ART GALLERIES DATABASE

Subject area analysis: structural and functional analysis:

The primary users of this art galleries database will consist of customers and visitors.

Customers, who are potential buyers, will use the database to:

- browse, search and purchase artworks based on their preferences and parameters (artist, gallery, style, theme, and tag);
- view detailed information about each artwork, including its name, author, medium, price history, and year of the painting;
- purchase artworks directly through the website, with a secure and user-friendly checkout process.

Visitors will have:

- access to basic information about the artworks, galleries, and exhibitions;
- ability to leave feedback and ratings for galleries they have experienced

Artists themselves can also benefit from this database by using it to:

- manage and update their artwork inventory,
- track sales and pricing history,
- provide relevant information to potential buyers.

The database can be accessed through a secure login system, ensuring that only authorized users can modify the data.

Art galleries database aims to automate several key functions within the art industry. These functions include:

- Browse and Search for Artworks;

Database will enable visitors to quickly browse and search for artworks based on several criteria such as the artist, gallery, style, theme, and tags. Customers and visitors will experience faster discovery because of this automation.

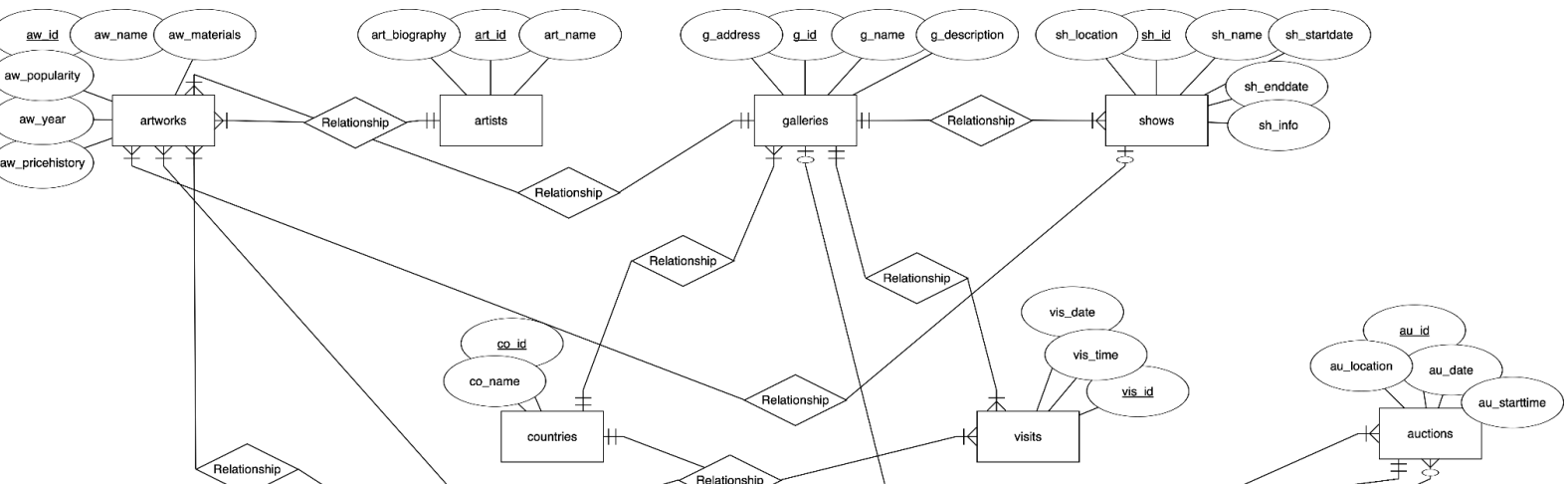
- Detailed Artwork Information;

Users will have access to in-depth information on each piece of art, including the name of the artist, the medium used, the pricing history, and the year the work was created. Potential buyers will be better able to make decisions based on this information, which will also improve their whole art-buying experience.

