 machines-(mach_id(**pk**),working,mach_cost,power_eff,ma intenance)
8.2 machine_type-(machine name(**PK**) , machine type);

=> if we join above schemas we will get original and also functional dependency is holding on decomposed schema.
=>so, schema is in 3 NF.

For BCNF :

=> Here for first schema, mach_id is defining all the attributes. so it is in BCNF.
=> For the second schema, machine name is defining machine type and machine name is candidate or primary key of schema. so, it is in BSNF.

9. **Stock-**
(stock_id(**pk**),(beer_id,beer_name)(**fk**),stock,stock_req,prod_stock,adv_amount)
Primary key : stock id
Foreign key : (beer_id,beer_name)
Functional Dependencies:stock_id -> (stock,stock_req,prod_stock,adv_amount)

=> there is no composite or multivalued attribute nor any partial dependency.
=> So, schema is in 2NF.
=> there is no transitive dependency. So, schema is in 3NF.
=> there is no case where any attribute which is not super key is determining any attributes.So, schema is in BCNF.

Final Schema : **Stock-**
(stock_id(**pk**),(beer_id,beer_name)(**fk**),stock,stock_req,prod_stock,adv_amount)

10. **Sales-**(sales_id(**pk**),beer_name,cost,demand,profit,sales,total_rev)
Primary key : sales_id

Functional Dependency : sales_id -> (beer_name,cost,demand,profit,sales,total_rev)

=> there is no composite or multivalued attributes, nor any partial key dependencies. So, schema is is 2NF.

=> there is no transitive dependency. So, schema is in 3NF.

=> there is no case where any attribute which is not super key is determining any attributes.So, schema is in BCNF.

FInal schema : **Sales-**(sales_id(**pk**),beer_name,cost,demand,profit,sales,total_rev)
Primary key : sales_id

DDL SCRIPTS FOR SCHEMAS

1. Admin-(admin_id(**pk**),first name,last name,age,gender,city,street)

DDL SCRIPT :

```
-- Table: 202001213_db.Admin

-- DROP TABLE IF EXISTS "202001213_db"."Admin";

CREATE TABLE IF NOT EXISTS "202001213_db"."Admin" (
admin_id integer NOT NULL,
first_name character varying COLLATE pg_catalog."default" NOT NULL,
last_name character varying COLLATE pg_catalog."default" NOT NULL, age integer NOT
NULL,
gender character varying COLLATE pg_catalog."default" NOT NULL,
email character varying COLLATE pg_catalog."default",
city character varying COLLATE pg_catalog."default" NOT NULL,
street character varying COLLATE pg_catalog."default" NOT NULL,
CONSTRAINT "Admin_pkey" PRIMARY KEY (admin_id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS "202001213_db"."Admin"
OWNER to postgres;
```

2. **Beer-** ((beer_id,beer_name)**(pk**),alcohol%,cost,availability)

```
-- Table: 202001213_db.Beer

-- DROP TABLE IF EXISTS "202001213_db"."Beer";

CREATE TABLE IF NOT EXISTS
"202001213_db"."Beer" (
        beer_id integer NOT NULL,
        beer_name character varying COLLATE pg_catalog."default" NOT NULL,
```

```
        "alcohol%" double precision NOT NULL,
        cost integer NOT NULL,
        availability boolean NOT NULL,
        CONSTRAINT "Beer_pkey" PRIMARY KEY (beer_id,beer_name)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS "202001213_db"."Beer"
        OWNER to postgres;
```

3. **Raw materials-**((beer_id,beer_name)(**pk)**,barley,naked rice,oats,syrup,sugars,enzymes,glucose,yeast,calcium,magnesium)

```
-- Table: 202001213_db.Raw_materials

-- DROP TABLE IF EXISTS "202001213_db"."Raw_materials";

CREATE TABLE IF NOT EXISTS
"202001213_db"."Raw_materials" (
        beer_id integer NOT NULL,
        beer_name character varying COLLATE
        pg_catalog."default", naked_rice boolean,
        oats boolean,
        syrup boolean,
        sugars boolean,
        enzymes boolean,
        glucose boolean,
        yiest boolean,
        calcium boolean,
        magnesium boolean,
        barley boolean,
        CONSTRAINT "Raw_materials_pkey" PRIMARY KEY (beer_id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS
        "202001213_db"."Raw_materials" OWNER to
        postgres;
```

4. **Employee-**(emp_id(**pk**),first name,last name,,age,gender,city,street,department,salary)

```
-- Table: 202001213_db.Employee

-- DROP TABLE IF EXISTS "202001213_db"."Employee";

CREATE TABLE IF NOT EXISTS "202001213_db"."Employee" (
emp_id integer NOT NULL,
first_name character varying COLLATE pg_catalog."default" NOT NULL,
last_name character varying COLLATE pg_catalog."default" NOT NULL,
email character varying COLLATE pg_catalog."default",
        age integer NOT NULL,
gender character varying COLLATE pg_catalog."default" NOT NULL,
city character varying COLLATE pg_catalog."default" NOT NULL,
street character varying COLLATE pg_catalog."default" NOT NULL,
department character varying COLLATE pg_catalog."default" NOT NULL, salary double precision NOT NULL,
CONSTRAINT "Employee_pkey" PRIMARY KEY (emp_id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS "202001213_db"."Employee"
OWNER to postgres;
```

5. Buyer- (buyer_id(**pk**),(beer_id,beer_name)(**fk**),first name,last name,city street,quantity,id proof(**business constraint**));

```
-- Table: 202001213_db.Buyer

-- DROP TABLE IF EXISTS "202001213_db"."Buyer";

CREATE TABLE IF NOT EXISTS "202001213_db"."Buyer"
(
```

```
        buyer_id integer NOT NULL,
        first_name character varying COLLATE pg_catalog."default" NOT
NULL,
        last_name character varying COLLATE pg_catalog."default" NOT
NULL,
        city character varying COLLATE pg_catalog."default" NOT NULL,
        street character varying COLLATE pg_catalog."default" NOT NULL,
        quantity integer NOT NULL,
        id_proof character varying COLLATE pg_catalog."default" NOT NULL,
        beer_id integer NOT NULL,
        beer_name character varying COLLATE pg_catalog."default" NOT
NULL,
    FOREIGN KEY (beer_id,beer_name) REFERENCES
"202001213_db"."Beer" (beer_id,beer_name),
        CONSTRAINT "Buyer_pkey" PRIMARY KEY (buyer_id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS "202001213_db"."Buyer"
        OWNER to postgres;
```

6. **Buyer_contact**-(buyer_id,contact_no);

```
-- Table: 202001213_db.Buyer_Contact

-- DROP TABLE IF EXISTS "202001213_db"."Buter_Contact";


CREATE TABLE IF NOT EXISTS "202001213_db"."Buyer_Contact"
(
        buyer_id integer NOT NULL,
    contact_no character varying COLLATE pg_catalog."default" NOT NULL
)
```

```
TABLESPACE pg_default;

ALTER TABLE IF EXISTS "202001213_db"."Buyer_Contact"
    OWNER to postgres;
```

7. Production department-

(production_id(**pk**),(beer_id,beer_name)**(fk**),req_amount,prod_time,cost)

```
-- Table: 202001213_db.Production_Department



-- DROP TABLE IF EXISTS "202001213_db"."Production_Department";



CREATE TABLE IF NOT EXISTS
"202001213_db"."Production_Department"
(
        production_id integer NOT NULL,
        req_amount integer NOT NULL,
        production_time character varying COLLATE pg_catalog."default"
NOT NULL,
    beer_id integer NOT NULL,
        beer_name character varying COLLATE pg_catalog."default" NOT
NULL,
```

```
        cost double precision NOT NULL,
    FOREIGN KEY (beer_id,beer_name) REFERENCES
"202001213_db"."Beer" (beer_id,beer_name),
        CONSTRAINT "Production_Department_pkey" PRIMARY KEY
(production_id)
  )

  TABLESPACE pg_default;

  ALTER TABLE IF EXISTS "202001213_db"."Production_Department"
        OWNER to postgres;
```

8. ingredients-(ingre_id(**pk**),ingre_name,availability,req_amount,amount_to_buy)

```
-- Table: 202001213_db.Ingredients

-- DROP TABLE IF EXISTS "202001213_db"."Ingredients";

CREATE TABLE IF NOT EXISTS "202001213_db"."Ingredients"
(
  ingre_id integer NOT NULL,
  ingre_name character varying COLLATE pg_catalog."default" NOT NULL,
  availability boolean NOT NULL,
  req_amount integer NOT NULL,
  available_amount integer NOT NULL,
  amount_to_buy integer NOT NULL,
  CONSTRAINT "Ingredients_pkey" PRIMARY KEY (ingre_id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS "202001213_db"."Ingredients"
```

```
    OWNER to postgres;



    TABLESPACE pg_default;


    ALTER TABLE IF EXISTS "202001213_db"."Ingredients"

    OWNER to postgres;
```

9. machines-(mach_id(**pk**),working,mach_cost,power_eff,ma intenance)

```
-- Table: 202001213_db.Machines

-- DROP TABLE IF EXISTS "202001213_db"."Machines";

CREATE TABLE IF NOT EXISTS "202001213_db"."Machines"
(
    mach_id integer NOT NULL,
    mach_name  character varying COLLATE pg_catalog."default",
    working boolean NOT NULL,
    mach_cost double precision NOT NULL,
    power_efficiency  integer ,
    maintanance character varying COLLATE pg_catalog."default",
```

```
        CONSTRAINT "Machines_pkey" PRIMARY KEY (mach_id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS "202001213_db"."Machines"
        OWNER to postgres;
```

10. Stock-

(stock_id(**pk**),(beer_id,beer_name)(**fk**),stock,stock_req,prod_stock,adv_amount)

```
-- Table: 202001213_db.Stock

-- DROP TABLE IF EXISTS "202001213_db"."Stock";

CREATE TABLE IF NOT EXISTS "202001213_db"."Stock"
(
        stock_id integer NOT NULL,
        stock double precision NOT NULL,
        stock_req double precision NOT NULL,
        prod_stock double precision NOT NULL,
        adv_amount double precision NOT NULL,
    beer_id integer NOT NULL,
        beer_name character varying COLLATE pg_catalog."default" NOT NULL,
    FOREIGN KEY (beer_id,beer_name) REFERENCES "202001213_db"."Beer"
(beer_id,beer_name),
        CONSTRAINT "Stock_pkey" PRIMARY KEY (stock_id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS "202001213_db"."Stock"
        OWNER to postgres;
```

11 . Sales-(sales_id(**pk**),beer_name,cost,demand,profit,sales,total_rev)

-- Table: 202001213_db.Sales

-- DROP TABLE IF EXISTS "202001213_db"."Sales";

CREATE TABLE IF NOT EXISTS "202001213_db"."Sales"

(

sales_id integer NOT NULL,

beer_id integer NOT NULL,

beer_name character varying COLLATE pg_catalog."default" NOT NULL,

cost double precision NOT NULL,

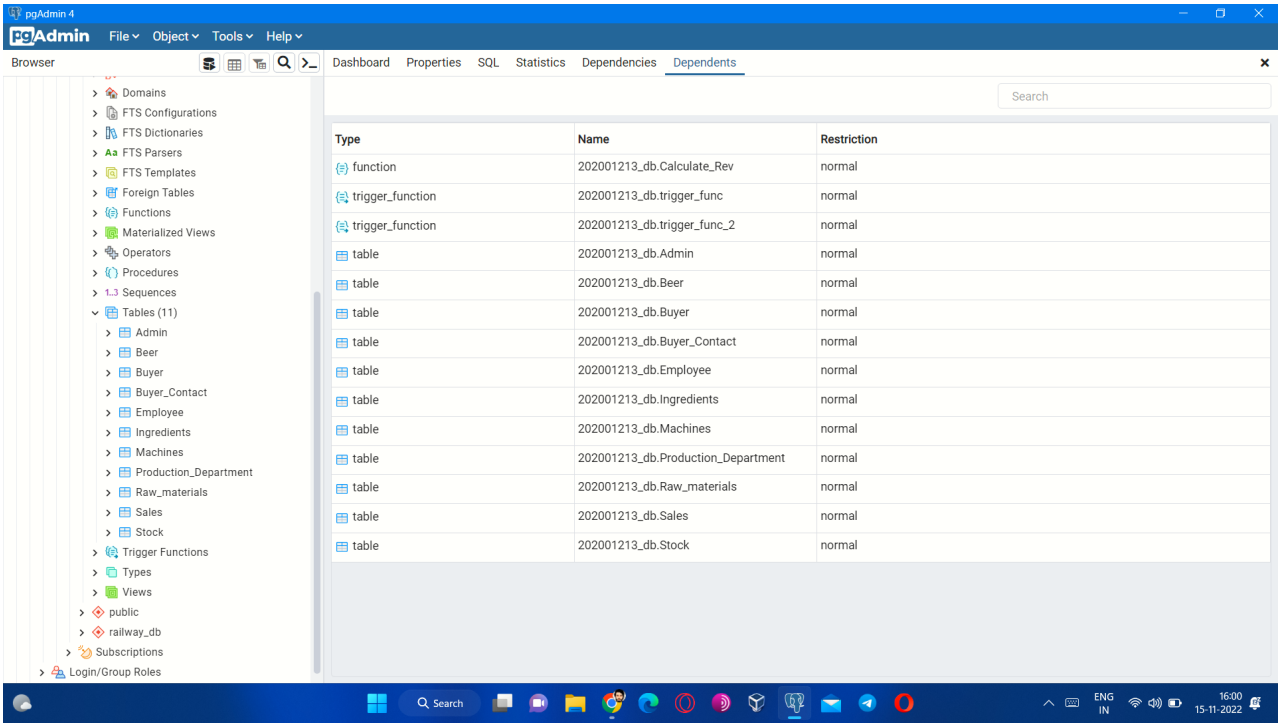
demand character varying COLLATE pg_catalog."default",

profit double precision,

sales integer,

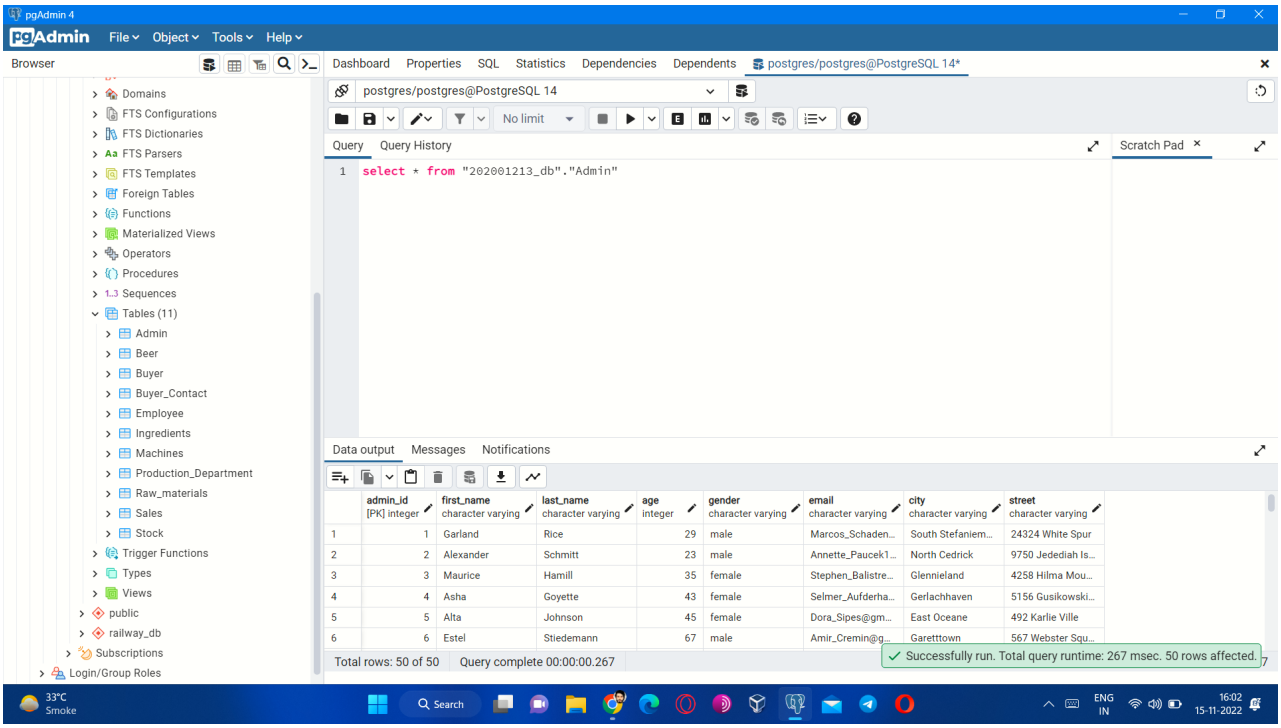
total_rev double precision NOT NULL,

```
        FOREIGN KEY (beer_id,beer_name) REFERENCES
"202001213_db"."Beer" (beer_id,beer_name),
        CONSTRAINT "Sales_pkey" PRIMARY KEY (sales_id)
)
TABLESPACE pg_default;
ALTER TABLE IF EXISTS "202001213_db"."Sales"
        OWNER to postgres;
```

