

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 \rightarrow (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,yeast,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 : buyer_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) \rightarrow (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_buy)

Primary key : ingre_id

Functional Dependencies : (ingre_id) \rightarrow (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) \rightarrow (ingre_name,availability,req_amount,amount_to_buy)

8. **machines**-(mach_id(**pk**),mach_name,type,working,mach_cost,power_eff,maintenance)

Primary key : mach_id

Functional dependencies : mach_id \rightarrow (mach_name,type,working,mach_cost,power_eff,maintenance)

mach_name \rightarrow 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 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 : **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,  
    yeast 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_b
uy)**

-- 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_amo
unt)
```

```
-- 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;

The screenshot shows the pgAdmin 4 interface with the 'Dependencies' tab selected. The left pane displays a tree view of database objects, including domains, FTS configurations, parsers, templates, foreign tables, functions, materialized views, operators, procedures, sequences, and tables. The 'Tables (11)' node is expanded, showing 11 tables: Admin, Beer, Buyer, Buyer_Contact, Employee, Ingredients, Machines, Production_Department, Raw_materials, Sales, and Stock. The right pane lists the dependencies for the 'Sales' table, which is highlighted in blue. The table has 11 entries:

Type	Name	Restriction
function	202001213_db.Calculate_Rev	normal
trigger_function	202001213_db.trigger_func	normal
trigger_function	202001213_db.trigger_func_2	normal
table	202001213_db.Admin	normal
table	202001213_db.Beer	normal
table	202001213_db.Buyer	normal
table	202001213_db.Buyer_Contact	normal
table	202001213_db.Employee	normal
table	202001213_db.Ingredients	normal
table	202001213_db.Machines	normal
table	202001213_db.Production_Department	normal
table	202001213_db.Raw_materials	normal
table	202001213_db.Sales	normal
table	202001213_db.Stock	normal

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

The screenshot shows the pgAdmin 4 interface. The left sidebar (Browser) lists various database objects: Domains, FTS Configurations, FTS Dictionaries, FTS Parsers, FTS Templates, Foreign Tables, Functions, Materialized Views, Operators, Procedures, Sequences, Tables (11), Trigger Functions, Types, Views, public, railway_db, Subscriptions, and Login/Group Roles. The main area shows a query window with the following SQL:

```
1 select * from "202001213_db"."Admin"
```

The results are displayed in a table titled "Data output". The columns are: admin_id [PK] integer, first_name character varying, last_name character varying, age integer, gender character varying, email character varying, city character varying, street character varying. The data consists of 6 rows:

admin_id	first_name	last_name	age	gender	email	city	street
1	Garland	Rice	29	male	Marcos_Schaden...	South Stefanie...	24324 White Spur
2	Alexander	Schmitt	23	male	Annette_Paucek1...	North Cedrick	9750 Jedediah Is...
3	Maurice	Hamill	35	female	Stephen_Baliste...	Glennieland	4258 Hilma Mou...
4	Asha	Goyette	43	female	Selmer_Aufderha...	Gerlachhaven	5156 Gusikowski...
5	Alta	Johnson	45	female	Dora_Sipes@gm...	East Oceane	492 Karlie Ville
6	Estel	Stedemann	67	male	Amir_Cremin@g...	Garettown	567 Webster Squ...

Total rows: 50 of 50 Query complete 00:00:00.267 Successfully run. Total query runtime: 267 msec. 50 rows affected.

2. Beer

Screenshot of pgAdmin 4 showing the results of a SQL query on the "Beer" table.

```
1 select * from "202001213_db"."Beer"
```

	beer_id	beer_name	alcohol%	cost	availability
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

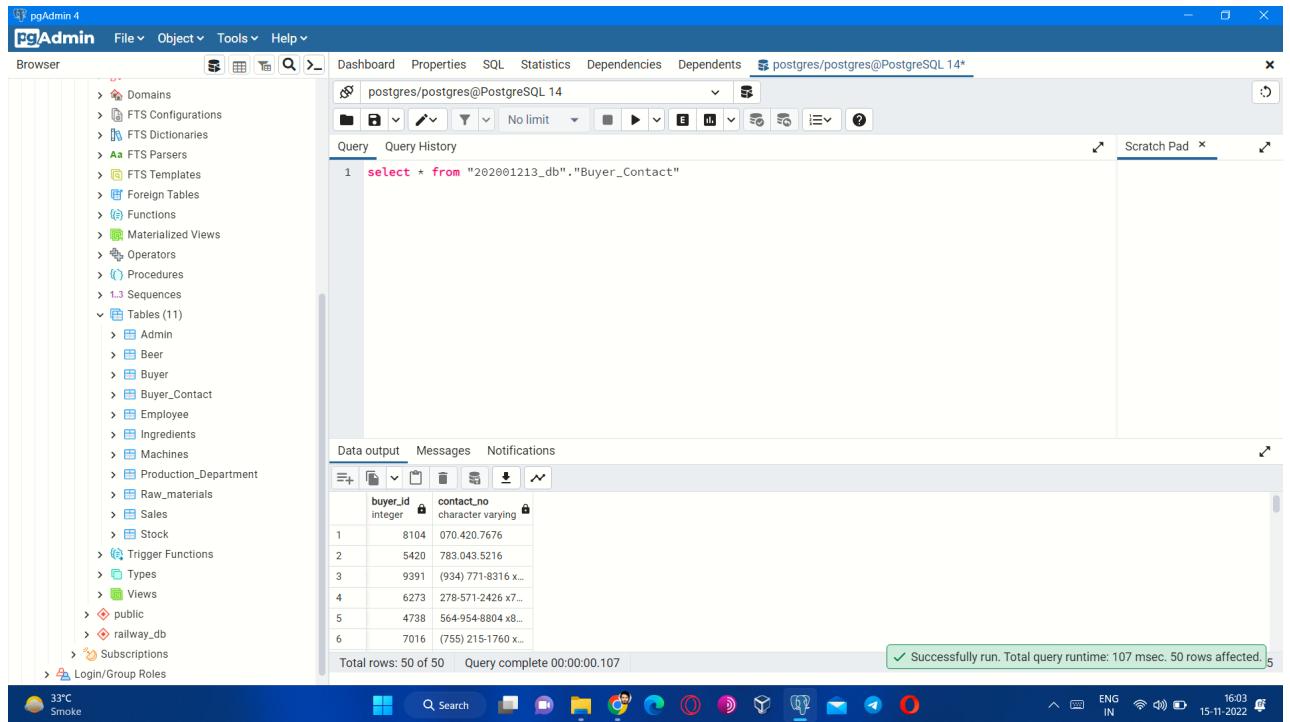
Screenshot of pgAdmin 4 showing the results of a SQL query on the "Buyer" table.

```
1 select * from "202001213_db"."Buyer"
```

	buyer_id	first_name	last_name	city	street	quantity	adv_amount	id_proof	beer_id	beer_name
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	Omaristed	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
6	7016	Julian	Williams	West Tremont	Darlene Flats	50	200000	water card	4287	German Helles

Total rows: 50 of 50 Query complete 00:00:00.060 ✓ Successfully run. Total query runtime: 60 msec. 50 rows affected.