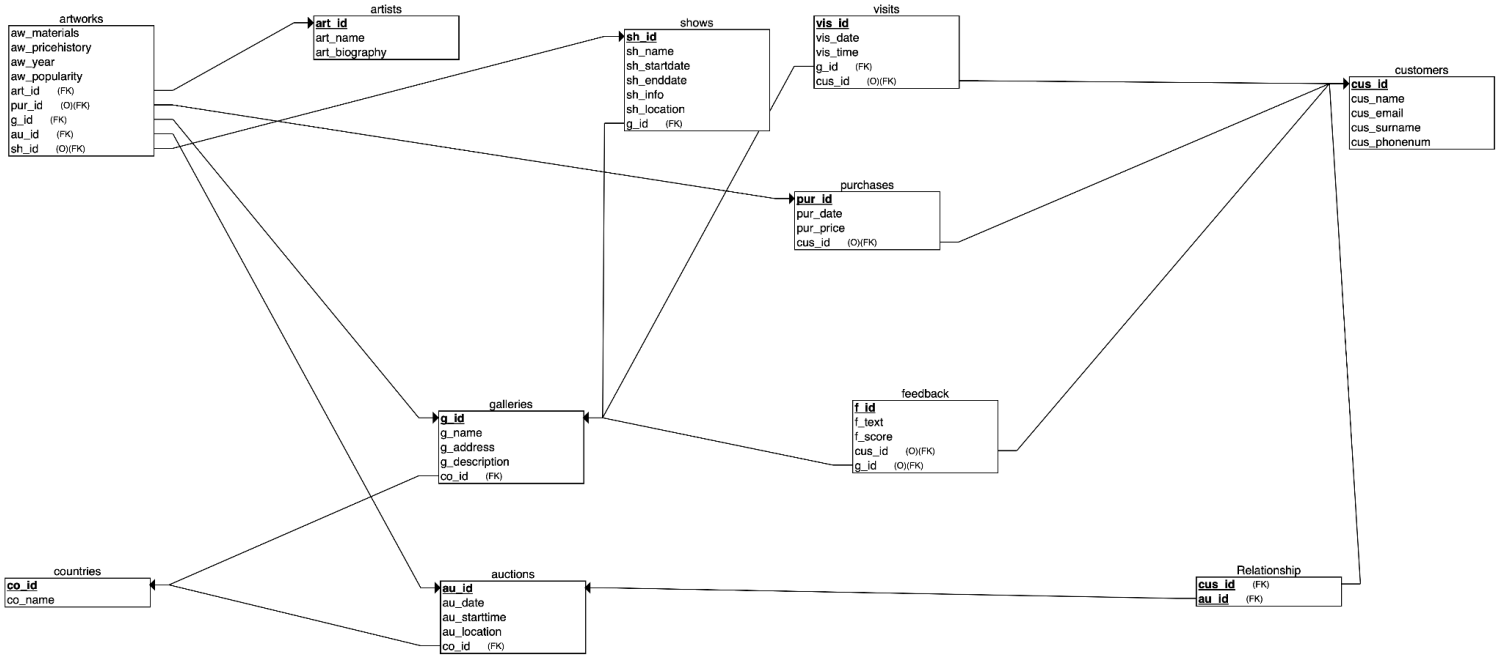
- Management of Exhibitions and Galleries.

Database can allow art galleries to handle their shows, including planning, setting up artwork displays, and updating the appropriate information. The organization and presentation of shows will be simplified by this automation, saving gallery staff members' effort as well as time.

ERDPlus schema:



Relational schema:



1. The "artists" table stores information about the artists, including their unique id, name, biography and artwork id. It is connected to entities like artworks (they've painted).
artist-artwork is one-to-many

2. The "countries" entity stores information about the countries, including their unique id and name, auctions id, gallery id and customer id. This table has relationships with entities like galleries and auctions. We didn't connect country locations with shows, because shows are located in galleries, so that's why we have a relationship only with galleries. However auctions have their own relationship, just because they are located in specified auctions buildings.
country-gallery is one-to-many

country- auctions one-to-many

3. The "customers" table stores information about the customers, including their unique id, name, email, surname, phone number, visit id, auctions id, feedback id and purchase id. The entity has relationships with auctions, visits, feedback and purchase. We didn't connect galleries with customers just because we wanted to know the time and date of the visit to the gallery (so we did it through visit entity), auctions don't have the relationship with customers too, because we didn't need the visit data about these entities, as auctions are held at specific date and time already.
customer-visits is one-to-many
customer-auctions is many-to-many
customer-feedback is one-to-many
customer-purchase is one-to-many