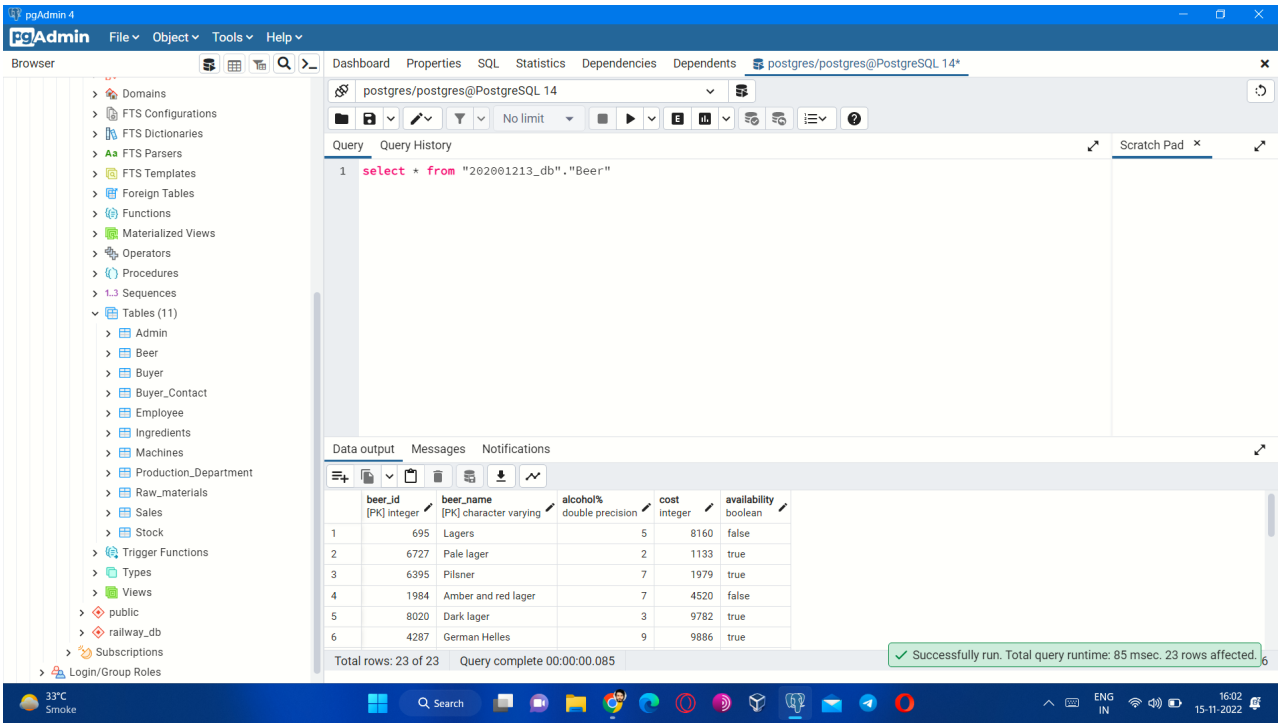


Data Snapshots: Put the snapshot of select * queries of all the tables after data insertion. Mention the number of records in each table.

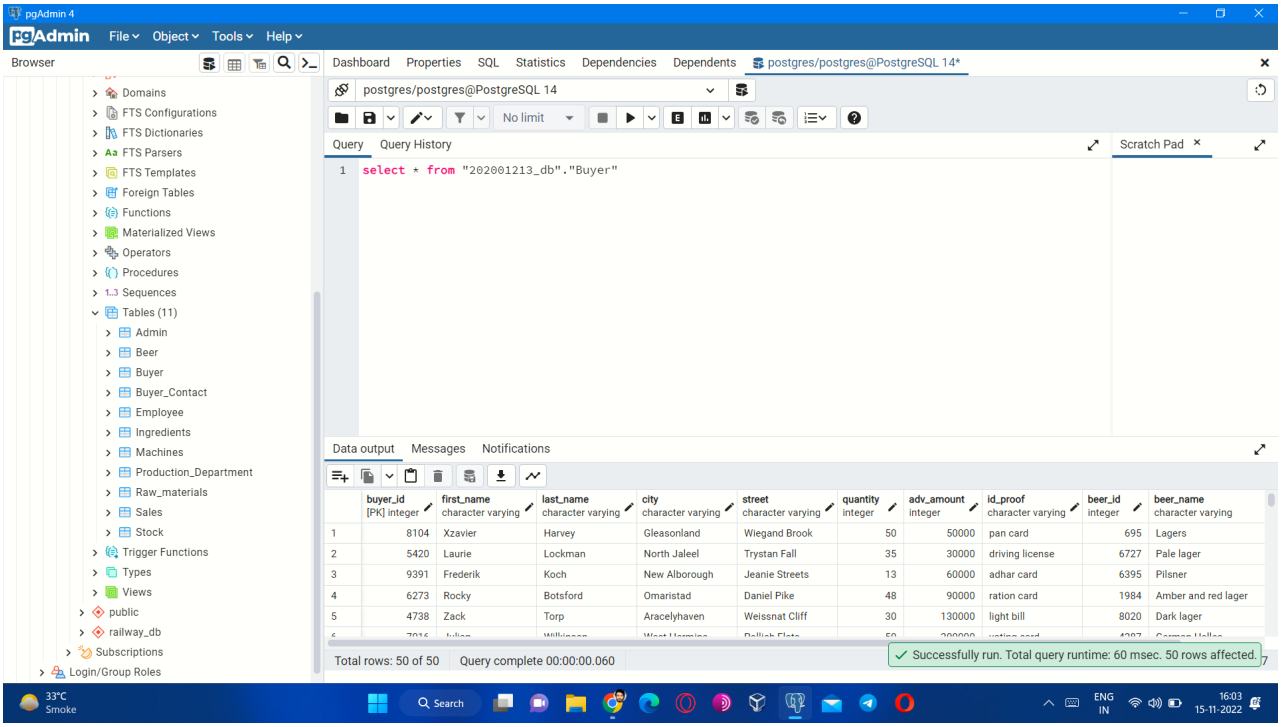
1. **Admin**



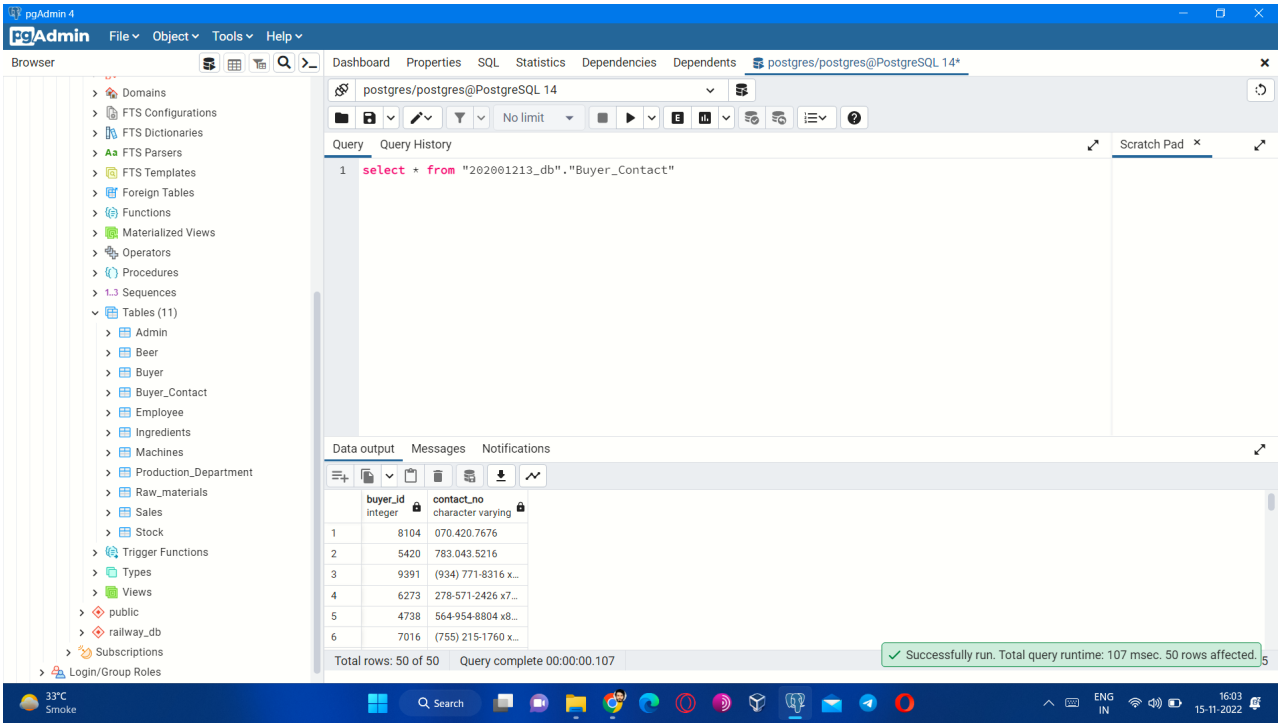
2. **Beer**


pgAdmin 4
File Object Tools Help
Browser
Dashboard Properties SQL Statistics Dependencies Dependents postgres/postgres@PostgreSQL 14*
postgres/postgres@PostgreSQL 14
No limit
Query Query History
Scratch Pad
1 select * from "202001213_db"."Beer"
Data output Messages Notifications
beer_id [PK] integer
beer_name [PK] character varying
alcohol% double precision
cost integer
availability boolean
1 695 Lagers 5 8160 false
2 6727 Pale lager 2 1133 true
3 6395 Pilsner 7 1979 true
4 1984 Amber and red lager 7 4520 false
5 8020 Dark lager 3 9782 true
6 4287 German Helles 9 9886 true
Total rows: 23 of 23
Query complete 00:00:00.085
Successfully run. Total query runtime: 85 msec. 23 rows affected.

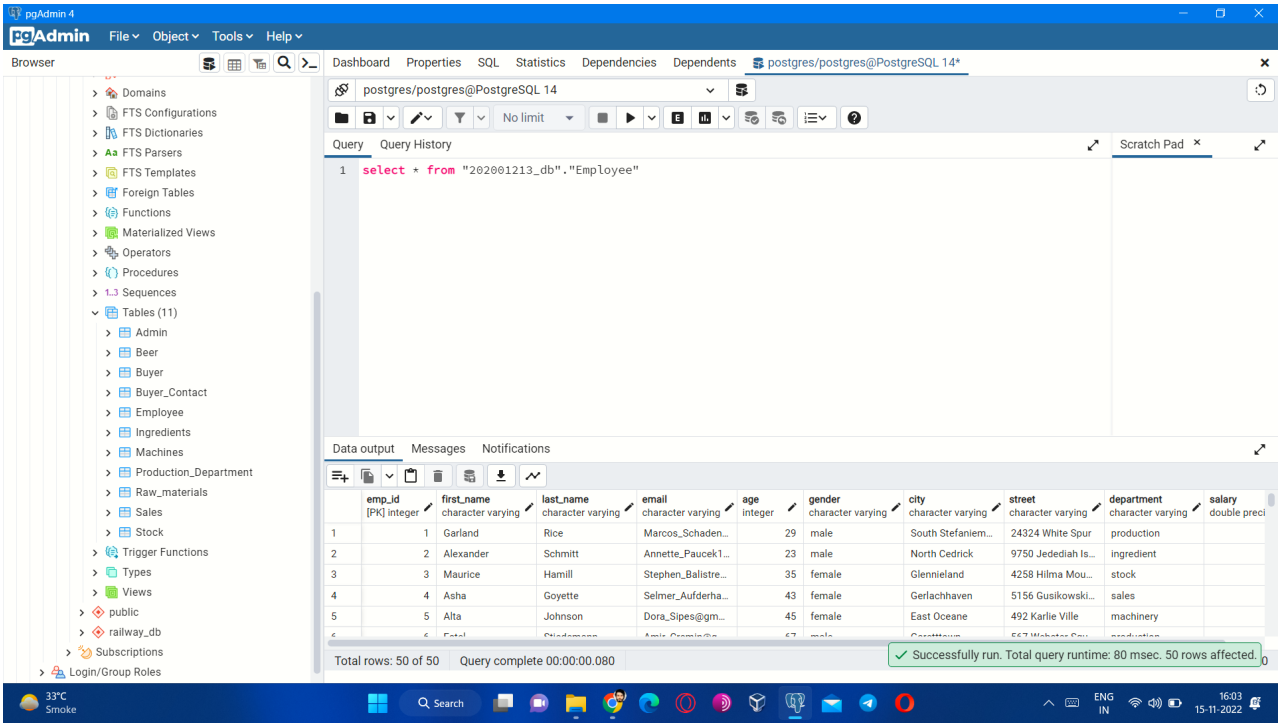
3. Buyer


pgAdmin 4
File Object Tools Help
Browser
Dashboard Properties SQL Statistics Dependencies Dependents postgres/postgres@PostgreSQL 14*
postgres/postgres@PostgreSQL 14
No limit
Query Query History
Scratch Pad
1 select * from "202001213_db"."Buyer"
Data output Messages Notifications
buyer_id [PK] integer
first_name character varying
last_name character varying
city character varying
street character varying
quantity integer
adv_amount integer
id_proof character varying
beer_id integer
beer_name character varying
1 8104 Xzavier Harvey Gleasonland Wiegand Brook 50 50000 pan card 695 Lagers
2 5420 Laurie Lockman North Jaleel Trystan Fall 35 30000 driving license 6727 Pale lager
3 9391 Frederik Koch New Alborough Jeanie Streets 13 60000 adhar card 6395 Pilsner
4 6273 Rocky Botsford Omaristad Daniel Pike 48 90000 ration card 1984 Amber and red lager
5 4738 Zack Torp Aracelyhaven Weissnat Cliff 30 130000 light bill 8020 Dark lager
Total rows: 50 of 50
Query complete 00:00:00.060
Successfully run. Total query runtime: 60 msec. 50 rows affected.