4. Buyer_Contact



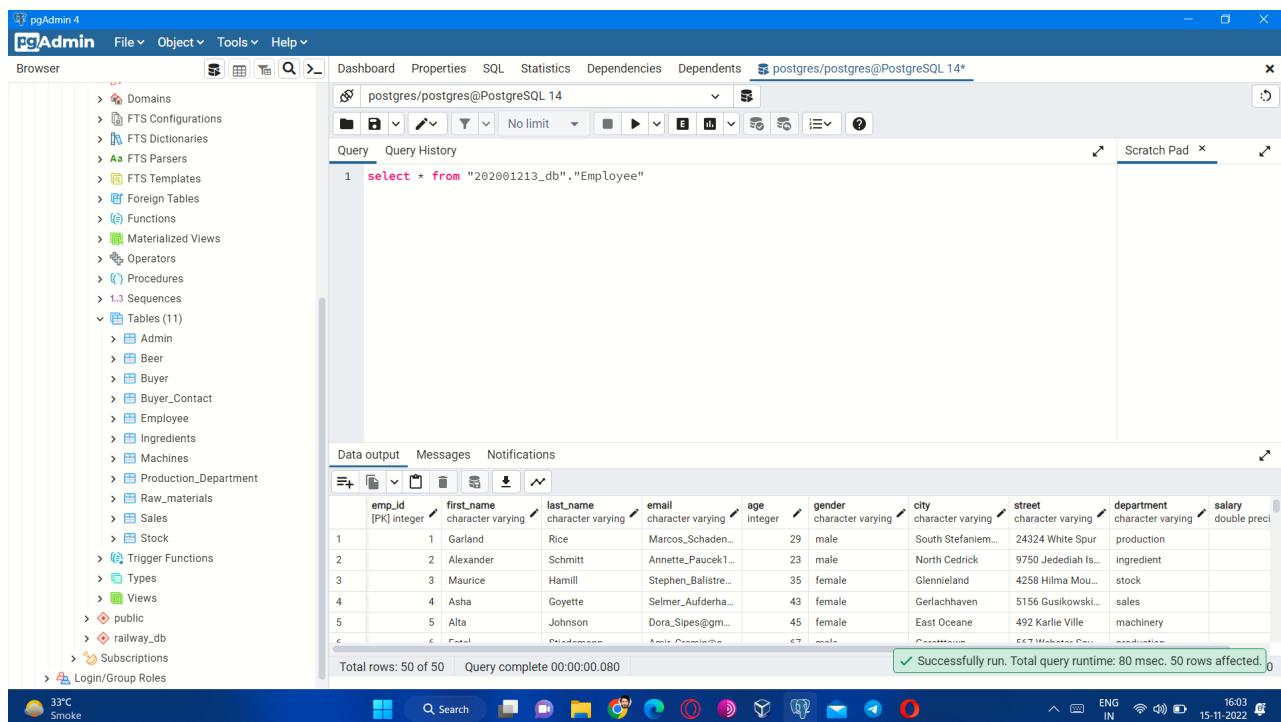
The screenshot shows the pgAdmin 4 interface with the following details:

- Toolbar:** File, Object, Tools, Help.
- Browser:** Shows the database schema with various objects like Domains, FTS Configurations, FTS Dictionaries, etc., and a list of Tables (11) including Admin, Beer, Buyer, Buyer_Contact, Employee, Ingredients, Machines, Production_Department, Raw_materials, Sales, Stock, Trigger Functions, Types, Views, public, railway_db, and Subscriptions.
- Query Editor:** Contains the SQL command: `select * from "202001213_db"."Buyer_Contact"`.
- Data Output:** Displays the results of the query in a tabular format. The columns are buyer_id (integer) and contact_no (character varying). The data consists of 6 rows:

buyer_id	contact_no
1	8104 070.420.7676
2	5420 783.043.5216
3	9391 (934) 771-8316 x...
4	6273 278.571-2426 x7...
5	4738 564-954-8804 x8...
6	7016 (755) 215-1760 x...

Total rows: 50 of 50 | Query complete 00:00:00.107 | Successfully run. Total query runtime: 107 msec. 50 rows affected.

5. Employee



The screenshot shows the pgAdmin 4 interface with the following details:

- Toolbar:** File, Object, Tools, Help.
- Browser:** Shows the database schema with various objects like Domains, FTS Configurations, FTS Dictionaries, etc., and a list of Tables (11) including Admin, Beer, Buyer, Buyer_Contact, Employee, Ingredients, Machines, Production_Department, Raw_materials, Sales, Stock, Trigger Functions, Types, Views, public, railway_db, and Subscriptions.
- Query Editor:** Contains the SQL command: `select * from "202001213_db"."Employee"`.
- Data Output:** Displays the results of the query in a tabular format. The columns are emp_id (PK integer), first_name (character varying), last_name (character varying), email (character varying), age (integer), gender (character varying), city (character varying), street (character varying), department (character varying), and salary (double precision). The data consists of 5 rows:

emp_id	first_name	last_name	email	age	gender	city	street	department	salary
1	Garland	Rice	Marcos.Schaden...	29	male	South Stefanie...	24324 White Spur	production	
2	Alexander	Schmitt	Annette_Paucek1...	23	male	North Cedrick	9750 Jeddiah Is...	ingredient	
3	Maurice	Hamill	Stephen_Ballstre...	35	female	Glenieland	4258 Hilma Mou...	stock	
4	Asha	Goyette	Selmer_Aufderha...	43	female	Gerlachhaven	5156 Gusikowski...	sales	
5	Alta	Johnson	Dora_Sipes@gm...	45	female	East Oceane	492 Karlie Ville	machinery	

Total rows: 50 of 50 | Query complete 00:00:00.080 | Successfully run. Total query runtime: 80 msec. 50 rows affected.