4. Relationship - because of many to many relationship for auctions and customers we have a separate entity
customer-auctions is many-to-many

5. The "purchases" table stores information about the purchases made by customers, including the purchase id, date, price, artwork id and the customer id associated with the purchase.
purchase-customer is many-to-one
purchase-artwork is one-to-many

6. The "galleries" table stores information about the galleries, including their unique id, name, address, description, feedback id and the country Id associated with the gallery. They're also connected to shows which are or will be held in galleries.
gallery-country is many-to-one
gallery-shows is one-to-many
gallery-feedback one-to-many

7. The "shows" table stores information about the art shows/exhibitions, including their unique Id, name, start date, end date, information, location, and the gallery id associated with the show.
shows-gallery is many-to-one
shows-artworks is one-to-many
9. The "visits" table stores information about the visits made by customers to galleries, including the visit id, date, time, and the customer id associated with the visit.
visit-customer is many-to-one

10. The "auctions" table stores information about the art auctions, including the auction id, date, start time, location, country id, and customer id associated with the auction.

auctions-country many-to-one
auctions-customer many-to-many

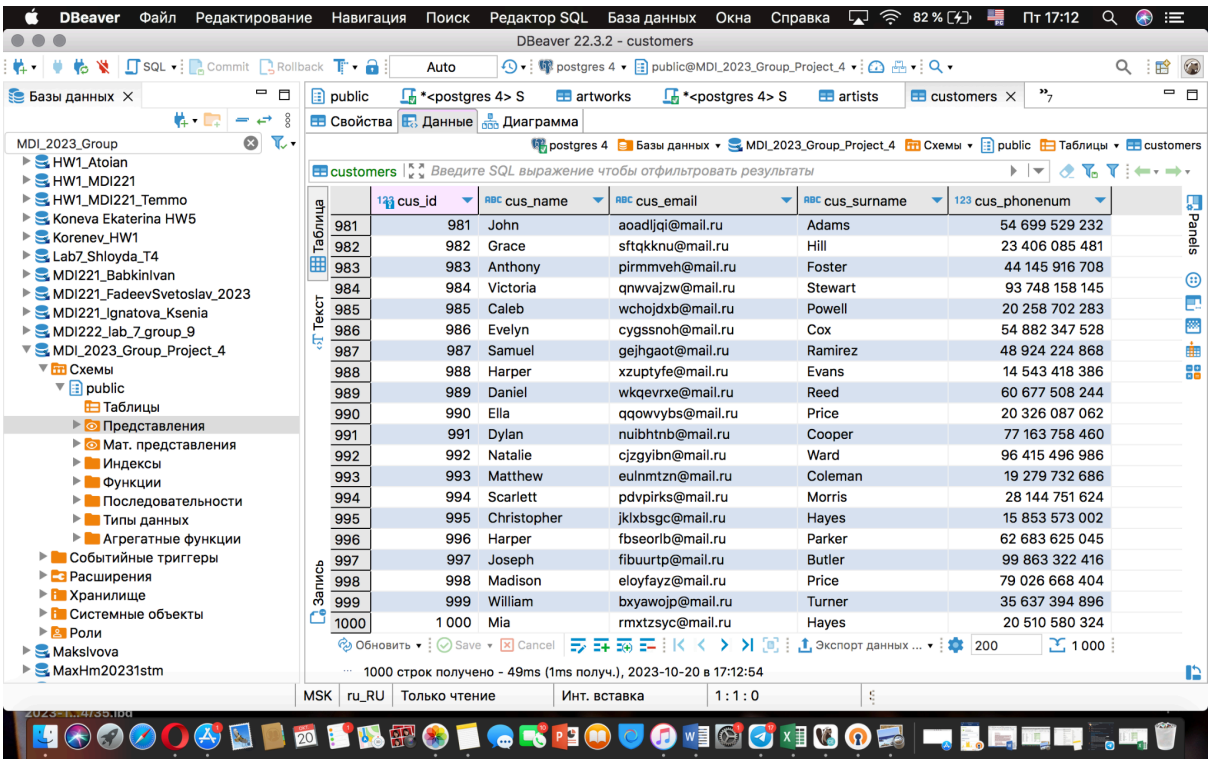
11. The "feedback" table stores information about the feedback given by customers for galleries, including the feedback id, text, score, customer id, and gallery id associated with the feedback. This relationship with customers and galleries gives an user the opportunity to give review on the galleries and the paintings which are located there.
feedback-gallery many-to-one
feedback-customer many-to-one