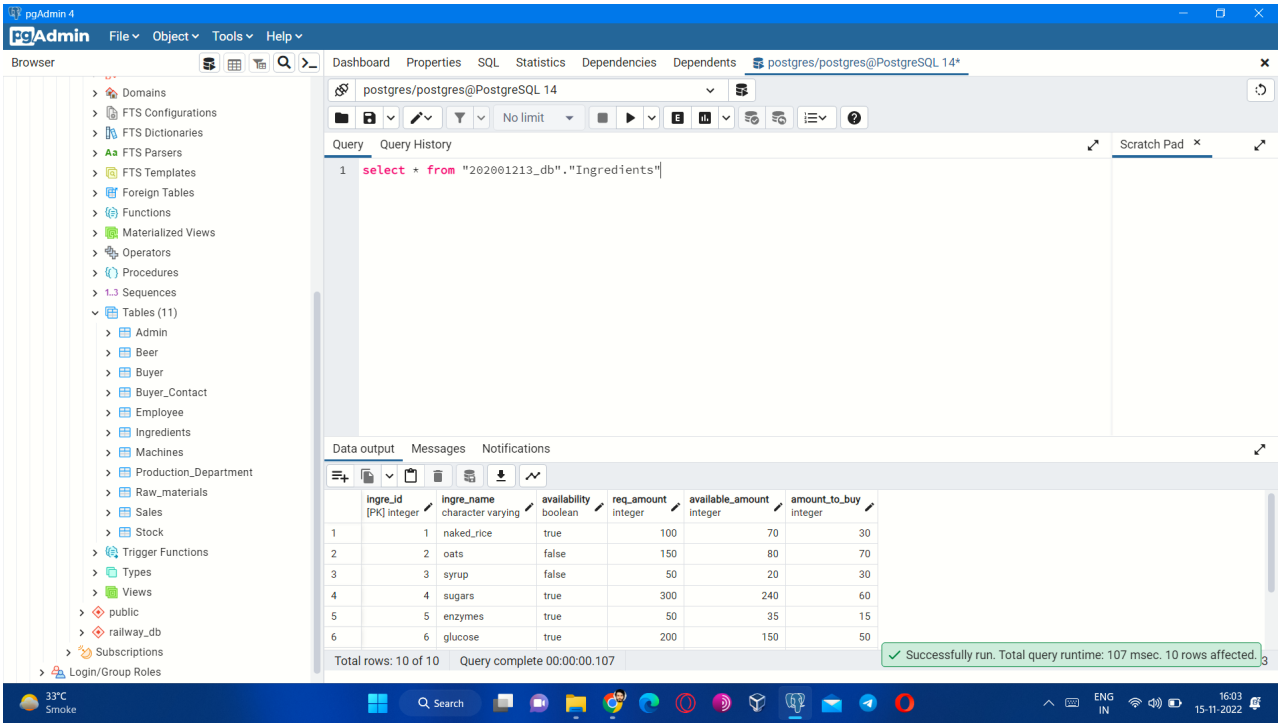
4. Buyer_Contact



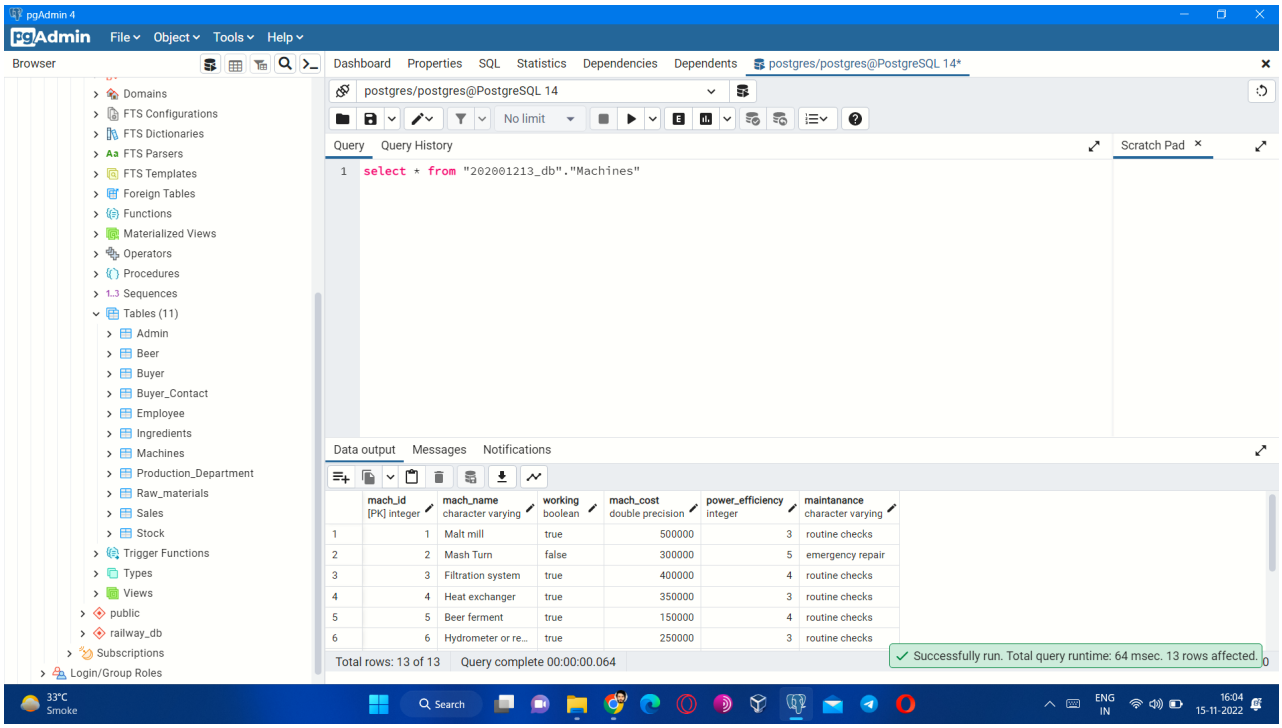
5. Employee



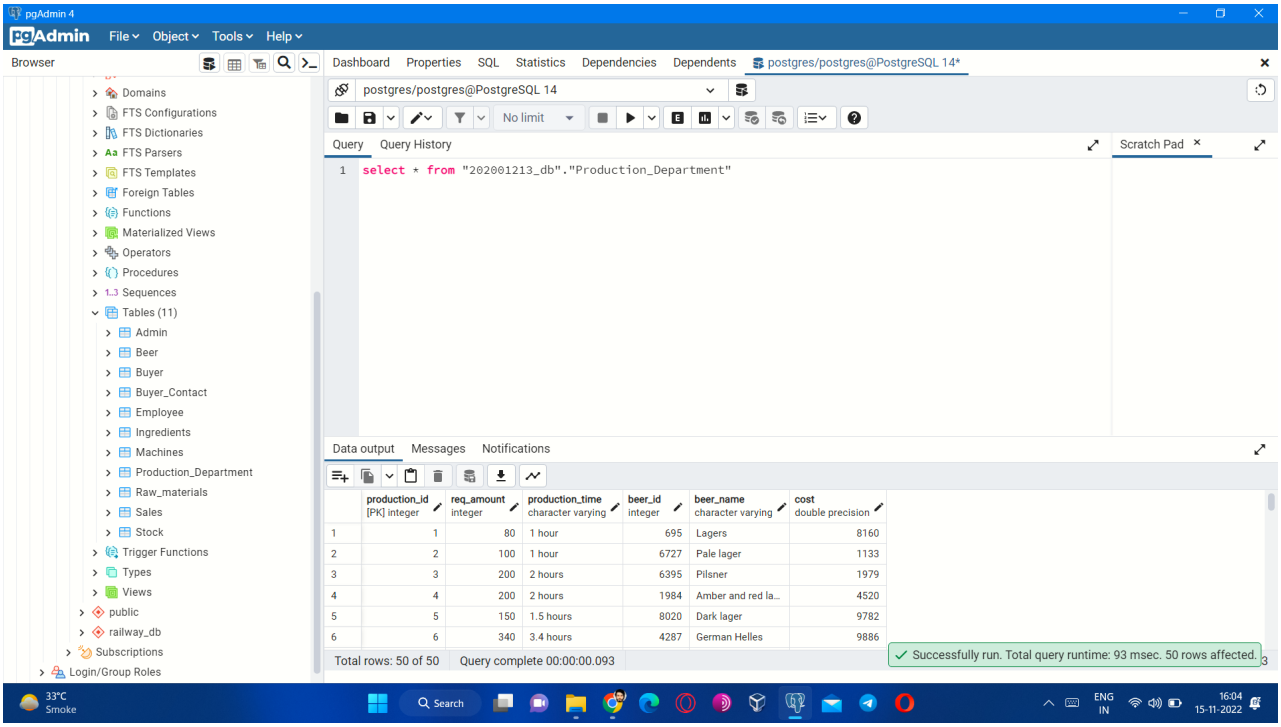
6. Ingredients



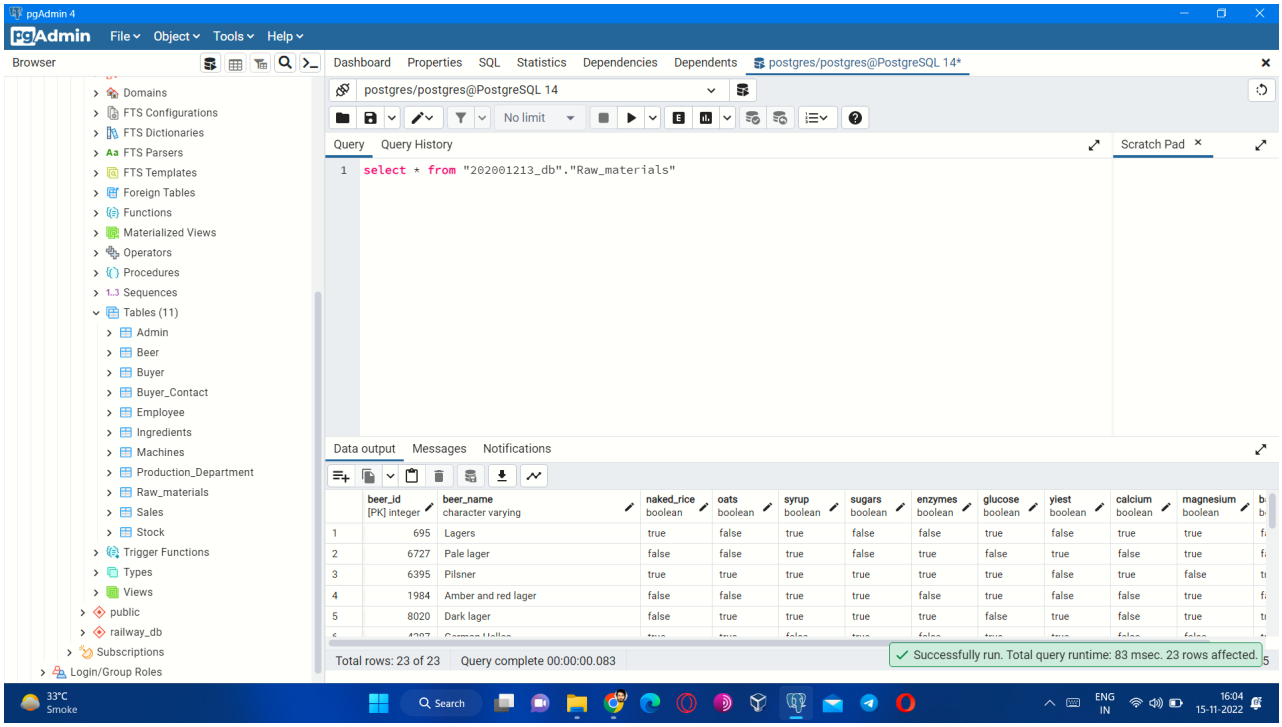
7. Machines



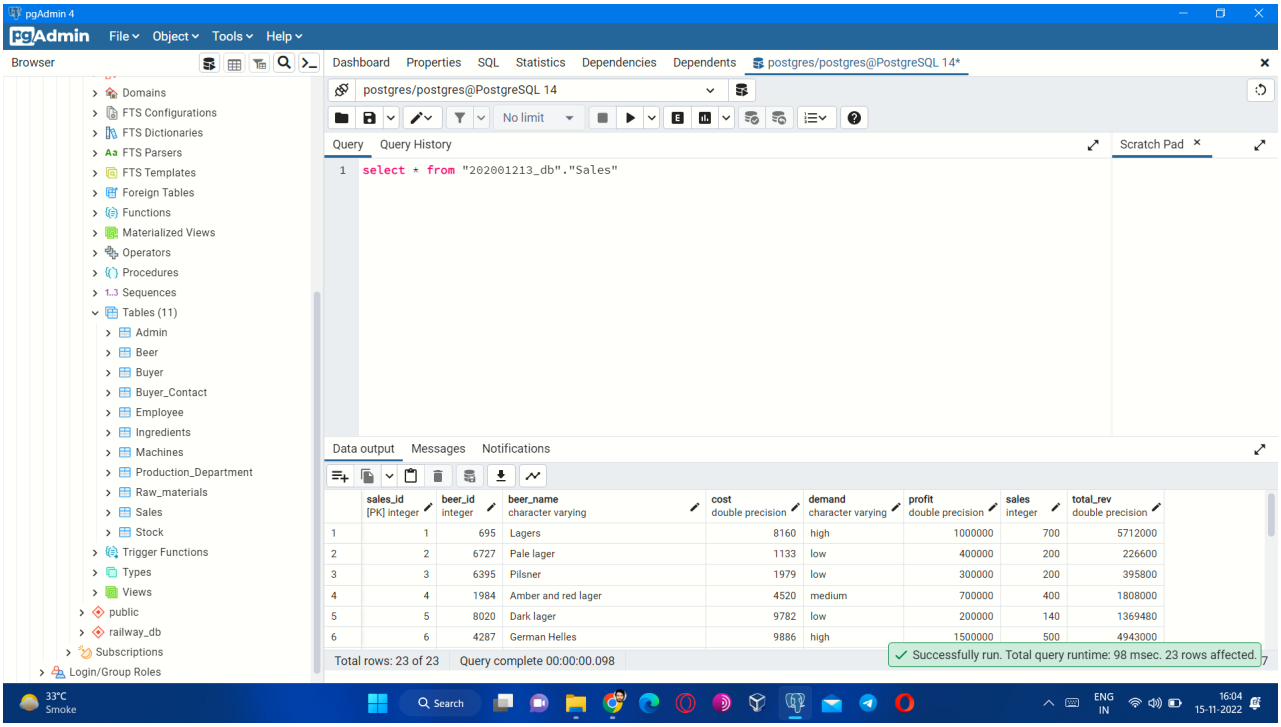
8. Production_Department



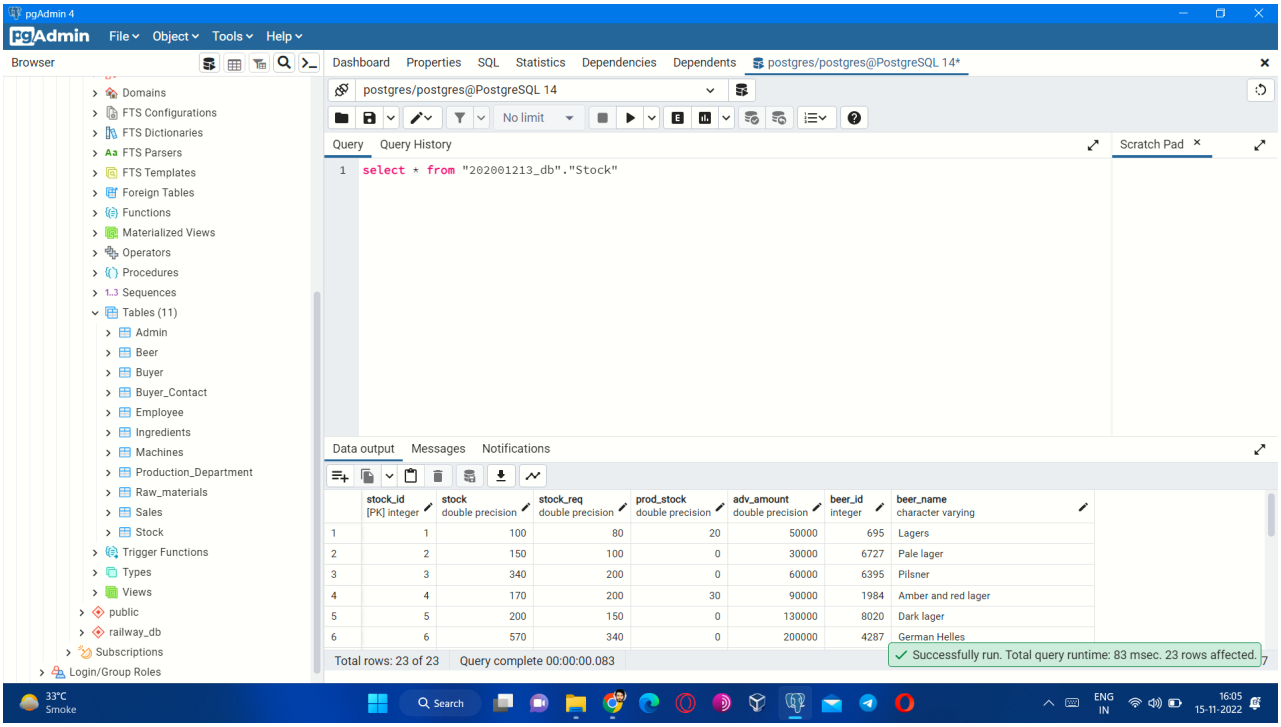
9. Row_Materials



10. Sales



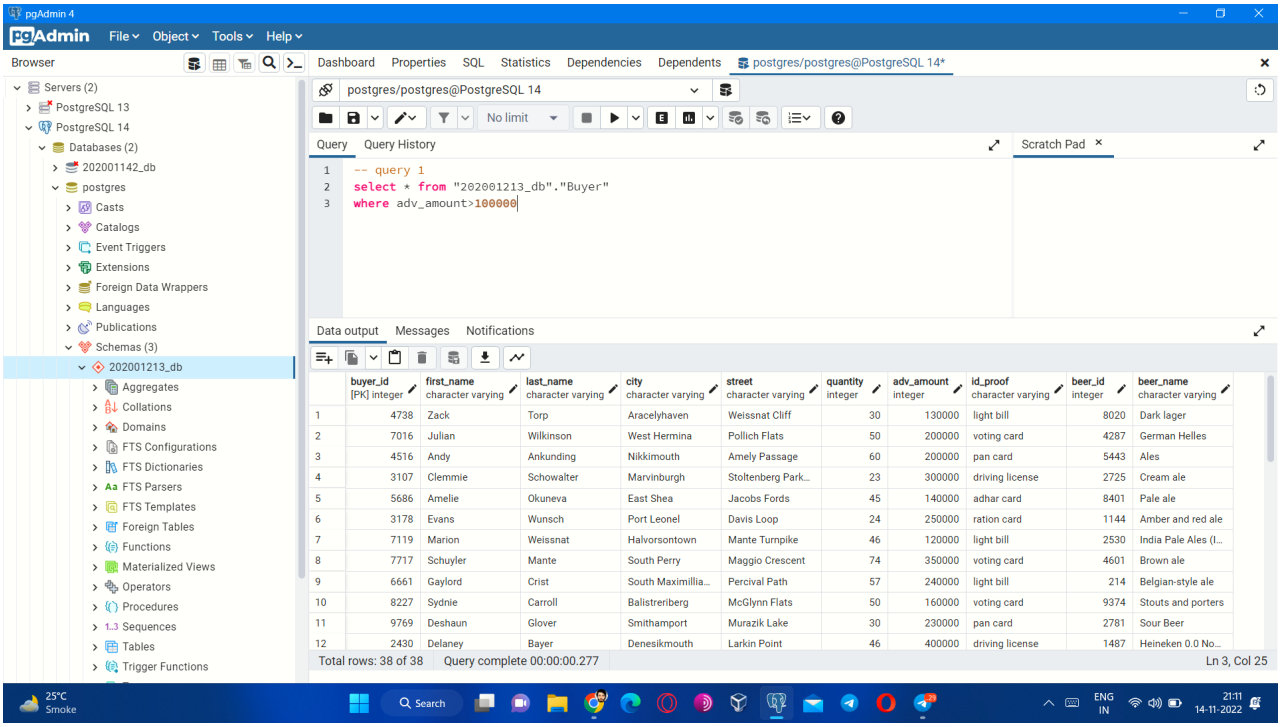
11. Stock



20 Queries

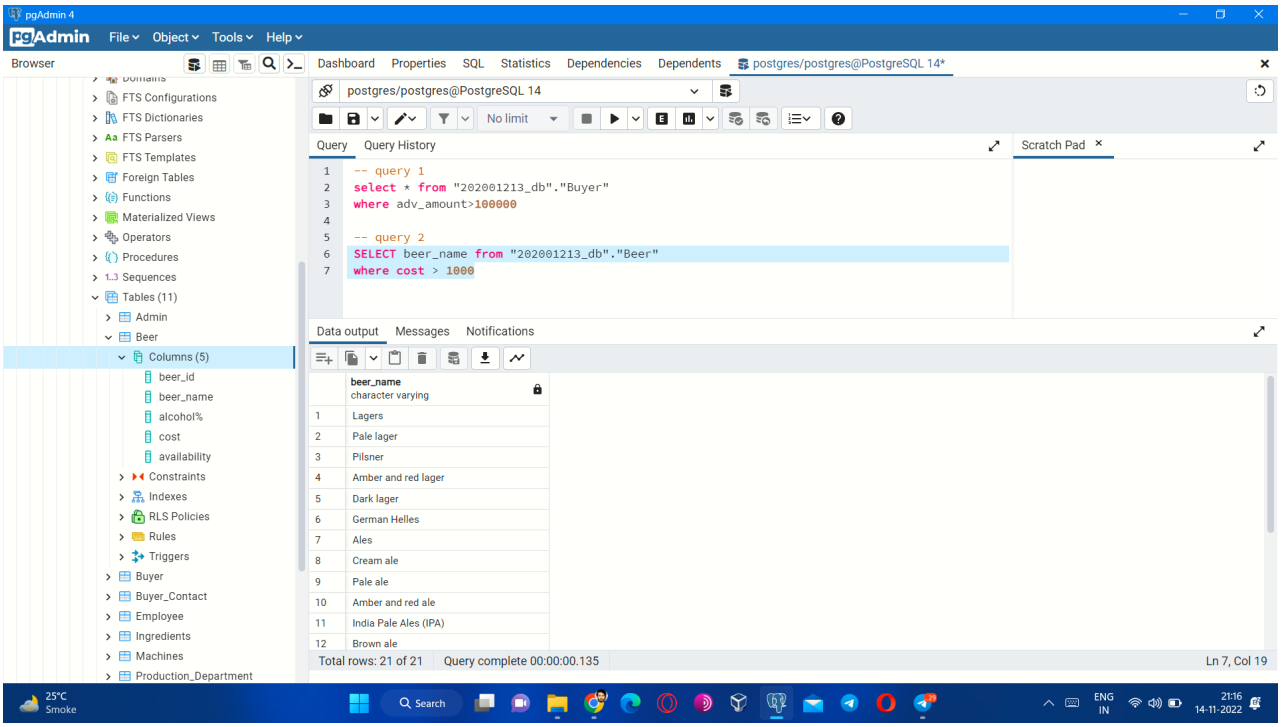
1. select buyers who have paid advance amount more than 100000.

Query : select * from "202001213_db"."Buyer"
where adv_amount>100000



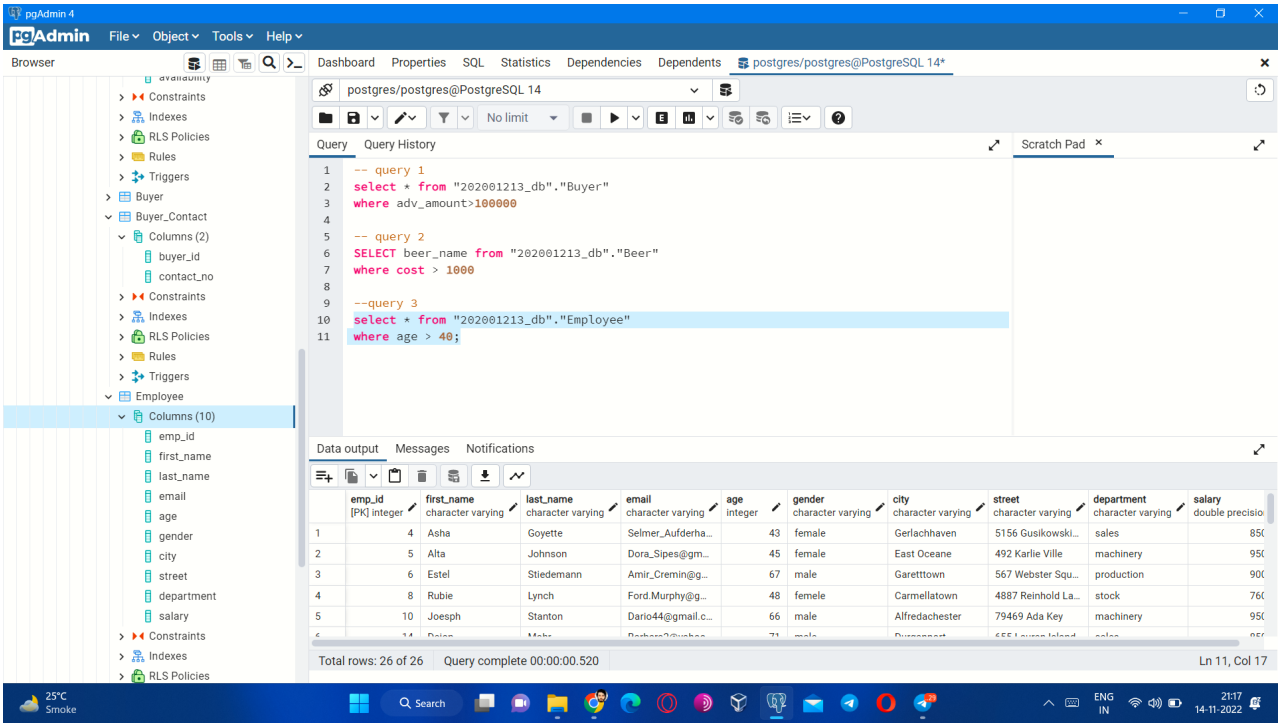