6. Ingredients

The screenshot shows the pgAdmin 4 interface for a PostgreSQL database. The left sidebar lists various database objects under 'Tables (11)'. The 'Ingredients' table is selected. The main pane displays the results of the SQL query: 'select * from "202001213_db"."Ingredients"'. The data output shows the following rows:

ingre_id	ingre_name	availability	req_amount	available_amount	amount_to_buy
1	naked_rice	true	100	70	30
2	oats	false	150	80	70
3	syrup	false	50	20	30
4	sugars	true	300	240	60
5	enzymes	true	50	35	15
6	glucose	true	200	150	50

Total rows: 10 of 10 | Query complete 00:00:10.07 | Successfully run. Total query runtime: 107 msec. 10 rows affected.

7. Machines

The screenshot shows the pgAdmin 4 interface for a PostgreSQL database. The left sidebar lists various database objects under 'Tables (11)'. The 'Machines' table is selected. The main pane displays the results of the SQL query: 'select * from "202001213_db"."Machines"'. The data output shows the following rows:

mach_id	mach_name	working	mach_cost	power_efficiency	maintenance
1	Malt mill	true	500000	3	routine checks
2	Mash Tun	false	300000	5	emergency repair
3	Filtration system	true	400000	4	routine checks
4	Heat exchanger	true	350000	3	routine checks
5	Beer ferment	true	150000	4	routine checks
6	Hydrometer or re...	true	250000	3	routine checks

Total rows: 13 of 13 | Query complete 00:00:00.064 | Successfully run. Total query runtime: 64 msec. 13 rows affected.

8. Production_Department

The screenshot shows the pgAdmin 4 interface with the following details:

- Toolbar:** File, Object, Tools, Help.
- Browser:** Shows a tree view of database objects including Domains, FTS Configurations, FTS Dictionaries, FTS Parsers, FTS Templates, Foreign Tables, Functions, Materialized Views, Operators, Procedures, Sequences, and Tables (11). The 'Tables (11)' node is expanded, showing Admin, Beer, Buyer, Buyer_Contact, Employee, Ingredients, Machines, Production_Department, Raw_materials, Sales, Stock, Trigger Functions, Types, Views, public, railway_db, and Subscriptions.
- Query Editor:** A tab labeled 'postgres/postgres@PostgreSQL 14*' contains the SQL query: `select * from "202001213_db"."Production_Department"`.
- Data Output:** A table titled 'Data output' displays the results of the query. The columns are production_id [PK] integer, req_amount integer, production_time character varying, beer_id integer, beer_name character varying, and cost double precision. The data consists of 6 rows:

production_id [PK]	req_amount	production_time	beer_id	beer_name	cost	
1	1	80	1 hour	695	Lagers	8160
2	2	100	1 hour	6727	Pale lager	1133
3	3	200	2 hours	6395	Pilsner	1979
4	4	200	2 hours	1984	Amber and red la...	4520
5	5	150	1.5 hours	8020	Dark lager	9782
6	6	340	3.4 hours	4287	German Helles	9886

- Messages:** Shows a success message: "Successfully run. Total query runtime: 93 msec. 50 rows affected."
- System Bar:** Shows system icons for weather (33°C), search, file, folder, browser, taskbar, notifications, and date/time (15-11-2022).

9. Row_Materials

The screenshot shows the pgAdmin 4 interface with the following details:

- Toolbar:** File, Object, Tools, Help.
- Browser:** Shows a tree view of database objects including Domains, FTS Configurations, FTS Dictionaries, FTS Parsers, FTS Templates, Foreign Tables, Functions, Materialized Views, Operators, Procedures, Sequences, and Tables (11). The 'Tables (11)' node is expanded, showing Admin, Beer, Buyer, Buyer_Contact, Employee, Ingredients, Machines, Production_Department, Raw_materials, Sales, Stock, Trigger Functions, Types, Views, public, railway_db, and Subscriptions.
- Query Editor:** A tab labeled 'postgres/postgres@PostgreSQL 14*' contains the SQL query: `select * from "202001213_db"."Raw_materials"`.
- Data Output:** A table titled 'Data output' displays the results of the query. The columns are beer_id [PK] integer, beer_name character varying, naked_rice boolean, oats boolean, syrup boolean, sugars boolean, enzymes boolean, glucose boolean, yeast boolean, calcium boolean, magnesium boolean, and b boolean. The data consists of 6 rows:

beer_id [PK]	beer_name	naked_rice	oats	syrup	sugars	enzymes	glucose	yeast	calcium	magnesium	b
1	695 Lagers	true	false	true	false	true	true	false	true	true	false
2	6727 Pale lager	false	false	true	false	false	false	true	false	true	false
3	6395 Pilsner	true	true	true	true	true	true	false	true	false	true
4	1984 Amber and red lager	false	false	true	true	false	true	false	false	true	false
5	8020 Dark lager	false	true	true	true	true	false	true	false	true	false
6	4287 German Helles	false	false	false	false	false	false	false	false	false	false

- Messages:** Shows a success message: "Successfully run. Total query runtime: 83 msec. 23 rows affected."
- System Bar:** Shows system icons for weather (33°C), search, file, folder, browser, taskbar, notifications, and date/time (15-11-2022).

10. Sales

The screenshot shows the pgAdmin 4 interface connected to a PostgreSQL 14 database. The left sidebar displays the database schema with various objects like Domains, FTS Configurations, and Tables (11). The main query editor window contains the SQL command: `select * from "202001213_db"."Sales"`. Below the query, the Data output tab shows the results of the query:

sales_id	beer_id	beer_name	cost	demand	profit	sales	total_rev
1	1	695 Lagers	8160	high	1000000	700	5712000
2	2	6727 Pale lager	1133	low	400000	200	226600
3	3	6395 Pilsner	1979	low	300000	200	395800
4	4	1984 Amber and red lager	4520	medium	700000	400	1808000
5	5	8020 Dark lager	9782	low	200000	140	1369480
6	6	4287 German Helles	9886	high	1500000	500	4943000

Total rows: 23 of 23 Query complete 00:00:00.098

Successfully run. Total query runtime: 98 msec. 23 rows affected.

11. Stock

The screenshot shows the pgAdmin 4 interface connected to a PostgreSQL 14 database. The left sidebar displays the database schema with various objects like Domains, FTS Configurations, and Tables (11). The main query editor window contains the SQL command: `select * from "202001213_db"."Stock"`. Below the query, the Data output tab shows the results of the query:

stock_id	stock	stock_req	prod_stock	adv_amount	beer_id	beer_name
1	100	80	20	50000	695	Lagers
2	150	100	0	30000	6727	Pale lager
3	340	200	0	60000	6395	Pilsner
4	170	200	30	90000	1984	Amber and red lager
5	200	150	0	130000	8020	Dark lager
6	570	340	0	200000	4287	German Helles

Total rows: 23 of 23 Query complete 00:00:00.083

Successfully run. Total query runtime: 83 msec. 23 rows affected.