12. The "artworks" table stores information about the artworks, including their unique ID, name, materials used, price history, year of creation, popularity score, artist id, purchase id (if sold), gallery id, shows id(if artist want to sell it), auction id (if applicable), and show id associated with the artwork (if it is located in some exhibition)
artworks-artist is many-to-one
artworks-purchase is many-to-one
artworks-gallery is many-to-one
artworks-auctions is many-to-one
artworks-shows is many-to-one

Overall, the ERDPlus diagram represents the relationships between various entities involved in the website's functionality, such as artists, customers, galleries, shows, auctions, and artworks.

Diagram in DB:

Example of filling the table with data:

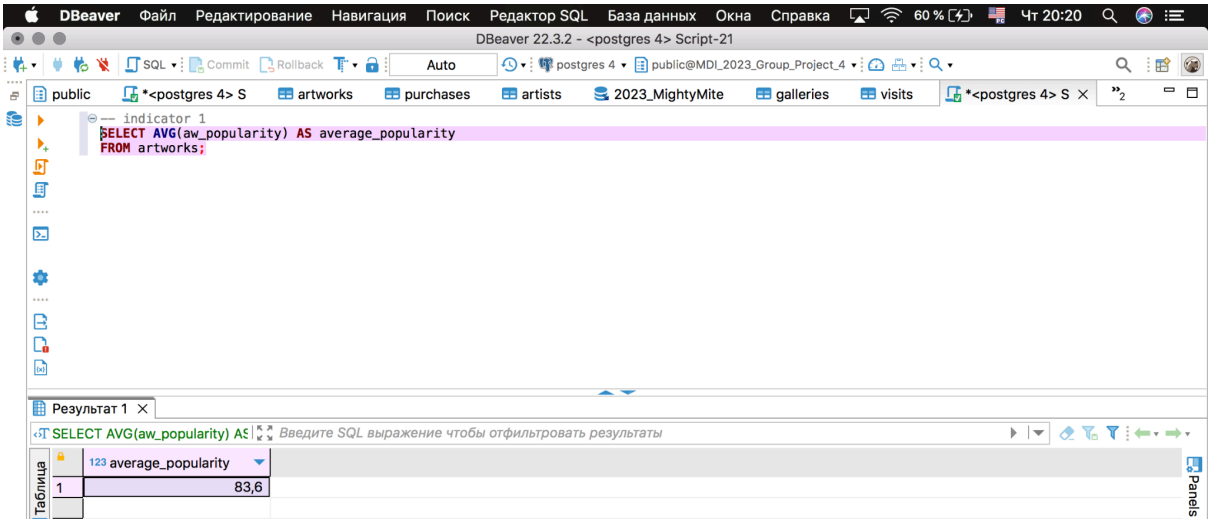


Additional requirements to DB:

Add more 3-4 average indicators about artworks.

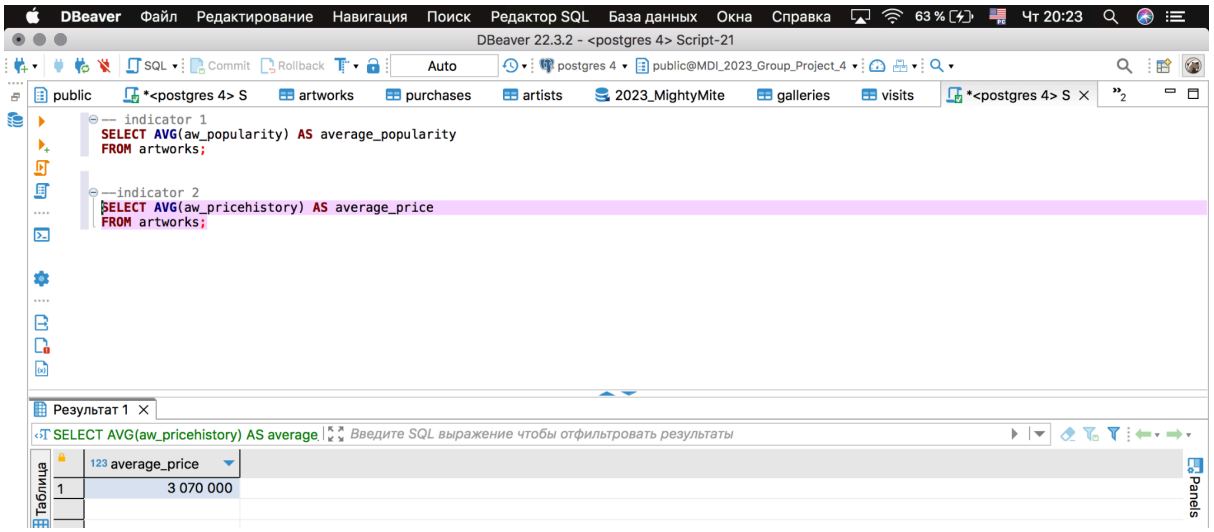
1. The first indicator finds the average popularity of all artworks.

```
SELECT AVG(aw_popularity) AS average_popularity
FROM artworks;
```