2. select beer whose price is more than 1000.

Query : SELECT beer_name from "202001213_db"."Beer"
where cost > 1000



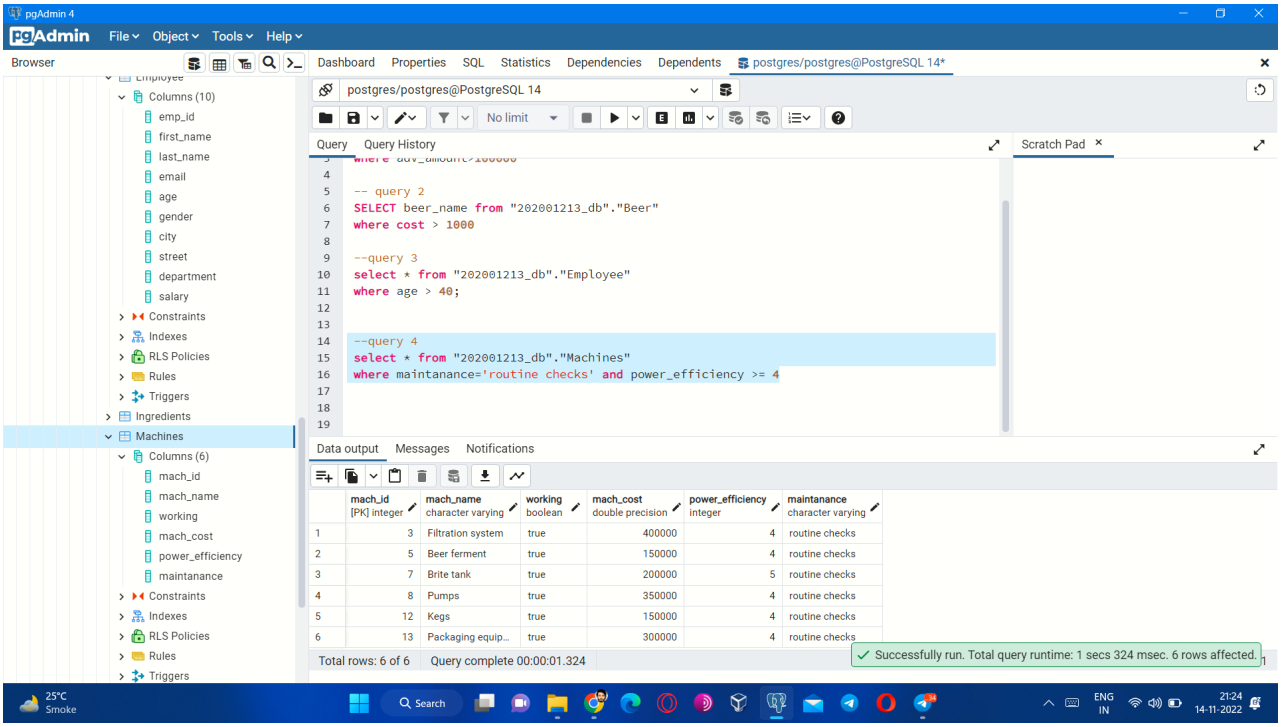
3. select employee whose age is more than 40.

Query : select * from "202001213_db"."Employee"
where age > 40;



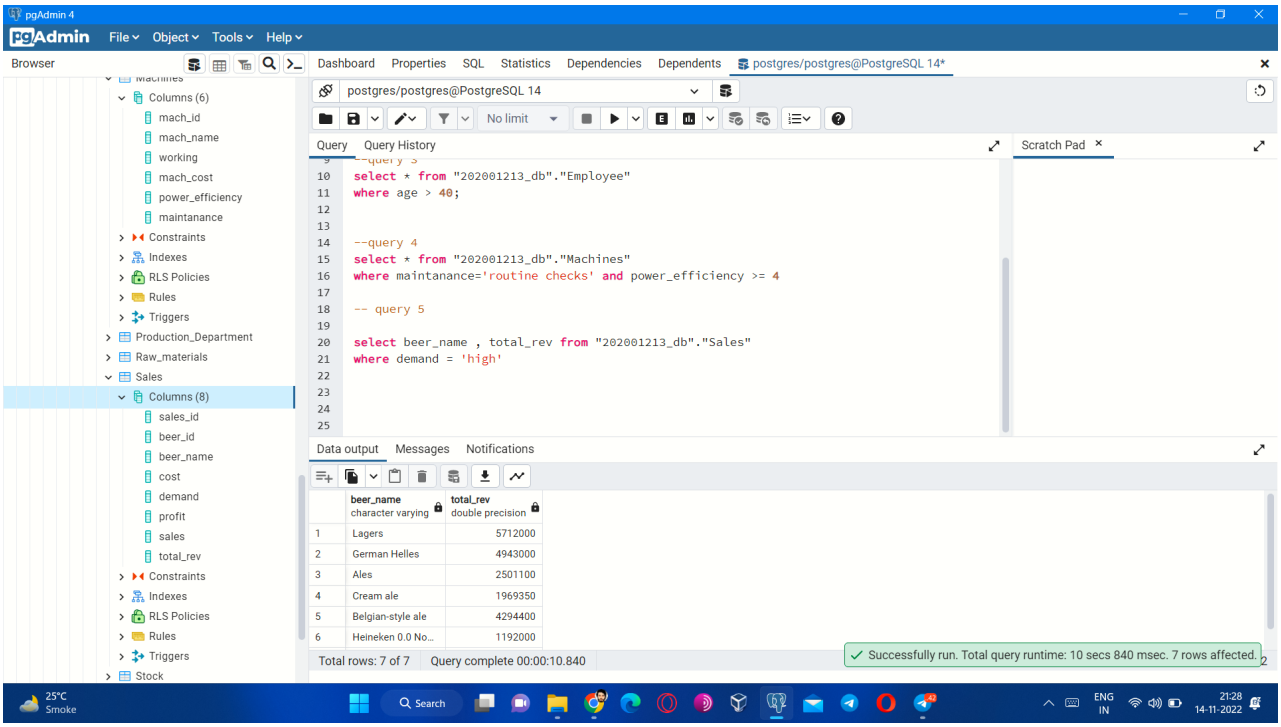
4. select machine which has power efficiency more than or equal to 4 and maintenance=”routine checks”

Query : select * from "202001213_db"."Machines"
where maintanance='routine checks' and power_efficiency >= 4



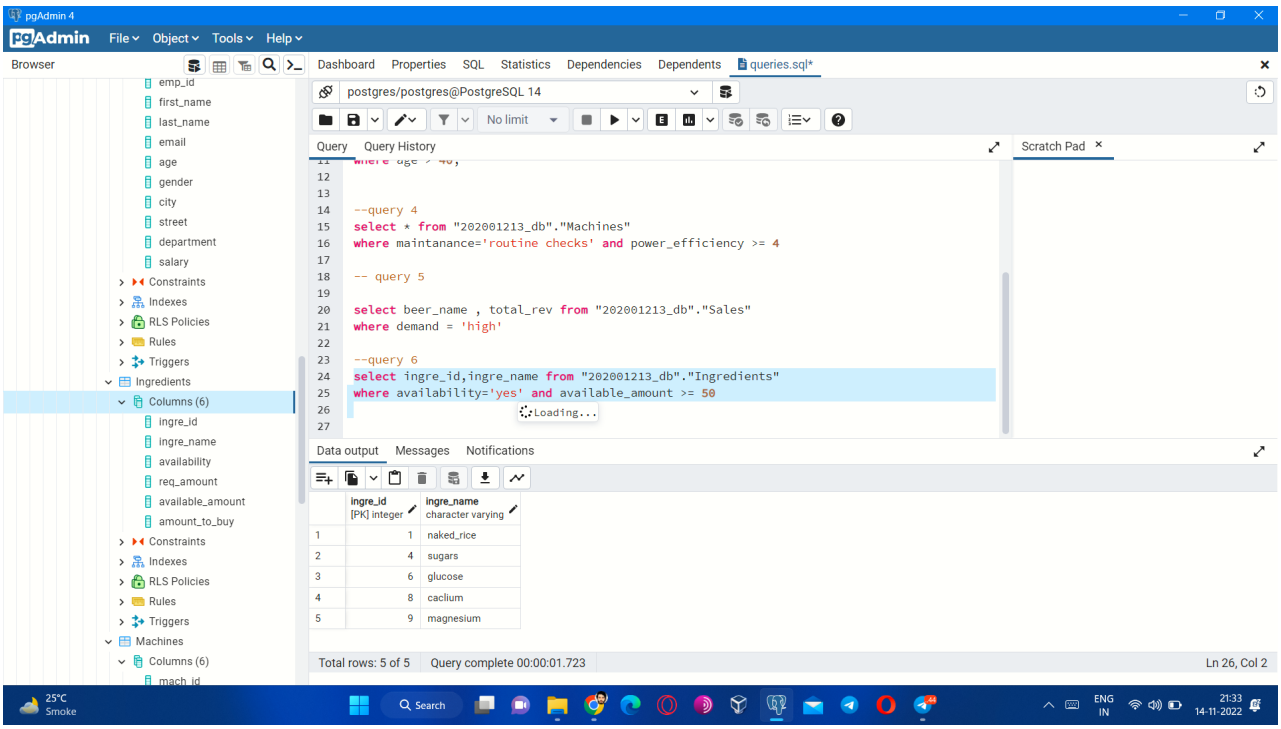
5. select name and total revenue of beer whose demand is high