20 Queries

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

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

The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' pane displays the database structure, including servers, databases, and tables. The '202001213_db' database is selected. In the center, the 'Query' pane contains the following SQL code:

```
-- query 1
select * from "202001213_db"."Buyer"
where adv_amount>100000;
```

Below the code, the 'Data output' pane shows the results of the query. The table has the following columns: buyer_id, first_name, last_name, city, street, quantity, adv_amount, id_proof, beer_id, and beer_name. The data consists of 12 rows. At the bottom of the results pane, it says 'Total rows: 38 of 38' and 'Query complete 00:00:00.277'. The status bar at the bottom right indicates 'Ln 3, Col 25'.

2. select beer whose price is more than 1000.

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

pgAdmin 4

File Object Tools Help

Browser

Dashboard Properties SQL Statistics Dependencies Dependents postgres/postgres@PostgreSQL 14*

Query Query History

```

1 -- query 1
2 select * from "202001213_db"."Buyer"
3 where adv_amount > 100000
4
5 -- query 2
6 SELECT beer_name from "202001213_db"."Beer"
7 where cost > 1000

```

Data output Messages Notifications

	beer_name
1	Lagers
2	Pale lager
3	Pilsner
4	Amber and red lager
5	Dark lager
6	German Helles
7	Ales
8	Cream ale
9	Pale ale
10	Amber and red ale
11	India Pale Ales (IPA)
12	Brown ale

Total rows: 21 of 21 Query complete 00:00:00.135 Ln 7, Col 19

25°C Smoke

Q Search

21:16 ENG IN 14-11-2022

3. select employee whose age is more than 40.

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

pgAdmin 4

File Object Tools Help

Browser

Dashboard Properties SQL Statistics Dependencies Dependents postgres/postgres@PostgreSQL 14*

Query Query History

```

1 -- query 1
2 select * from "202001213_db"."Buyer"
3 where adv_amount > 100000
4
5 -- query 2
6 SELECT beer_name from "202001213_db"."Beer"
7 where cost > 1000
8
9 --query 3
10 select * from "202001213_db"."Employee"
11 where age > 40;

```

Data output Messages Notifications

emp_id	first_name	last_name	email	age	gender	city	street	department	salary
1	4	Asha	Goyette	43	female	Gerlachhaven	5156 Gusikowski...	sales	850
2	5	Alta	Johnson	45	female	East Oceane	492 Karlie Ville	machinery	950
3	6	Estel	Stiedemann	67	male	Garetstown	567 Webster Squ...	production	900
4	8	Rubie	Lynch	48	female	Carmellatown	4887 Reinhold La...	stock	760
5	10	Joesh	Stanton	66	male	Alfredchester	79469 Ada Key	machinery	950
6	9	Dawn	Waelchi	71	male	Owendale	6651 Anna Island	sales	650

Total rows: 26 of 26 Query complete 00:00:00.520 Ln 11, Col 17

25°C Smoke

Q Search

21:17 ENG IN 14-11-2022

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

The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database schema with tables like Employee and Machines. The main area shows a query editor with the following SQL code:

```
--query 1
SELECT beer_name FROM "202001213_db"."Beer"
WHERE cost > 1000
--query 2
SELECT * FROM "202001213_db"."Employee"
WHERE age > 40;
--query 3
SELECT * FROM "202001213_db"."Machines"
WHERE maintanance='routine checks' AND power_efficiency >= 4
```

The results pane shows a table with the following data:

	mach_id	mach_name	working	mach_cost	power_efficiency	maintenance
1	3	Filtration system	true	400000	4	routine checks
2	5	Beer ferment	true	150000	4	routine checks
3	7	Brite tank	true	200000	5	routine checks
4	8	Pumps	true	350000	4	routine checks
5	12	Kegs	true	150000	4	routine checks
6	13	Packaging equip...	true	300000	4	routine checks

Total rows: 6 of 6 Query complete 00:00:01.324 Successfully run. Total query runtime: 1 secs 324 msec. 6 rows affected.

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'

```

pgAdmin 4
File Object Tools Help
Browser
  Machines
    Columns (6)
      mach_id
      mach_name
      working
      mach_cost
      power_efficiency
      maintenance
    Constraints
    Indexes
    RLS Policies
    Rules
    Triggers
  Production_Department
  Raw_materials
  Sales
    Columns (8)
      sales_id
      beer_id
      beer_name
      cost
      demand
      profit
      sales
      total_rev
    Constraints
    Indexes
    RLS Policies
    Rules
    Triggers
    Stock

```

Query History

```

9 --query 3
10 select * from "202001213_db"."Employee"
11 where age > 40;
12
13
14 --query 4
15 select * from "202001213_db"."Machines"
16 where maintanance='routine checks' and power_efficiency >= 4
17
18 -- query 5
19
20 select beer_name , total_rev from "202001213_db"."Sales"
21 where demand = 'high'
22
23
24
25

```

Data output

beer_name	total_rev
Lagers	5712000
German Helles	4943000
Ales	2501100
Cream ale	1969350
Belgian-style ale	4294400
Heineken 0.0 No...	1192000

Total rows: 7 of 7 Query complete 00:00:10.840 Successfully run. Total query runtime: 10 secs 840 msec. 7 rows affected.

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

```

pgAdmin 4
File Object Tools Help
Browser
  Ingredients
    Columns (6)
      ingre_id
      ingre_name
      availability
      req_amount
      available_amount
      amount_to_buy
    Constraints
    Indexes
    RLS Policies
    Rules
    Triggers
  Machines
    Columns (6)
      mach_id

```

Query History

```

11
12
13
14 --query 4
15 select * from "202001213_db"."Machines"
16 where maintanance='routine checks' and power_efficiency >= 4
17
18 -- query 5
19
20
21
22
23
24 select ingre_id,ingre_name from "202001213_db"."Ingredients"
25 where availability='yes' and available_amount >= 50
26
27

```

Data output

ingre_id	ingre_name
1	naked_rice
4	sugars
6	glucose
8	caclium
9	magnesium

Total rows: 5 of 5 Query complete 00:00:01.723 Ln 26, Col 2

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

The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database schema with tables like Buyer, Buyer_Contact, and Employee. The main area contains a query editor with the following SQL code:

```
19
20 select beer_name , total_rev from "202001213_db"."Sales"
21 where demand = 'high'
22
--query 6
23 select ingre_id,ingre_name from "202001213_db"."Ingredients"
24 where availability='yes' and available_amount >= 50
25
26
--query 7
27
28 select department,sum(salary) as total_dep_salary from "202001213_db"."Employee"
29 group by department
30
31
32
33
34
```

The results pane shows a table with the following data:

	department	total_dep_salary
1	ingredient	437000
2	production	668000
3	machinery	885000
4	stock	882000
5	sales	630000

Two green success messages are visible at the bottom right:

- Successfully run. Total query runtime: 2 secs 954 msec. 5 rows affected.
- Successfully run. Total query runtime: 230 msec. 5 rows affected.