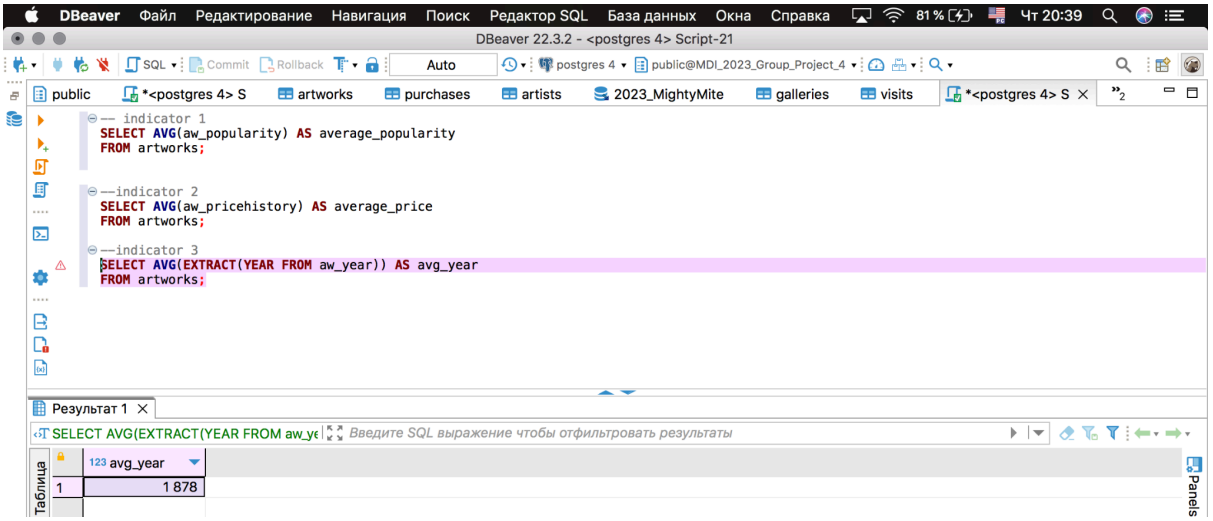
2. The second indicator finds the average price of the artworks.

```
SELECT AVG(aw_pricehistory) AS average_price
FROM artworks;
```



3. The third indicator finds an average year of the artwork painted.

```
SELECT AVG(EXTRACT(YEAR FROM aw_year)) AS avg_year
FROM artworks;
```