Query : select beer_name , total_rev from "202001213_db"."Sales"
where demand = 'high'



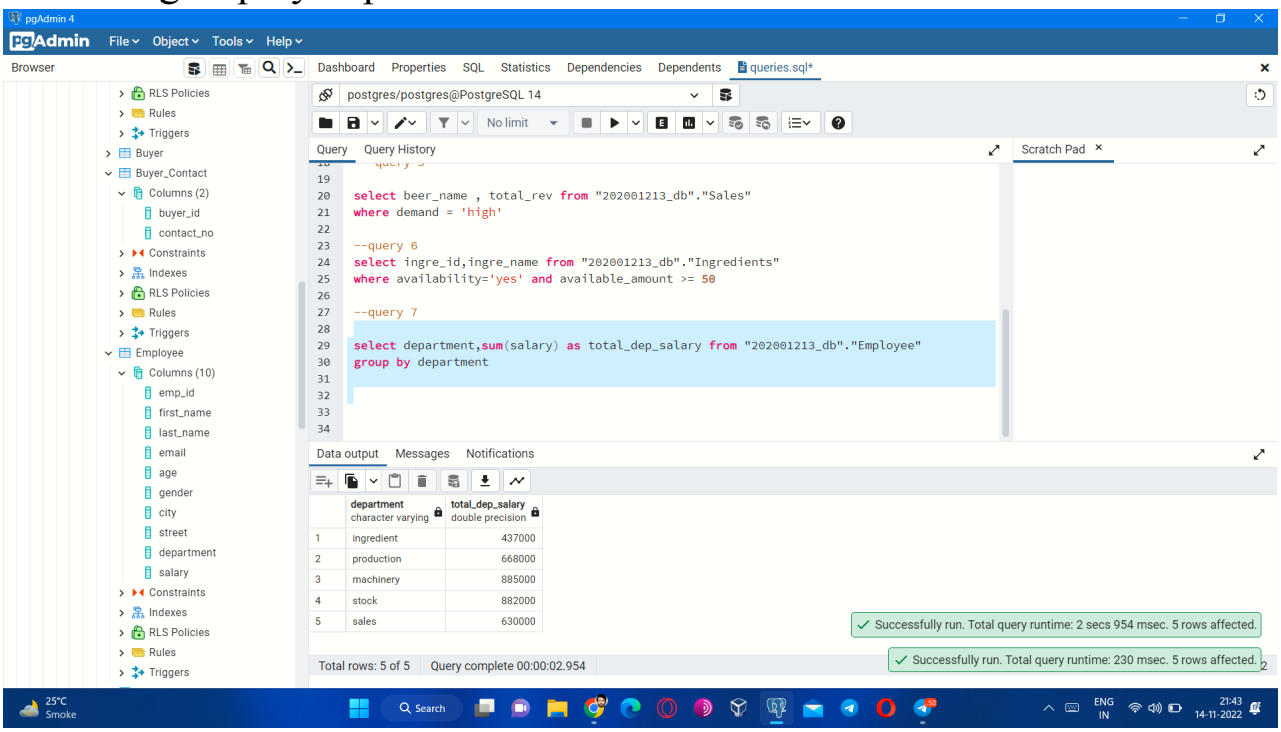
6. select id and name of ingredients which are available and available amount is more than 50 kgs.

Query: select ingre_id,ingre_name from "202001213_db"."Ingredients" where availability='yes' and available_amount >= 50



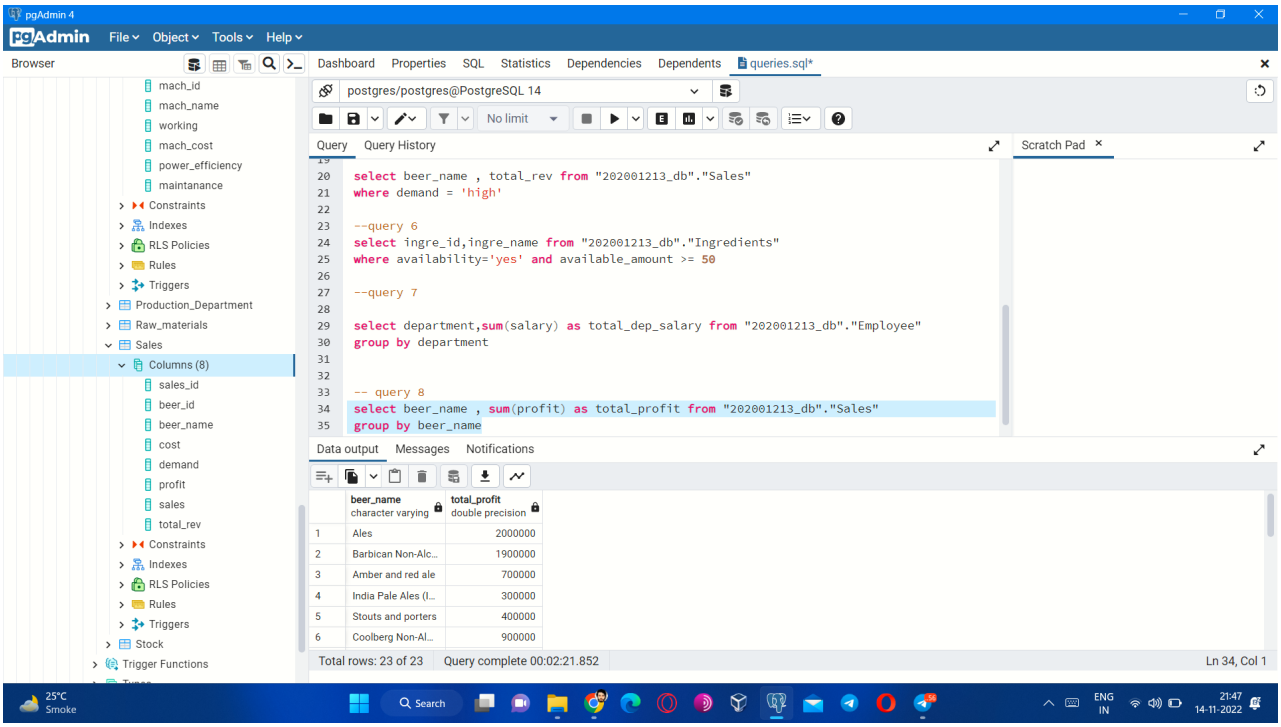
7. select sum of total salary that is provided to particular department

Query : select department,sum(salary) as total_dep_salary
from "202001213_db"."Employee"
group by department



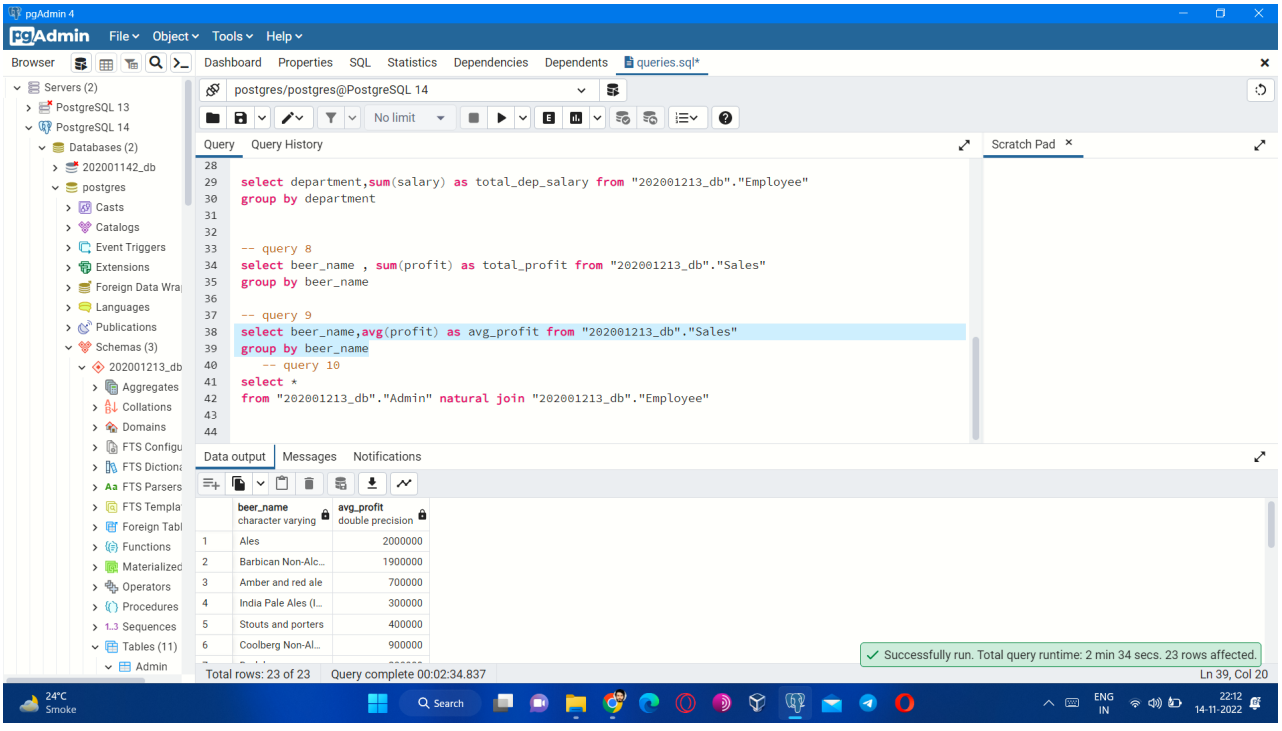
8. select beer name and total profit that is particular beer has made.

Query : select beer_name , sum(profit) as total_profit
from "202001213_db"."Sales"
group by beer_name



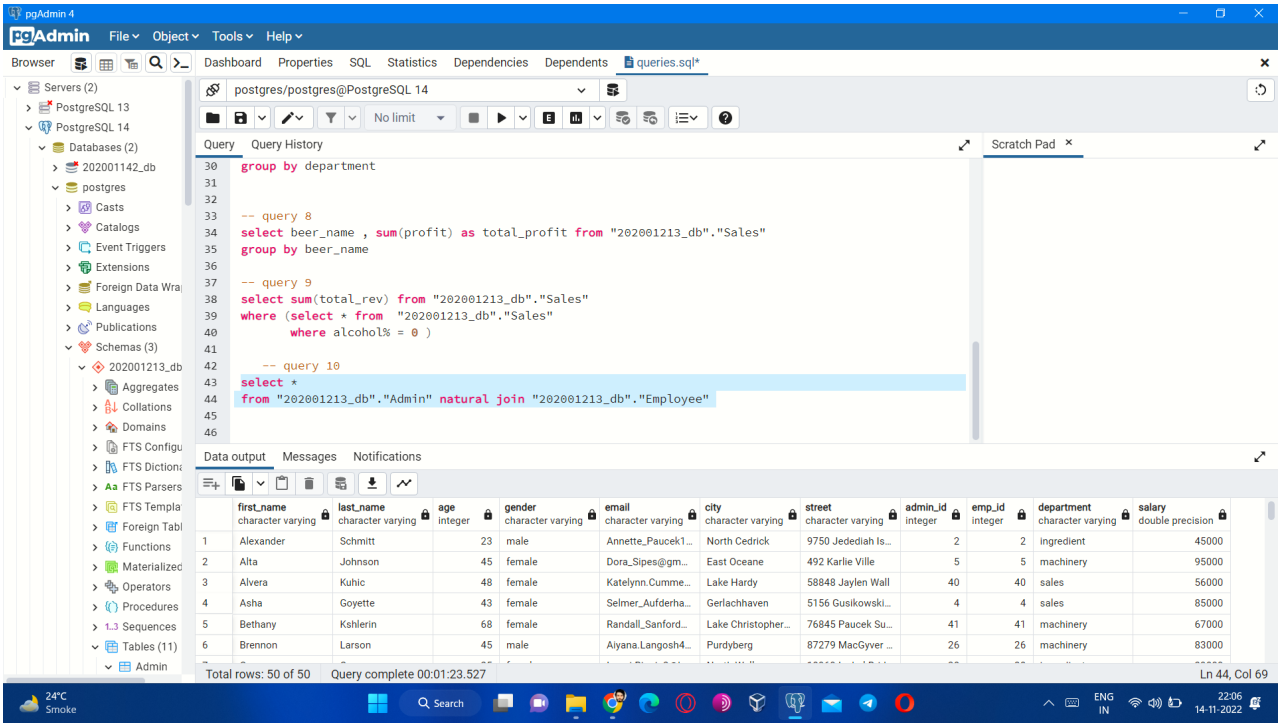
9. select beer name and average profit that is particular beer has made.

Query : select beer_name,avg(profit) as avg_profit
from "202001213_db"."Sales"
group by beer_name



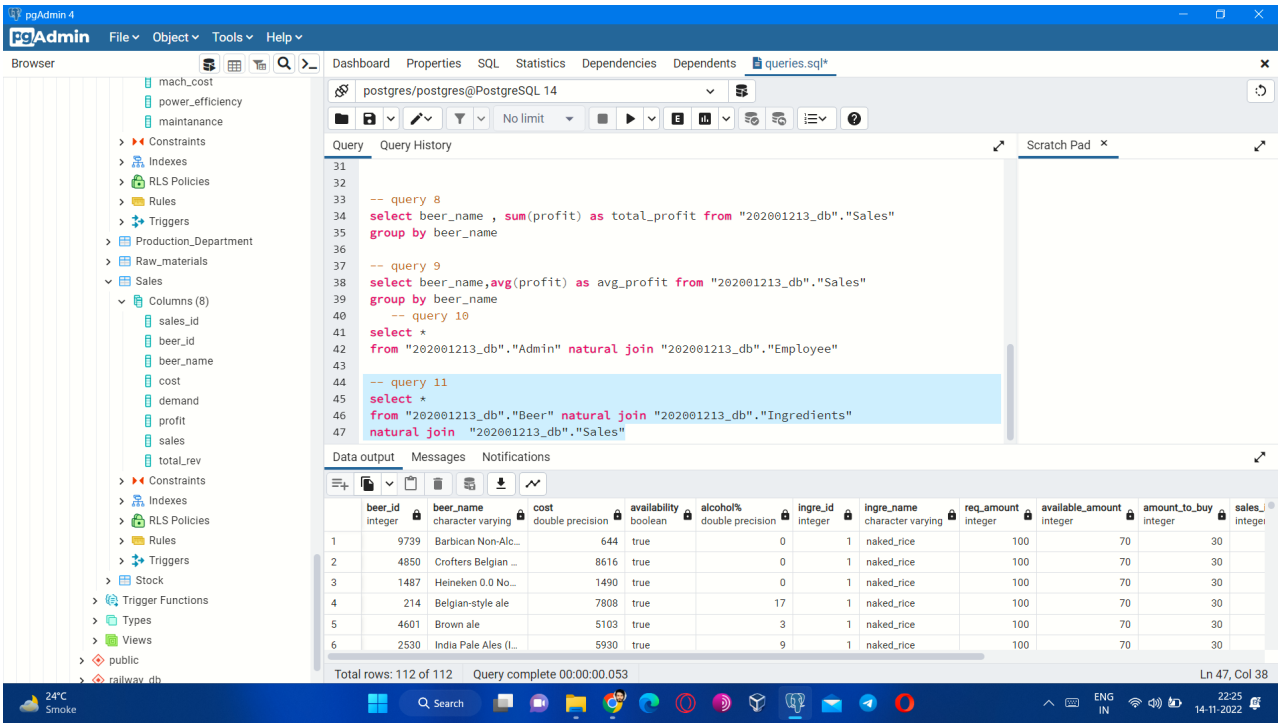
10. Do natural join of tables Admin and Employee

Query : select *
from "202001213_db"."Admin" natural join
"202001213_db"."Employee"