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

pgAdmin 4

File Object Tools Help

Browser

```

    mach_id
    mach_name
    working
    mach_cost
    power_efficiency
    maintenance
  > Constraints
  > Indexes
  > RLS Policies
  > Rules
  > Triggers
  > Production_Department
  > Raw_materials
  > Sales
    > Columns (8)
      sales_id
      beer_id
      beer_name
      cost
      demand
      profit
      sales
      total_rev
  > Constraints
  > Indexes
  > RLS Policies
  > Rules
  > Triggers
  > Stock
  > Trigger Functions
  > Tunes

```

Dashboard Properties SQL Statistics Dependencies Dependents queries.sql*

Query History

```

19
20 select beer_name , total_rev from "202001213_db"."Sales"
21 where demand = 'high'
22
--query 6
23 select ingre_id,ingre_name from "202001213_db"."Ingredients"
24 where availability='yes' and available_amount >= 50
25
26
--query 7
27
28
29 select department,sum(salary) as total_dep_salary from "202001213_db"."Employee"
30 group by department
31
32
33 -- query 8
34 select beer_name , sum(profit) as total_profit from "202001213_db"."Sales"
35 group by beer_name

```

Data output Messages Notifications

beer_name	total_profit
Ales	2000000
Barbican Non-Alc...	1900000
Amber and red ale	700000
India Pale Ales (I...	300000
Stouts and porters	400000
Coolberg Non-Al...	900000

Total rows: 23 of 23 Query complete 00:02:21.852 Ln 34, Col 1

25°C Smoke

21:47 ENG IN 14-11-2022

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

pgAdmin 4

File Object Tools Help

Servers (2)

- PostgreSQL 13
- PostgreSQL 14
 - 202001142_db
 - postgres
 - Casts
 - Catalogs
 - Event Triggers
 - Extensions
 - Foreign Data Wrappers
 - Languages
 - Publications
 - Schemas (3)
 - 202001213_db
 - Aggregates
 - Collations
 - Domains
 - FTS Configurations
 - FTS Dictionaries
 - FTS Parsers
 - FTS Templates
 - Foreign Tables
 - Functions
 - Materialized Views
 - Operators
 - Procedures
 - Sequences
 - Tables (11)
 - Admin

Dashboard Properties SQL Statistics Dependencies Dependents queries.sql*

Query History

```

28
29 select department,sum(salary) as total_dep_salary from "202001213_db"."Employee"
30 group by department
31
32
33 -- query 8
34 select beer_name , sum(profit) as total_profit from "202001213_db"."Sales"
35 group by beer_name
36
37 -- query 9
38 select beer_name,avg(profit) as avg_profit from "202001213_db"."Sales"
39 group by beer_name
40
41 select *
42 from "202001213_db"."Admin" natural join "202001213_db"."Employee"
43
44

```

Data output Messages Notifications

beer_name	avg_profit
Ales	2000000
Barbican Non-Alc...	1900000
Amber and red ale	700000
India Pale Ales (I...	300000
Stouts and porters	400000
Coolberg Non-Al...	900000

Successfully run. Total query runtime: 2 min 34 secs. 23 rows affected.

Total rows: 23 of 23 Query complete 00:02:34.837 Ln 39, Col 20

24°C Smoke

22:12 ENG IN 14-11-2022

10. Do natural join of tables Admin and Employee

Query : select *

```
from "202001213_db"."Admin" natural join  
"202001213_db"."Employee"
```

The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database structure under 'Servers' and 'PostgreSQL 14'. The main area is a query editor with the following SQL code:

```
group by department
-- query 8
select beer_name , sum(profit) as total_profit from "202001213_db"."Sales"
group by beer_name
-- query 9
select sum(total_rev) from "202001213_db"."Sales"
where (select * from "202001213_db"."Sales"
       where alcohol% = 0 )
-- query 10
select *
from "202001213_db"."Admin" natural join "202001213_db"."Employee"
```

The results pane shows a table with columns: first_name, last_name, age, gender, email, city, street, admin_id, emp_id, department, and salary. The data consists of six rows:

	first_name	last_name	age	gender	email	city	street	admin_id	emp_id	department	salary
1	Alexander	Schmitt	23	male	Annette_Paucek1...	North Cedrick	9750 Jedediah Is...	2	2	ingredient	45000
2	Alta	Johnson	45	female	Dora_Sipes@gm...	East Oceane	492 Karlie Ville	5	5	machinery	95000
3	Alvera	Kuhic	48	female	Katelynn.Cumme...	Lake Hardy	58848 Jaylen Wall	40	40	sales	56000
4	Asha	Goyette	43	female	Selmer_Aufderha...	Gerlachhaven	5156 Guskowski...	4	4	sales	85000
5	Bethany	Kshlerin	68	female	Randall_Sanford...	Lake Christopher...	76845 Paucek Su...	41	41	machinery	67000
6	Brennon	Larson	45	male	Allyana.Langosh4...	Purdyberg	87279 MacGyver ...	26	26	machinery	83000

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"
```

```

31
32
33 -- query 8
34 select beer_name , sum(profit) as total_profit from "202001213_db"."Sales"
35 group by beer_name
36
37 -- query 9
38 select beer_name,avg(profit) as avg_profit from "202001213_db"."Sales"
39 group by beer_name
40   -- query 10
41 select *
42 from "202001213_db"."Admin" natural join "202001213_db"."Employee"
43
44 -- query 11
45 select *
46 from "202001213_db"."Beer" natural join "202001213_db"."Ingredients"
47 natural join "202001213_db"."Sales"

```

Total rows: 112 of 112 Query complete 00:00:00.053 Ln 47, Col 38

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")