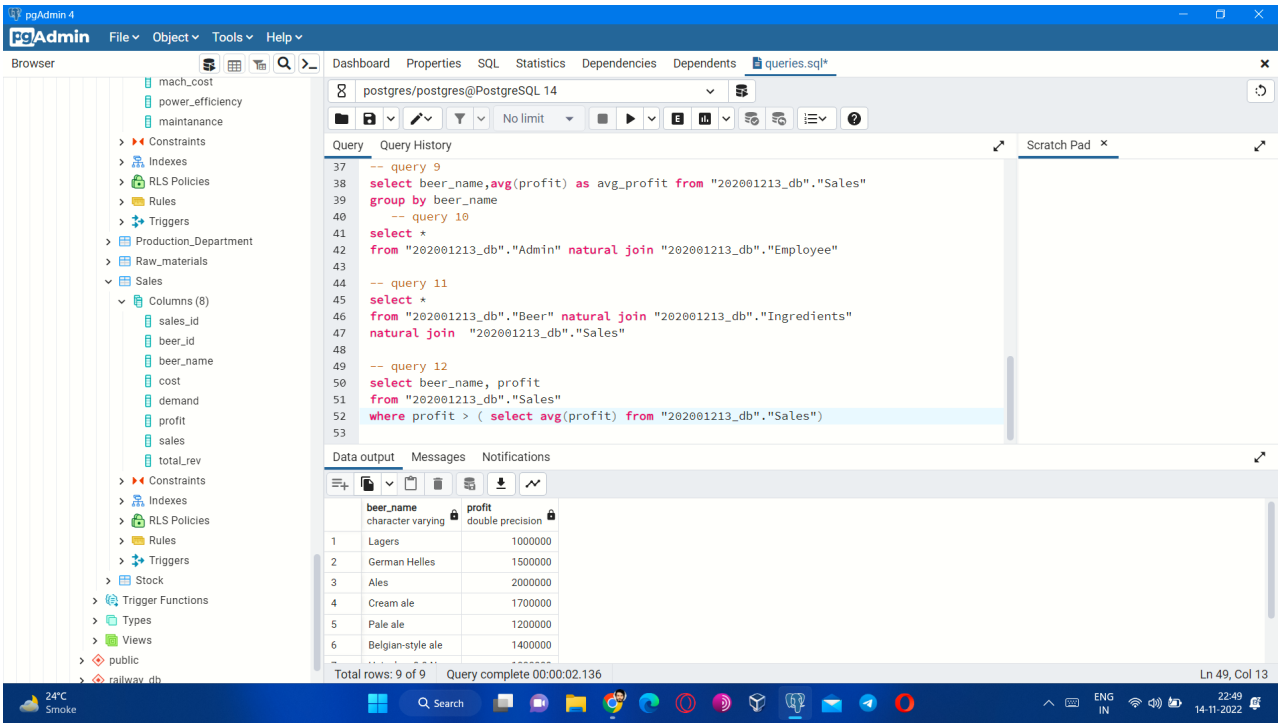
11 . Do natural join of tables Beer , Ingredients and Sales

Query : select *
from "202001213_db"."Beer" natural join
"202001213_db"."Ingredients"
natural join "202001213_db"."Sales"



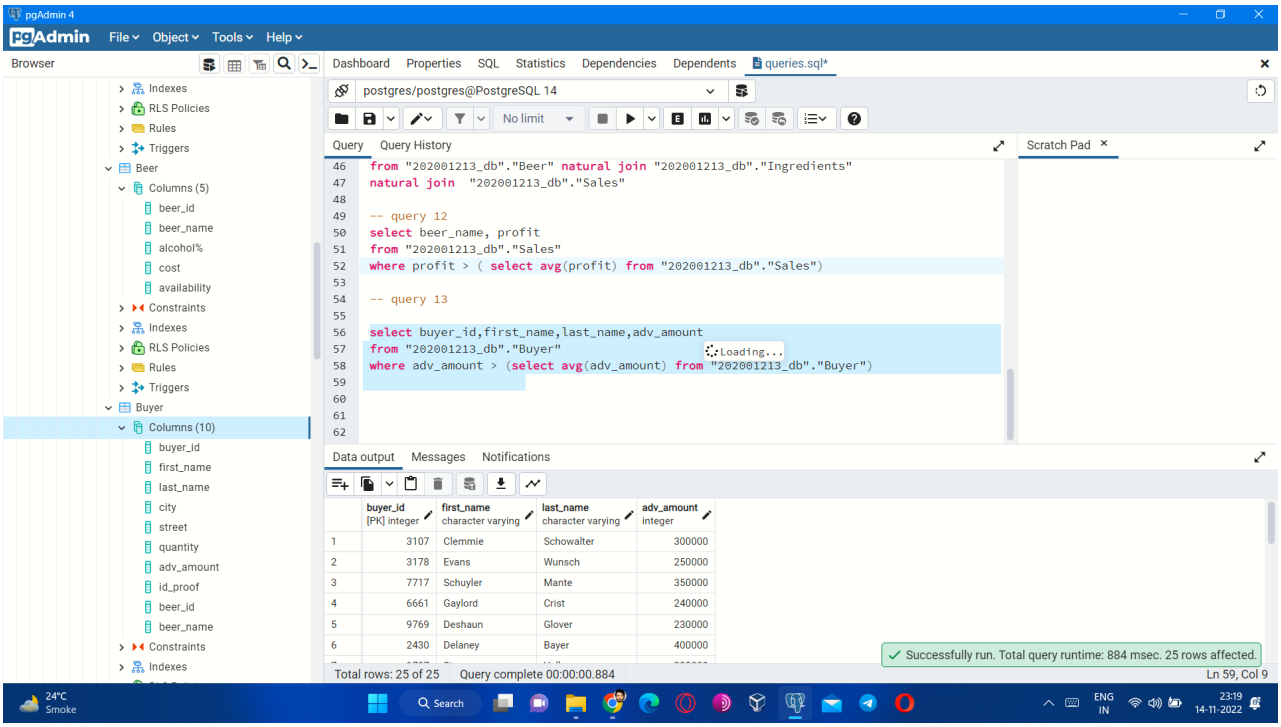
12 . select beer names whose profit is more than average profit of all beers.

Query : select beer_name, profit
from "202001213_db"."Sales"
where profit > (select avg(profit) from "202001213_db"."Sales")



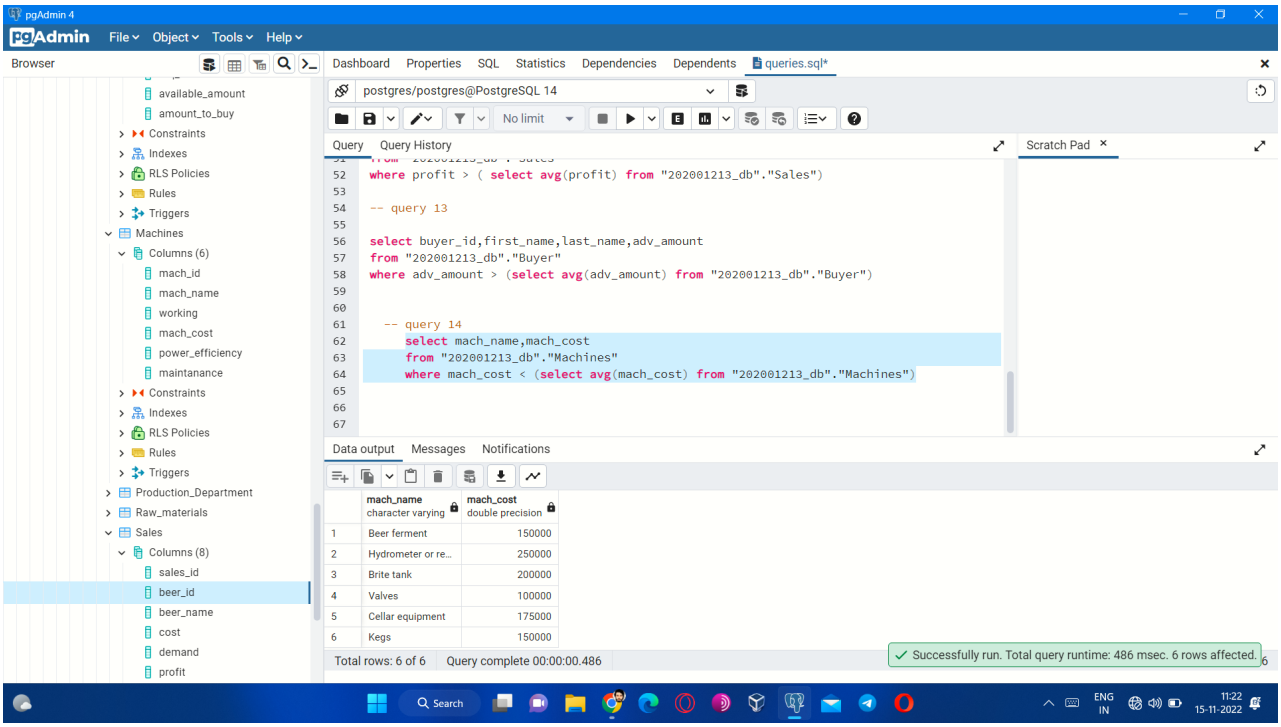
13. select buyer_id,first_name,last_name and advance amount of buyer who has paid advance amount more than average of advance amount of all buyers.

Query : select buyer_id,first_name,last_name,adv_amount
from "202001213_db"."Buyer"
where adv_amount > (select avg(adv_amount) from "202001213_db"."Buyer")



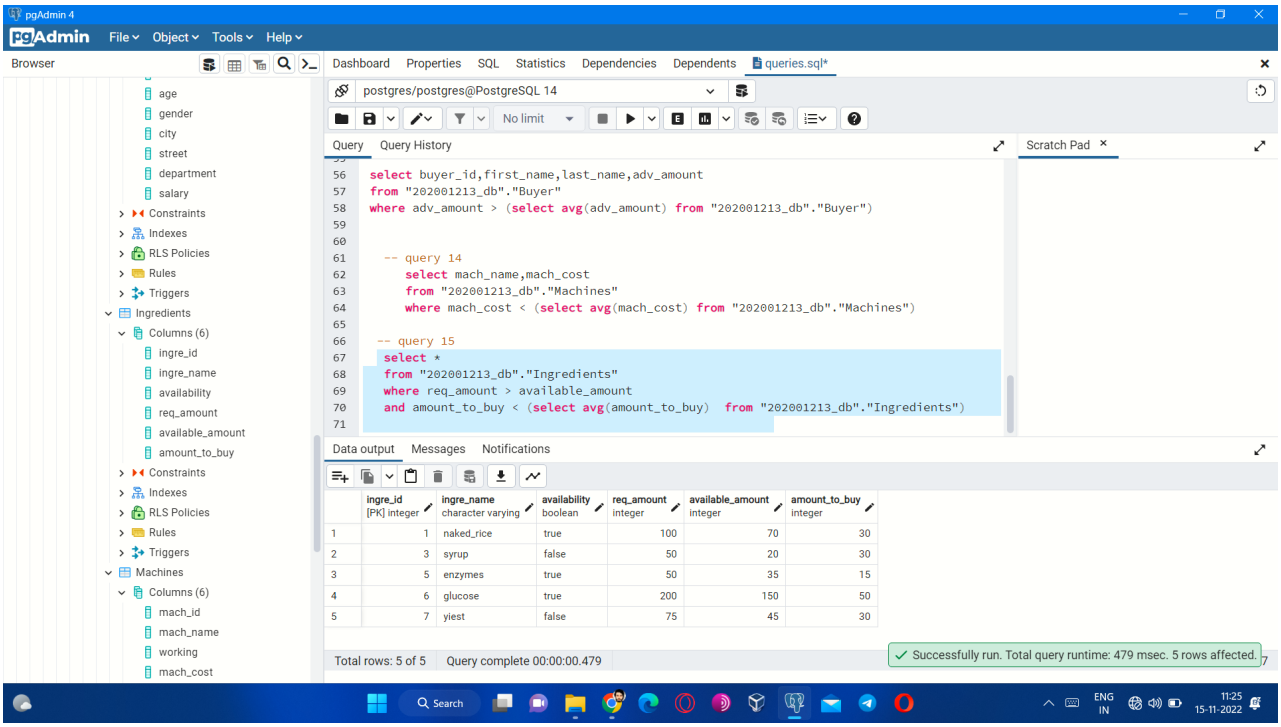
14. select machine name and machine cost of machines whose cost is less than average cost of all machines.

Query : select mach_name,mach_cost
from "202001213_db"."Machines"
where mach_cost < (select avg(mach_cost) from "202001213_db"."Machines")



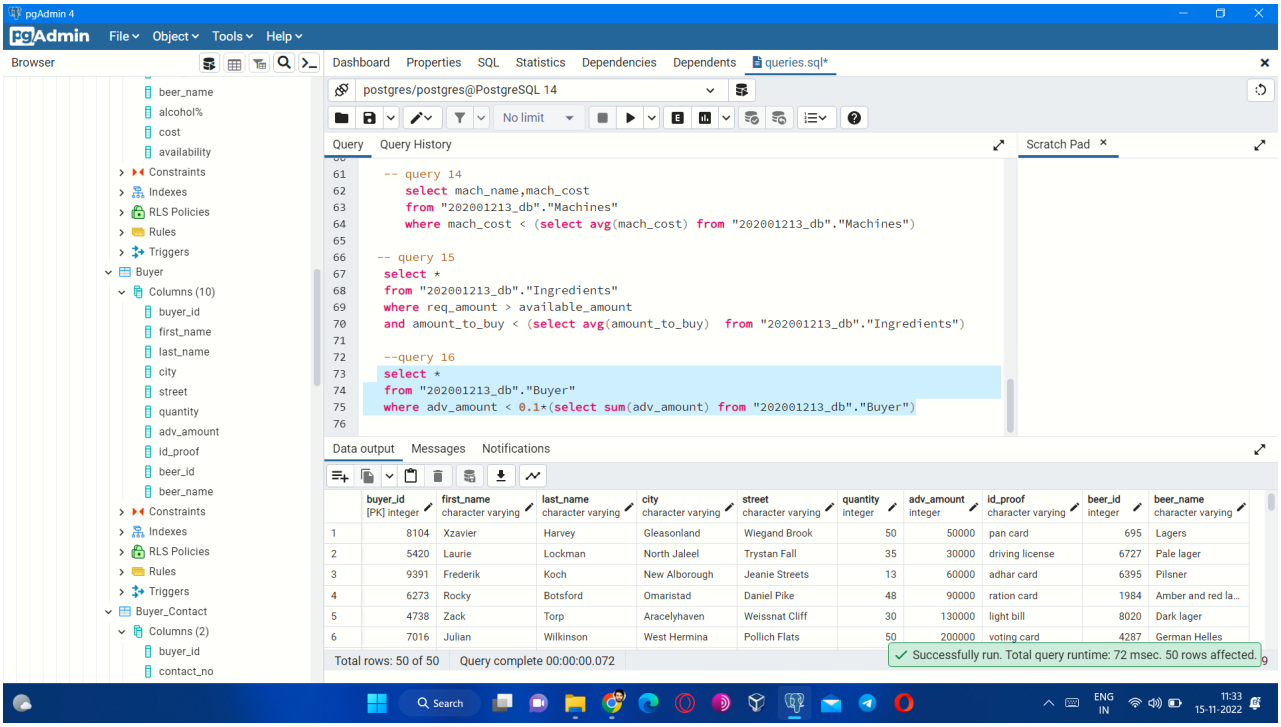
15. select ingredients whose required amount is greater than available amount and amount to be bought is less than average amount to be bought.

Query : select *
from "202001213_db"."Ingredients"
where req_amount > available_amount
and amount_to_buy < (select avg(amount_to_buy) from "202001213_db"."Ingredients")



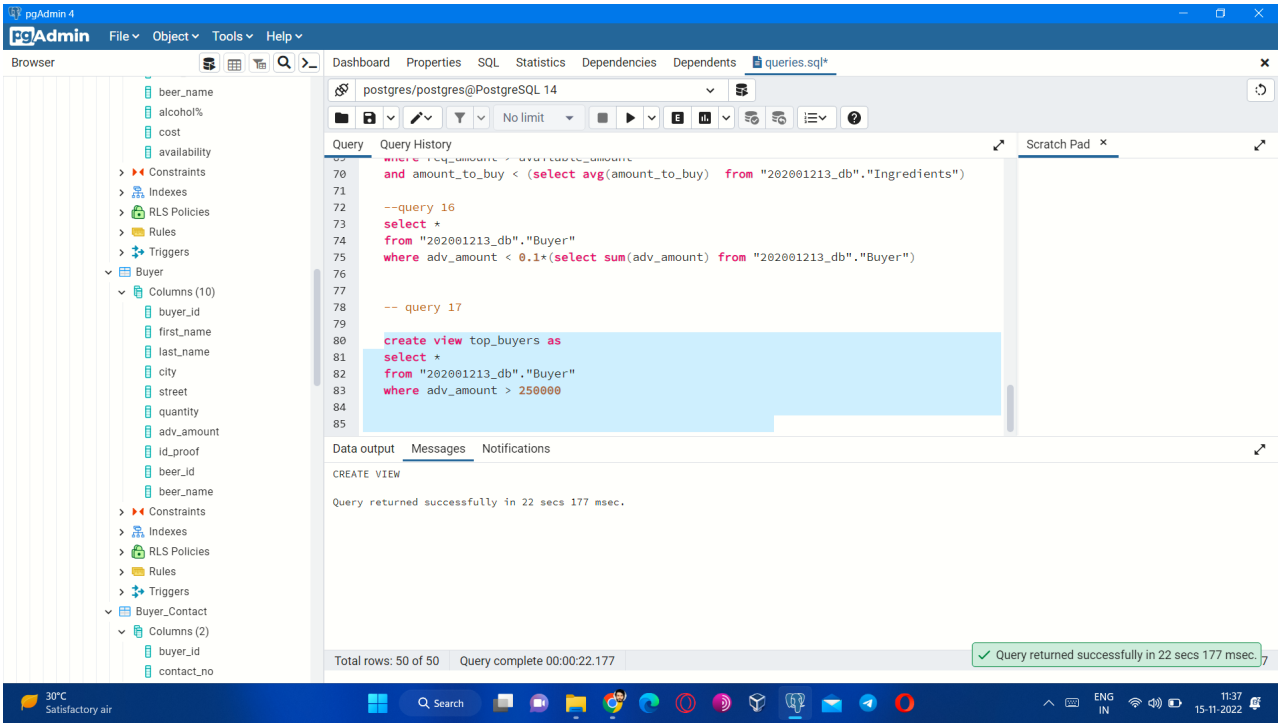
16. select buyer whose advance amount is less than 10 % of sum of all advance amount of all buyers.

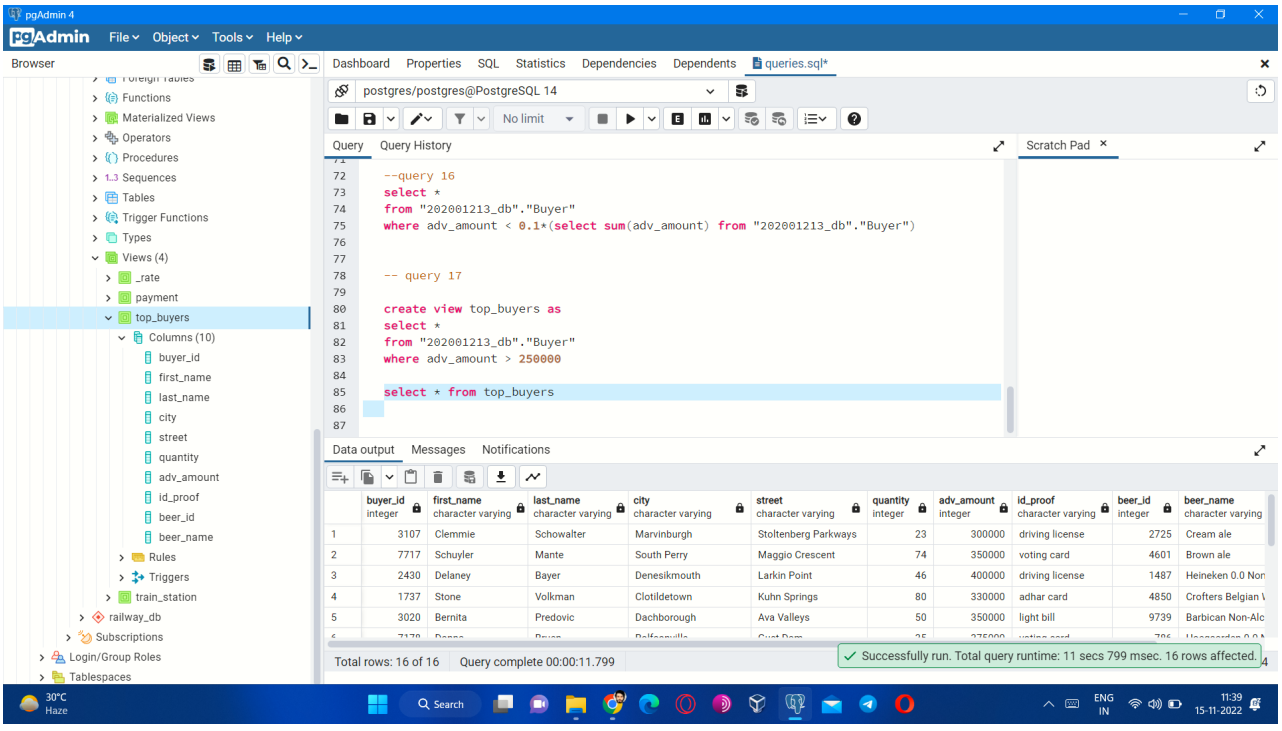
Query : select *
from "202001213_db"."Buyer"
where adv_amount < 0.1*(select sum(adv_amount) from "202001213_db"."Buyer")



17. create view top_buyers which will store only that buyers who has paid advance amount more than 250000.

Query : create view top_buyers as
select *
from "202001213_db"."Buyer"
where adv_amount > 250000

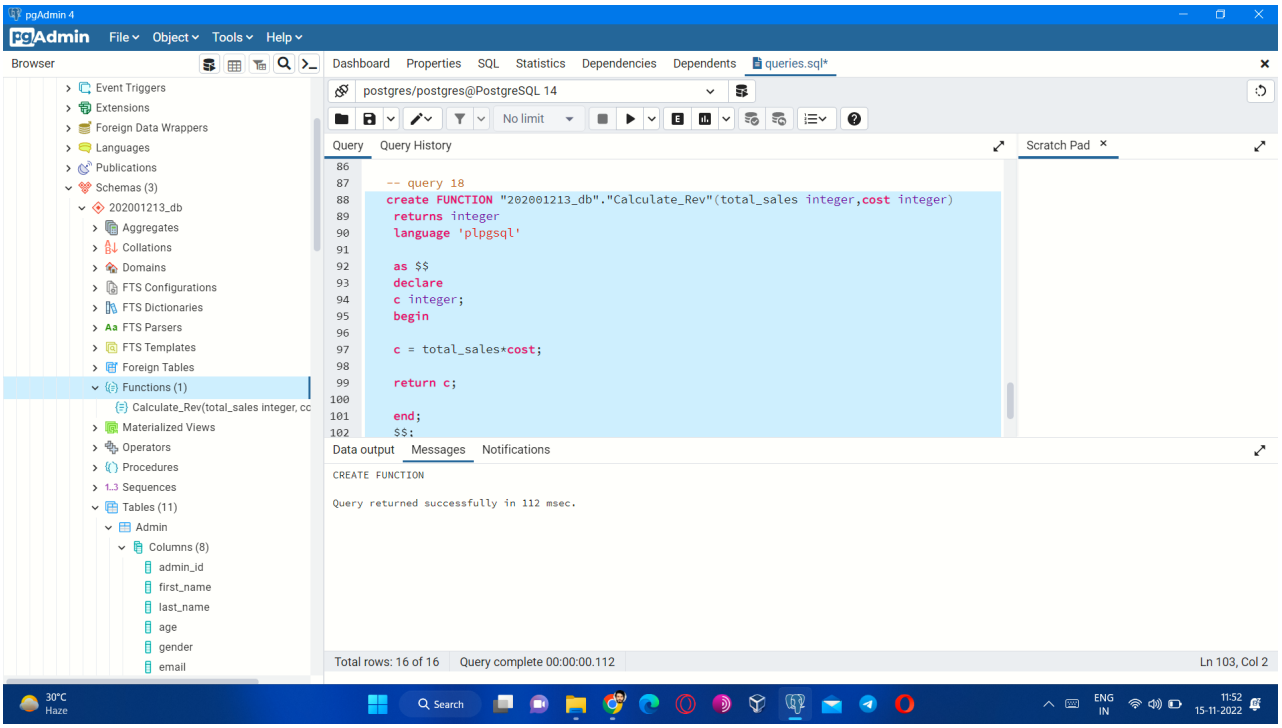




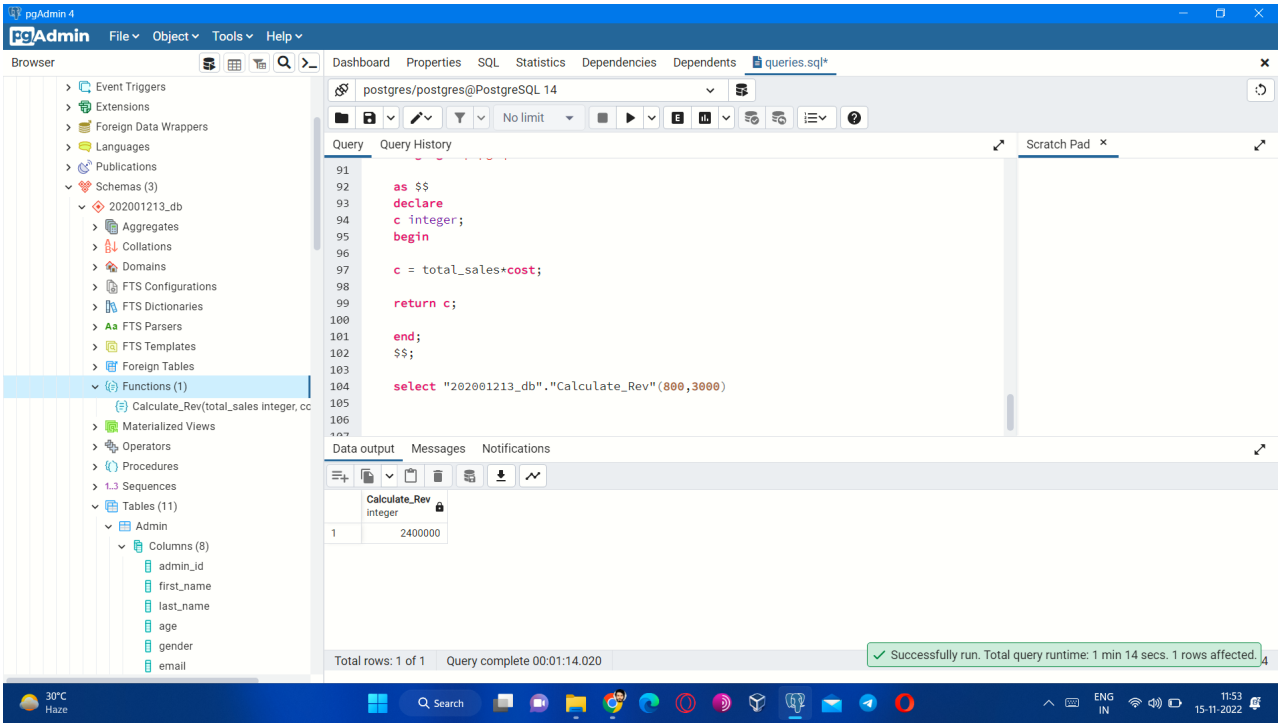
18. create function that will take two integers “beer sales” and “cost” as parameters and return total revenue generated by selling of that beer.

Query :

```
create FUNCTION   "202001213_db"."Calculate_Rev"(total_sales integer,cost integer)
    returns integer
        language 'plpgsql'

        as $$
        declare
        c integer;
        begin

        c = total_sales*cost;

        return c;

        end;
        $$;
```



select "202001213_db"."Calculate_Rev"(800,3000)

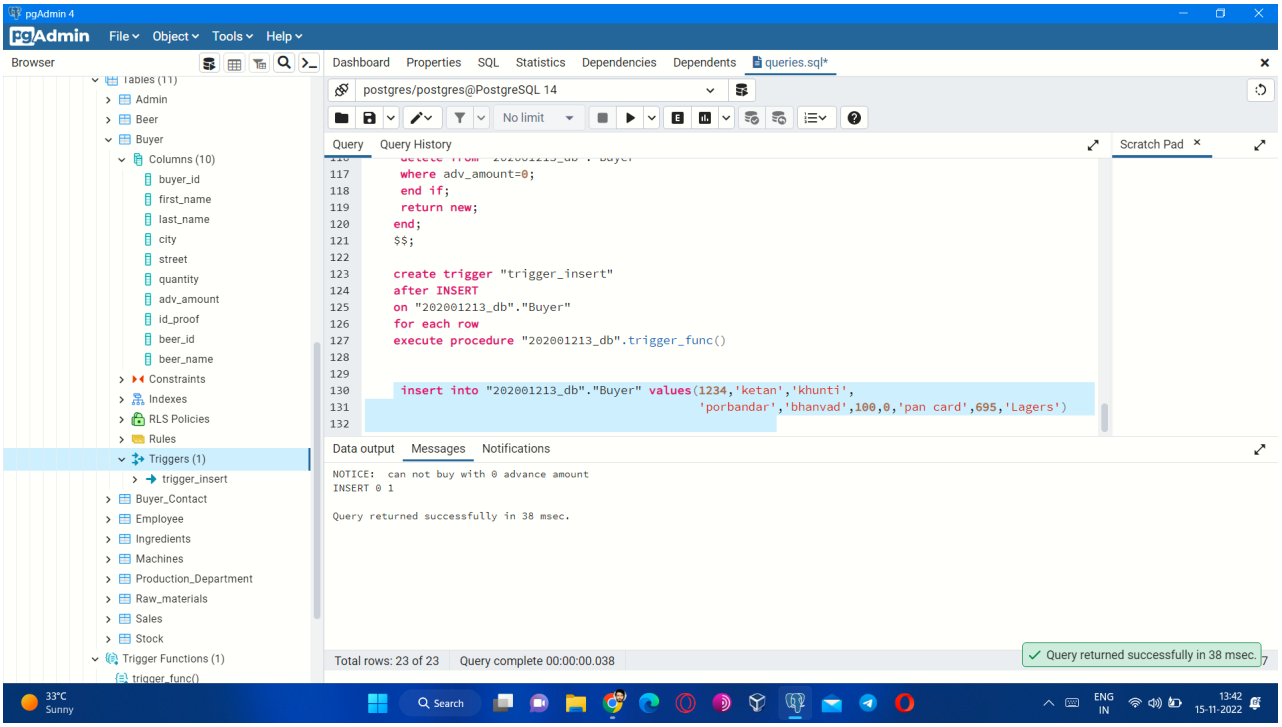


19. create trigger function that will check whether a buyer has paid advance amount or not , if not it will not allow entry of that buyer in Buyer table.