```

Browser
  - mach_cost
  - power_efficiency
  - maintenance
  > Constraints
  > Indexes
  > RLS Policies
  > Rules
  > Triggers
  > Production_Department
  > Raw_materials
  > Sales
    > Columns (8)
      sales_id
      beer_id
      beer_name
      cost
      demand
      profit
      sales
      total_rev
    > Constraints
    > Indexes
    > RLS Policies
    > Rules
    > Triggers
    > Stock
    > Trigger Functions
    > Types
    > Views
    > public
    > railway.db

Dashboard Properties SQL Statistics Dependencies Dependents queries.sql*
postres/postgres@PostgreSQL 14
No limit ▾
Scratch Pad ×

Query History
-- query 9
select beer_name,avg(profit) as avg_profit from "202001213_db"."Sales"
group by beer_name
-- query 10
select *
from "202001213_db"."Admin" natural join "202001213_db"."Employee"
-- query 11
select *
from "202001213_db"."Beer" natural join "202001213_db"."Ingredients"
natural join "202001213_db"."Sales"
-- query 12
select beer_name, profit
from "202001213_db"."Sales"
where profit > ( select avg(profit) from "202001213_db"."Sales")
-- query 13
-- query 14
-- query 15
-- query 16
-- query 17
-- query 18
-- query 19
-- query 20
-- query 21
-- query 22
-- query 23
-- query 24
-- query 25
-- query 26
-- query 27
-- query 28
-- query 29
-- query 30
-- query 31
-- query 32
-- query 33
-- query 34
-- query 35
-- query 36
-- query 37
-- query 38
-- query 39
-- query 40
-- query 41
-- query 42
-- query 43
-- query 44
-- query 45
-- query 46
-- query 47
-- query 48
-- query 49
-- query 50
-- query 51
-- query 52
-- query 53

Data output Messages Notifications
beer_name profit
character varying double precision
1 Lagers 1000000
2 German Helles 1500000
3 Ales 2000000
4 Cream ale 1700000
5 Pale ale 1200000
6 Belgian-style ale 1400000
Total rows: 9 of 9 Query complete 00:00:02.136
Ln 49, Col 13

```

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")

```

46   from "202001213_db"."Beer" natural join "202001213_db"."Ingredients"
47   natural join "202001213_db"."Sales"
48
49 -- query 12
50 select beer_name, profit
51 from "202001213_db"."Sales"
52 where profit > ( select avg(profit) from "202001213_db"."Sales")
53
54 -- query 13
55
56 select buyer_id, first_name, last_name, adv_amount
57 from "202001213_db"."Buyer"
58 where adv_amount > (select avg(adv_amount) from "202001213_db"."Buyer")
59
60
61
62

```

buyer_id	first_name	last_name	adv_amount
1	3107	Clemmie	Schowalter
2	3178	Evans	Wunsch
3	7717	Schuylar	Mante
4	6661	Gaylord	Crist
5	9769	Deshawn	Glover
6	2430	Delaney	Bayer

Total rows: 25 of 25 Query complete 00:00:00.884

Successfully run. Total query runtime: 884 msec. 25 rows affected.

Ln 59, Col 9

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")

The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' panel displays a tree view of database objects for the 'postgres' schema in 'PostgreSQL 14'. The 'Sales' table under the 'Production_Department' schema is currently selected. The main area contains a 'Query' window with the following SQL code:

```
202001213_db . sales
where profit > ( select avg(profit) from "202001213_db"."Sales")
-- query 13
select buyer_id,first_name,last_name,adv_amount
from "202001213_db"."Buyer"
where adv_amount > (select avg(adv_amount) from "202001213_db"."Buyer")
-- query 14
select mach_name,mach_cost
from "202001213_db"."Machines"
where mach_cost < (select avg(mach_cost) from "202001213_db"."Machines")
```

The 'Data output' tab is active, showing the results of the last query:

	mach_name	mach_cost
1	Beer ferment	150000
2	Hydrometer or re...	250000
3	Brile tank	200000
4	Valves	100000
5	Cellar equipment	175000
6	Kegs	150000

At the bottom, a message indicates: 'Successfully run. Total query runtime: 486 msec. 6 rows affected.'

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")
```

```

56 select buyer_id,first_name,last_name,adv_amount
57 from "202001213_db"."Buyer"
58 where adv_amount > (select avg(adv_amount) from "202001213_db"."Buyer")
59
60 -- query 14
61 select mach_name,mach_cost
62 from "202001213_db"."Machines"
63 where mach_cost < (select avg(mach_cost) from "202001213_db"."Machines")
64
65 -- query 15
66 select *
67 from "202001213_db"."Ingredients"
68 where req_amount > available_amount
69 and amount_to_buy < (select avg(amount_to_buy) from "202001213_db"."Ingredients")
70
71

```

ingre_id [PK] integer	ingre_name character varying	availability boolean	req_amount integer	available_amount integer	amount_to_buy integer
1	naked_rice	true	100	70	30
2	syrup	false	50	20	30
3	enzymes	true	50	35	15
4	glucose	true	200	150	50
5	yest	false	75	45	30

Total rows: 5 of 5 Query complete 00:00:00.479 Successfully run. Total query runtime: 479 msec. 5 rows affected.

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")

```

61 -- query 14
62 select mach_name,mach_cost
63 from "202001213_db"."Machines"
64 where mach_cost < (select avg(mach_cost) from "202001213_db"."Machines")
65
66 -- query 15
67 select *
68 from "202001213_db"."Ingredients"
69 where req_amount > available_amount
70 and amount_to_buy < (select avg(amount_to_buy) from "202001213_db"."Ingredients")
71
72 --query 16
73 select *
74 from "202001213_db"."Buyer"
75 where adv_amount < 0.1*(select sum(adv_amount) from "202001213_db"."Buyer")
76

```

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	Xzavier	Harvey	Gleasonland	Wiegand Brook	50	50000	pan card	695	Lagers
2	Laurie	Lockman	North Jaleel	Trystan Fall	35	30000	driving license	6727	Pale lager
3	Frederik	Koch	New Alborough	Jeanie Streets	13	60000	adhar card	6395	Pilsner
4	Rocky	Botsford	Omaristed	Daniel Pike	48	90000	ration card	1984	Amber and red la...
5	Zack	Torp	Aracelyhaven	Weissnat Cliff	30	130000	light bill	8020	Dark lager
6	Julian	Wilkinson	West Hermina	Pollrich Flatts	50	200000	voting card	4287	German Helles

Total rows: 50 of 50 Query complete 00:00:00.072 Successfully run. Total query runtime: 72 msec. 50 rows affected.

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

The screenshot shows the pgAdmin 4 interface. The left sidebar is titled 'Browser' and lists several database objects under 'Buyer'. The main area is a query editor window with the following SQL code:

```
--query 16
select *
from "202001213_db"."Buyer"
where adv_amount < 0.1*(select sum(adv_amount) from "202001213_db"."Buyer")

-- query 17
create view top_buyers as
select *
from "202001213_db"."Buyer"
where adv_amount > 250000
```

The message bar at the bottom right indicates: 'Query returned successfully in 22 secs 177 msec.'

The screenshot shows the pgAdmin 4 interface. The left sidebar is titled 'Browser' and lists various database objects, including 'top_buyers' which is currently selected. The main area is a query editor window with the following SQL code:

```
--query 16
select *
from "202001213_db"."Buyer"
where adv_amount < 0.1*(select sum(adv_amount) from "202001213_db"."Buyer")

-- query 17
create view top_buyers as
select *
from "202001213_db"."Buyer"
where adv_amount > 250000

select * from top_buyers
```

The message bar at the bottom right indicates: 'Successfully run. Total query runtime: 11 secs 799 msec. 16 rows affected.'

Below the query editor, a results grid displays 16 rows of data from the 'top_buyers' view. The columns are: buyer_id, first_name, last_name, city, street, quantity, adv_amount, id_proof, beer_id, and beer_name. The data includes entries for buyers like Clemmie Schowalter, Mante Schuyler, and others, with their respective addresses and purchase details.

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;
```

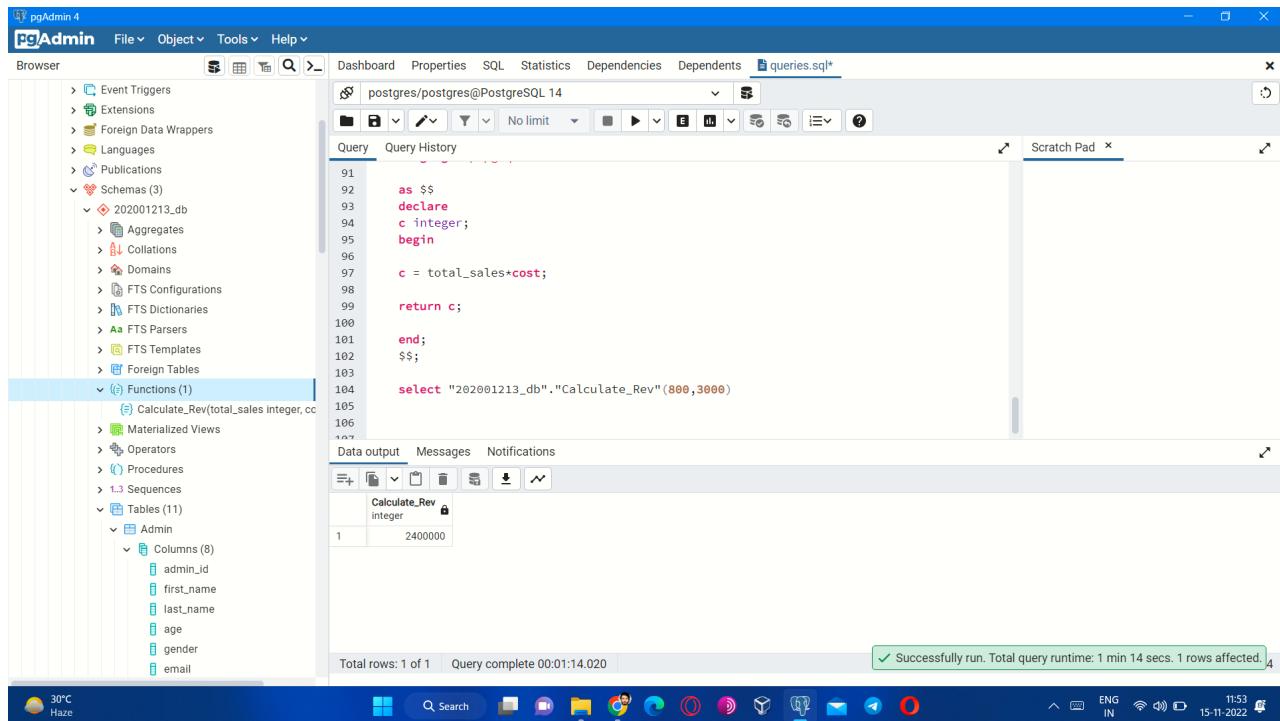
```
end;
```

```
$$;
```

The screenshot shows the pgAdmin 4 interface with the following details:

- Toolbar:** File, Object, Tools, Help.
- Left Panel (Browser):** Shows the database structure:
 - Schemas (3): 202001213_db
 - Functions (1): Calculate_Rev(total_sales integer, cost integer)
 - Tables (11): Admin, Columns (8) with columns: admin_id, first_name, last_name, age, gender, email.
- Center Panel (Query Editor):** A code editor window titled "queries.sql" containing the SQL code for creating the function. The code is identical to the one provided in the text above, including the function definition, variables, assignment, return statement, end block, and final semicolon.
- Status Bar:** Shows "Total rows: 16 of 16" and "Query complete 00:00:00.112".
- Bottom Bar:** Shows system icons for battery, signal, and date/time (15-11-2022, 11:52).

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



The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database schema, including Schemas, Functions (with one named 'Calculate_Rev'), and Tables (with one named 'Calculate_Rev' containing a single row with value 2400000). The main window shows a SQL query editor with the following code:

```
91      as $$  
92      declare  
93          c integer;  
94      begin  
95          c = total_sales*cost;  
96      end;  
97      return c;  
98      $$;  
99  
100     select "202001213_db"."Calculate_Rev"(800,3000)  
101  
102  
103  
104  
105  
106
```

The status bar at the bottom right indicates "Successfully run. Total query runtime: 1 min 14 secs. 1 rows affected.".

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()

```

The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' pane displays the database schema, including tables like Admin, Beer, and Buyer, and triggers like trigger_insert. In the center, the 'Query' pane contains the SQL code for creating the trigger and executing it. The code is as follows:

```

116    -- create trigger "trigger_insert"
117        after INSERT
118        on "202001213_db"."Buyer"
119        for each row
120        execute procedure "202001213_db".trigger_func()
121
122
123
124
125
126
127
128
129
130    insert into "202001213_db"."Buyer" values(1234,'ketan','khunti',
131                                'porbandar','bhanvad',100,0,'pan card',695,'Lagers')
132

```

The 'Messages' tab in the 'Data output' section shows the following notice and query results:

- NOTICE: can not buy with 0 advance amount
- INSERT 0 1
- Query returned successfully in 38 msec.