Query :

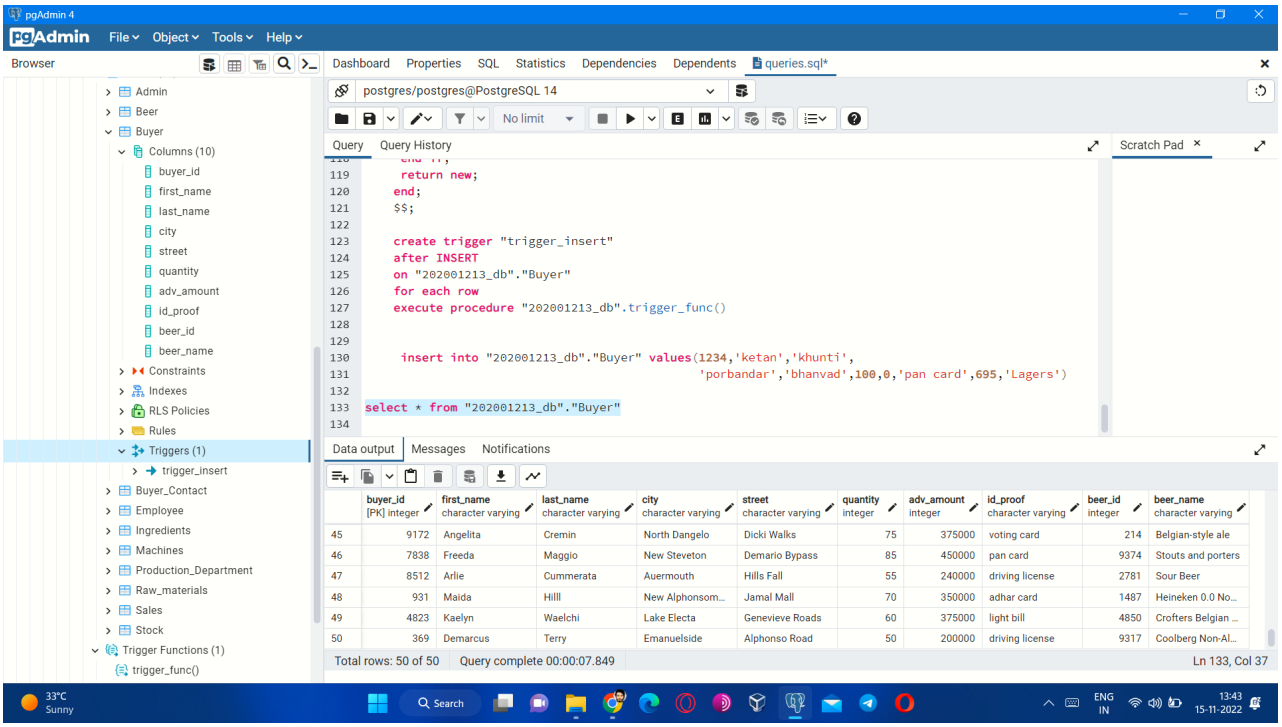
```
create or replace function "202001213_db".trigger_func()
        returns trigger
        language 'plpgsql'
        as $$
        BEGIN
         if new.adv_amount = 0
         THEN
         raise notice 'can not buy with 0 advance amount';
         delete from "202001213_db"."Buyer"
         where adv_amount=0;
         end if;
         return new;
        end;
        $$;
```

```
create trigger "trigger_insert"
after INSERT
on "202001213_db"."Buyer"
for each row
execute procedure "202001213_db".trigger_func()
```



insert into "202001213_db"."Buyer" values(1234,'ketan','khunti',

'porbandar','bhanvad',100,0,'pan card',695,'Lagers')



=> As we can see, buyer with buyer__id 1234 does not exists, as he did not pay advance amount.

20. create a trigger function that will check whether a beer with particular id and name exists or not. if exists it will not insert that data into “Beer” table.

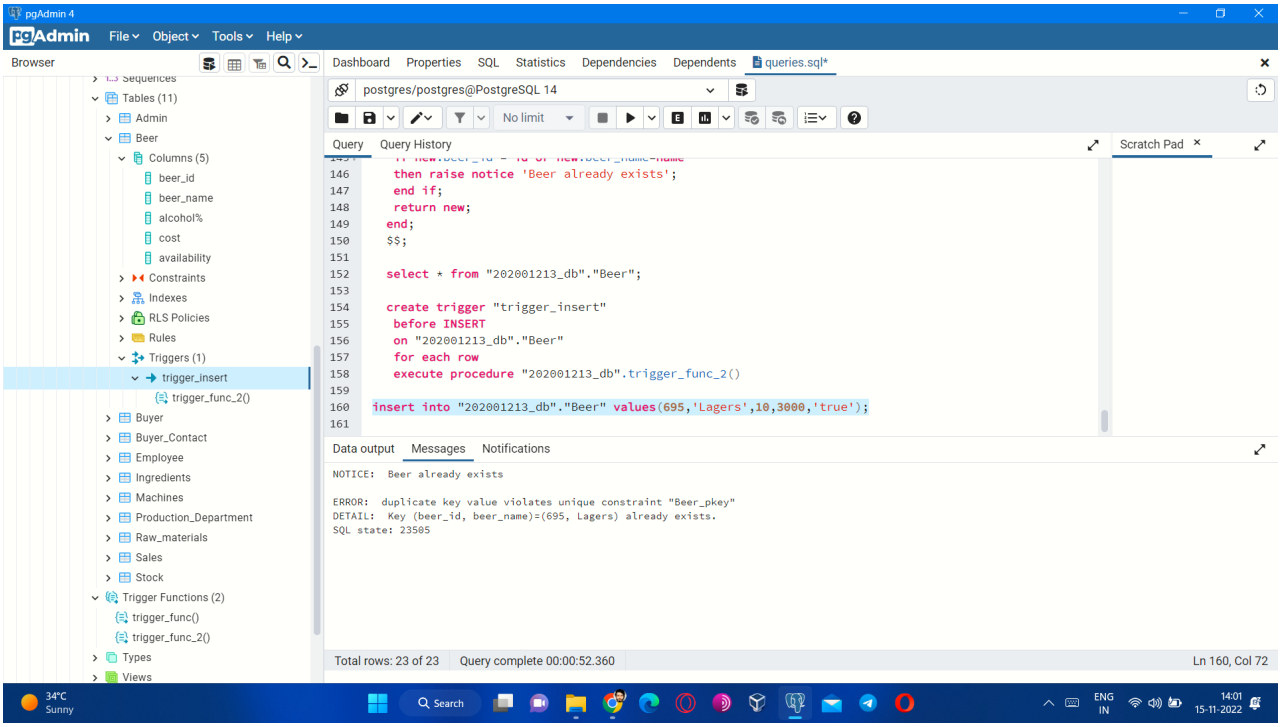
Query :

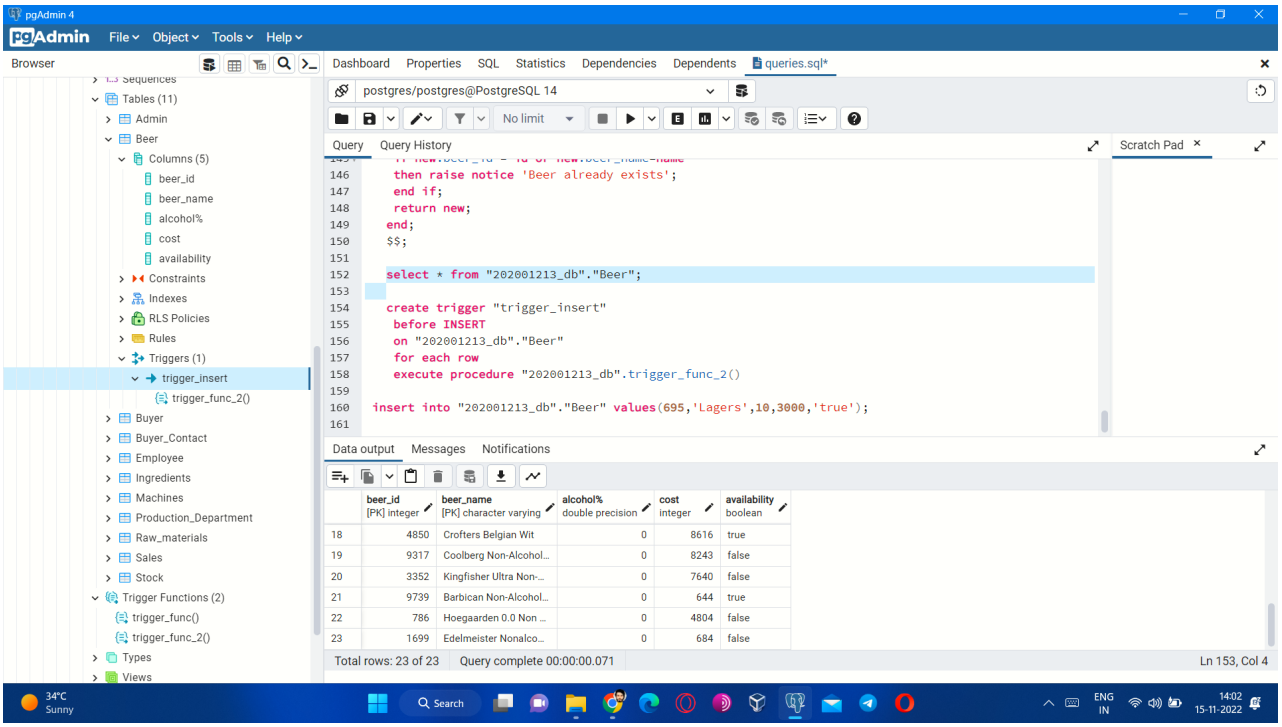
```
create or replace function "202001213_db".trigger_func_2()
  returns trigger
  language 'plpgsql'
  as $$
  declare id integer;
  declare name CHARACTER varying;
  begin
       select beer_id into id
       from "202001213_db"."Beer";
       if new.beer_id = id or new.beer_name=name
       then raise notice 'Beer already exists';
       end if;
       return new;
  end;
```

$$;

```
create trigger "trigger_insert"
    before INSERT
    on "202001213_db"."Beer"
    for each row
    execute procedure "202001213_db".trigger_func_2()
```

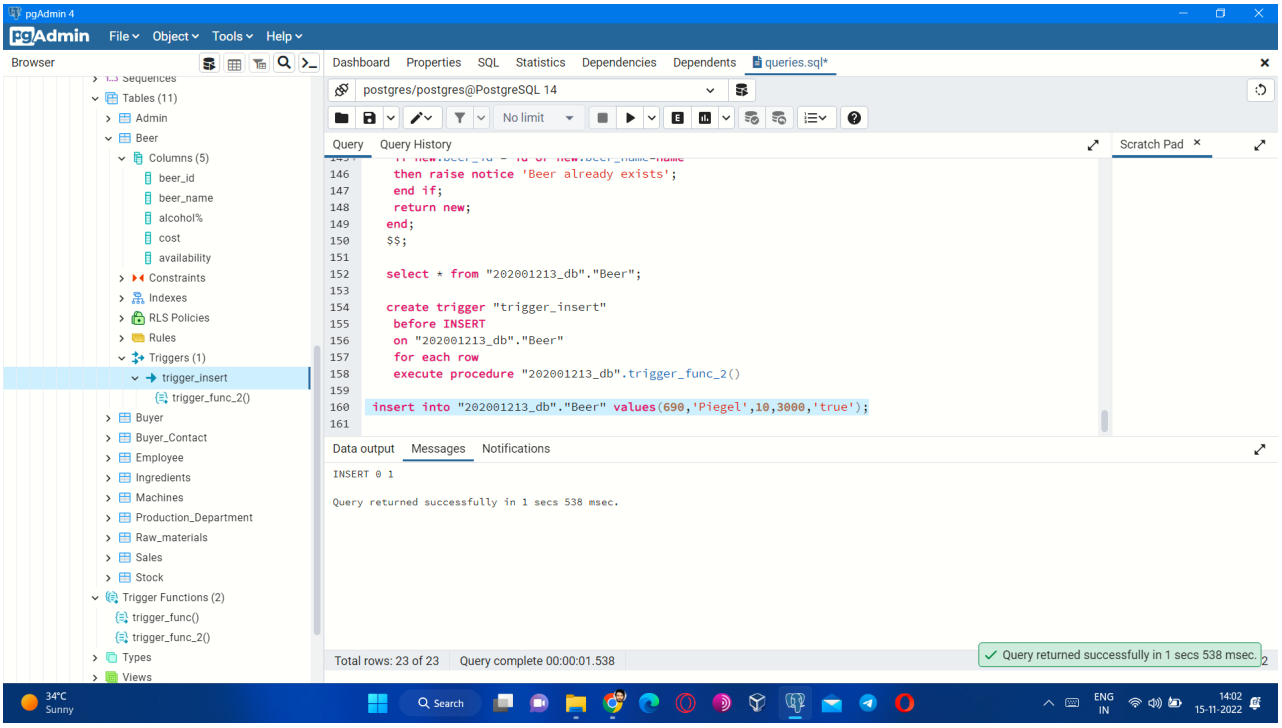
insert into "202001213_db"."Beer"
values(695,'Piegel',10,3000,'true');

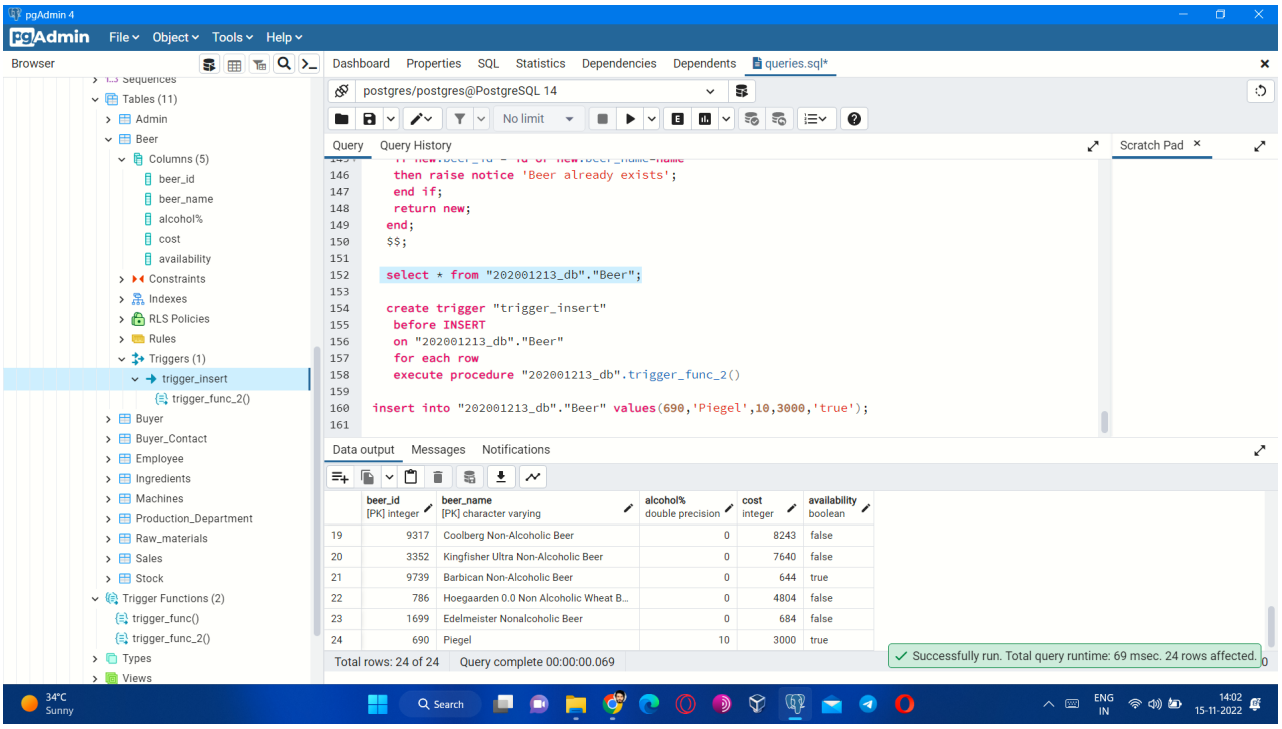




=> if we add beer with unique id and name then it will be inserted into beer table.

insert into "202001213_db"."Beer"
values(690,'Piegel',10,3000,'true');




pgAdmin 4
File
Object
Tools
Help
Browser
Dashboard
Properties
SQL
Statistics
Dependencies
Dependents
queries.sql*
postgres/postgres@PostgreSQL 14
No limit
Query
Query History
Scratch Pad
then raise notice 'Beer already exists';
end if;
return new;
end;
$$;
select * from "202001213_db"."Beer";
create trigger "trigger_insert"
before INSERT
on "202001213_db"."Beer"
for each row
execute procedure "202001213_db".trigger_func_2()
insert into "202001213_db"."Beer" values(690,'Piegel',10,3000,'true');
Data output
Messages
Notifications
beer_id [PK] integer
beer_name [PK] character varying
alcohol% double precision
cost integer
availability boolean
19 9317 Coolberg Non-Alcoholic Beer 0 8243 false
20 3352 Kingfisher Ultra Non-Alcoholic Beer 0 7640 false
21 9739 Barbican Non-Alcoholic Beer 0 644 true
22 786 Hoegaarden 0.0 Non Alcoholic Wheat B... 0 4804 false
23 1699 Edelmeister Nonalcoholic Beer 0 684 false
24 690 Piegel 10 3000 true
Total rows: 24 of 24
Query complete 00:00:00.069
Successfully run. Total query runtime: 69 msec. 24 rows affected.
Tables (11)
Admin
Beer
Columns (5)
beer_id
beer_name
alcohol%
cost
availability
Constraints
Indexes
RLS Policies
Rules
Triggers (1)
trigger_insert
trigger_func_2()
Buyer
Buyer_Contact
Employee
Ingredients
Machines
Production_Department
Raw_materials
Sales
Stock
Trigger Functions (2)
trigger_func()
trigger_func_2()
Types
Views
34°C
Sunny
Search
ENG
IN
14:02
15-11-2022