The status bar at the bottom indicates 'Query returned successfully in 38 msec.' and the current date and time as 15-11-2022 13:42.

```

insert into "202001213_db"."Buyer" values(1234,'ketan','khunti',
'porbandar','bhanvad',100,0,'pan card',695,'Lagers')

```

```

    118     end ;
119     return new;
120   end;
121   $$;
122
123   create trigger "trigger_insert"
124   after INSERT
125   on "202001213_db"."Buyer"
126   for each row
127   execute procedure "202001213_db".trigger_func()
128
129
130   insert into "202001213_db"."Buyer" values(1234,'ketan','khunti',
131                                         'porbandar','bhavnad',100,0,'pan card',695,'Lagers')
132
133 select * from "202001213_db"."Buyer"
134

```

buyer_id	first_name	last_name	city	street	quantity	adv_amount	id_proof	beer_id	beer_name	
45	9172	Angelita	Cremin	North Dangelo	Dicki Walks	75	375000	voting card	214	Belgian-style ale
46	7838	Fredra	Maggio	New Steveton	Demario Bypass	85	450000	pan card	9374	Stouts and porters
47	8512	Arlie	Cummerata	Auermouth	Hills Fall	55	240000	driving license	2781	Sour Beer
48	931	Maido	Hill	New Alphonson...	Jamal Mall	70	350000	adhar card	1487	Heineken 0.0 No...
49	4823	Kaelyn	Waelchi	Lake Electa	Genevieve Roads	60	375000	light bill	4850	Crofters Belgian ...
50	369	Demarcus	Terry	Emanuelside	Alphonso Road	50	200000	driving license	9317	Coolberg Non-Al...

Total rows: 50 of 50 Query complete 00:00:07.849 Ln 133, Col 37

=> 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);
```

The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' panel displays the database schema, including tables like Beer, Buyer, and Employee, along with their columns, constraints, indexes, and triggers. A specific trigger named 'trigger_insert' is selected in the Beer table's triggers section. On the right, the 'Query' tab of the main window contains the SQL code for creating the trigger and inserting data. The code includes a check for existing rows and a notice message if it finds one. The insert statement attempts to add a row with beer_id 695 and name 'Lagers', but fails due to a unique constraint violation because a row with beer_id 695 already exists. The 'Messages' tab shows the resulting error message.

```
146     if new.beer_name = old.beer_name
147     then raise notice 'Beer already exists';
148 end if;
149 return new;
150 end;
151
152 select * from "202001213_db"."Beer";
153
154 create trigger "trigger_insert"
155 before INSERT
156 on "202001213_db"."Beer"
157 for each row
158 execute procedure "202001213_db".trigger_func_2()
159
160 insert into "202001213_db"."Beer" values(695,'Lagers',10,3000,true);
161
```

NOTICE: Beer already exists

ERROR: duplicate key value violates unique constraint "Beer_pkey"
DETAIL: Key (beer_id, beer_name)=(695, Lagers) already exists.
SQL state: 23505

Total rows: 23 of 23 | Query complete 00:00:52.360 | Ln 160, Col 72

The screenshot shows the pgAdmin 4 interface. In the left sidebar, under the 'Tables' section for the 'Beer' table, there is a 'Triggers' folder containing a single entry: 'trigger_insert'. The main pane displays a SQL query window with the following code:

```

146     if new.beer_id = 10 or new.beer_name='name'
147     then raise notice 'Beer already exists';
148     end if;
149     return new;
150   end;
151   $$;
152
153   select * from "202001213_db"."Beer";
154
155   create trigger "trigger_insert"
156   before INSERT
157   on "202001213_db"."Beer"
158   for each row
159   execute procedure "202001213_db".trigger_func_2();
160
161   insert into "202001213_db"."Beer" values(695,'Lagers',10,3000,'true');
162

```

Below the query window, the 'Data output' tab is selected, showing a table with 23 rows of beer data. The columns are: beer_id, beer_name, alcohol%, cost, and availability. The data includes various beer names like Crofters Belgian Wit, Coorberg Non-Alcohol..., Kingfisher Ultra Non..., Barbican Non-Alcohol..., Hoegaarden 0.0 Non..., and Edelmeister Nonalco... with their respective details.

=> 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');**

The screenshot shows the pgAdmin 4 interface with the same setup as the previous one. The SQL query window now contains the following modified code:

```

146     if new.beer_id = 10 or new.beer_name='name'
147     then raise notice 'Beer already exists';
148     end if;
149     return new;
150   end;
151   $$;
152
153   select * from "202001213_db"."Beer";
154
155   create trigger "trigger_insert"
156   before INSERT
157   on "202001213_db"."Beer"
158   for each row
159   execute procedure "202001213_db".trigger_func_2();
160
161   insert into "202001213_db"."Beer" values(690,'Piegel',10,3000,'true');
162

```

The 'Messages' tab is selected in the bottom navigation bar, showing the message: 'INSERT 0 1'. Below that, it says 'Query returned successfully in 1 secs 538 msec.' A green checkmark icon is present next to the message. The status bar at the bottom right shows the date and time as '15-11-2022 14:02'.

pgAdmin 4

File Object Tools Help

Browser

- Sequences
- 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

Dashboard Properties SQL Statistics Dependencies Dependents queries.sql*

Query History

```
146 if new.beer_id = 690 then raise notice 'Beer already exists';
147 end if;
148 return new;
149 end;
150 $$;
151
152 select * from "202001213_db"."Beer";
153
154 create trigger "trigger_insert"
155 before INSERT
156 on "202001213_db"."Beer"
157 for each row
158 execute procedure "202001213_db".trigger_func_2();
159
160 insert into "202001213_db"."Beer" values(690,'Piegel',10,3000,true);
161
```

Scratch Pad

Data output Messages Notifications

beer_id	beer_name	alcohol%	cost	availability
19	Coolberg Non-Alcoholic Beer	0	8243	false
20	Kingfisher Ultra Non-Alcoholic Beer	0	7640	false
21	Barbican Non-Alcoholic Beer	0	644	true
22	Hoegaarden 0.0 Non Alcoholic Wheat B...	0	4804	false
23	Edelmeister Nonalcoholic Beer	0	684	false
24	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.

34°C Sunny

Windows Taskbar: Search, File Explorer, Internet Explorer, Mail, Control Panel, Task View, Start, File, Settings, Weather, Date/Time (14:02, 15-11-2022), ENG IN, WiFi, Battery