1. Functions

Function that will search by artist, gallery, style, theme and tag and return list of artworks with parameters (number of records in the output and sort order on attribute).

```
CREATE OR REPLACE FUNCTION search_artworks(
    artist_name varchar,
    gallery_name varchar,
    art_style varchar,
    art_theme varchar,
    tag varchar,
    record_count int,
    sort_order varchar
)
RETURNS TABLE (
    aw_id int,
    aw_name varchar,
    aw_materials varchar,
    aw_pricehistory numeric,
    aw_year date,
    aw_popularity numeric,
    art_id int,
    pur_id int,
    g_id int,
    au_id int,
    sh_id int,
    aw_style varchar
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        aw.aw_id,
        aw.aw_name,
        aw.aw_materials,
        aw.aw_pricehistory,
        aw.aw_year,
        aw.aw_popularity,
        aw.art_id,
        aw.pur_id,
        aw.g_id,
        aw.au_id,
        aw.sh_id,
        aw.aw_style
    FROM
        public.artworks AS aw
    LEFT JOIN
        public.artists AS art ON aw.art_id = art.art_id
    LEFT JOIN
        public.galleries AS gal ON aw.g_id = gal.g_id
    WHERE
        (artist_name IS NULL OR art.art_name ILIKE '%' || artist_name || '%')
        AND (gallery_name IS NULL OR gal.g_name ILIKE '%' || gallery_name || '%')
        AND (art_style IS NULL OR aw.aw_style ILIKE '%' || art_style || '%')
        AND (art_theme IS NULL OR aw.aw_name ILIKE '%' || art_theme || '%')
        AND (tag IS NULL OR aw.aw_id = CAST(tag AS int))
    ORDER BY
        CASE WHEN sort_order = 'popularity' THEN aw.aw_popularity END DESC,
        CASE WHEN sort_order = 'year' THEN aw.aw_year END DESC,
        CASE WHEN sort_order = 'name' THEN aw.aw_name END
    LIMIT
        record_count;
END;
$$ LANGUAGE plpgsql;
```

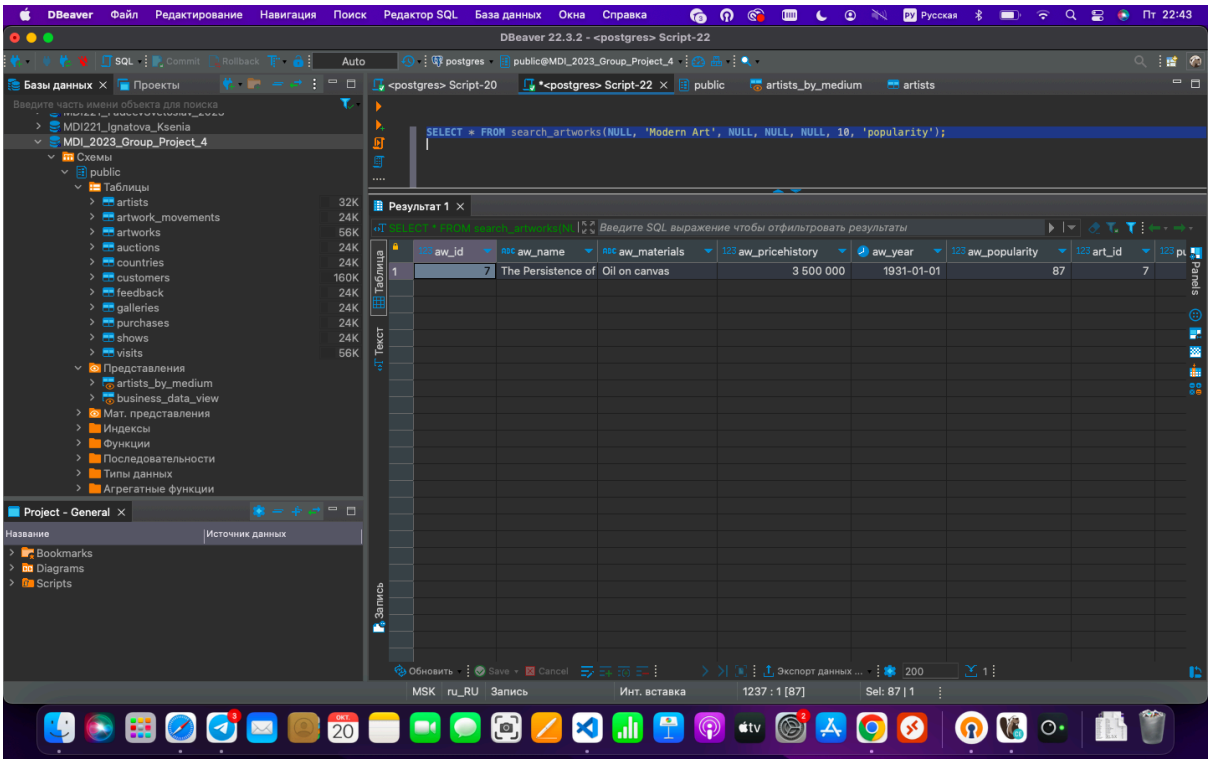
Explanation:

- The BEGIN block marks the beginning of the function's executable code.
- RETURN QUERY is used to indicate that a query result will be returned as the output of the function.
- The SELECT statement retrieves data from the public.artworks table and uses LEFT JOIN to connect with the public.artists and public.galleries tables based on foreign key relationships.
- The WHERE clause filters the results based on the provided search criteria. Each condition checks whether the corresponding parameter is NULL or, if not, whether the column values match the search criteria (e.g., artist name containing the artist_name parameter).
- The ORDER BY clause specifies the sorting order of the results based on the sort_order parameter. It can sort by popularity, year, or name in descending order.
- The LIMIT clause limits the number of records returned to the value specified in the record_count parameter.

Example:

Search for artworks in the "Modern Art" gallery, sorted by popularity:

SELECT * FROM search_artworks(NULL, 'Modern Art', NULL, NULL, NULL, 10, 'popularity');



2. Trigger

Create a trigger that will write a record to a separate table with information about the artwork and time when it was moved to a new exhibition.

– Creating artwork_movements table

```
CREATE TABLE public.artwork_movements (
    movement_id serial PRIMARY KEY,
    aw_id int4 NOT NULL REFERENCES public.artworks(aw_id),
    previous_sh_id int4 REFERENCES public.shows(sh_id),
    new_sh_id int4 REFERENCES public.shows(sh_id),
    movement_date timestamp DEFAULT current_timestamp
);
```

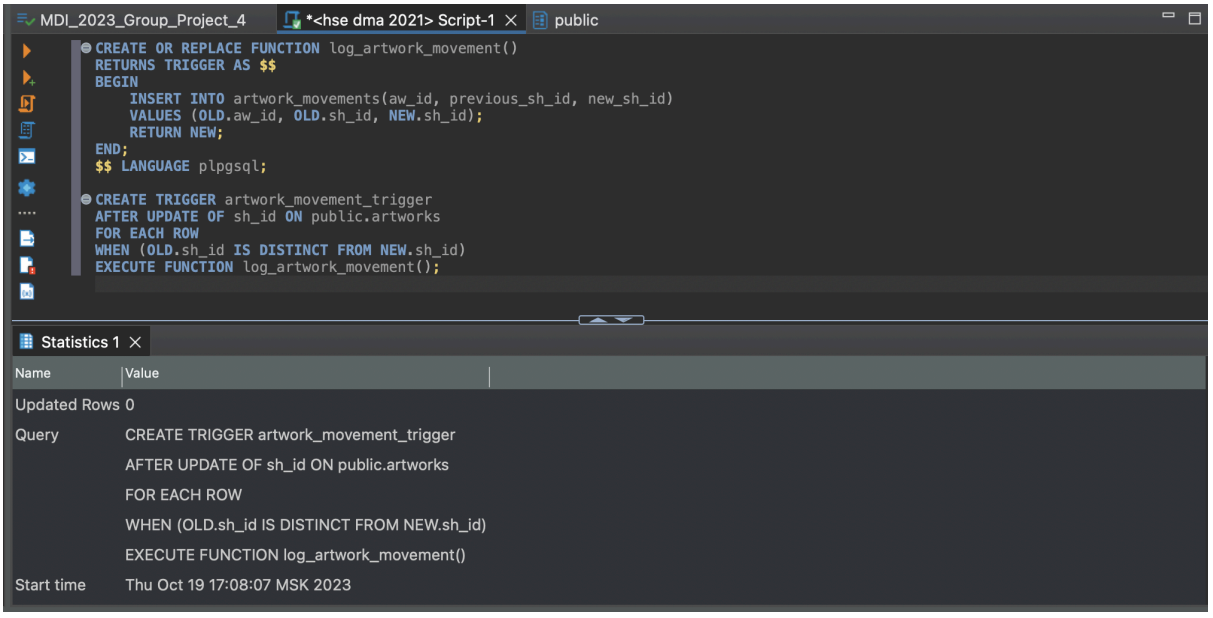
Table description:

- movement_id: A unique identifier for the movement record.
- aw_id: The ID of the artwork that has been moved.
- previous_sh_id: The ID of the previous exhibition where the artwork was.
- new_sh_id: The ID of the new exhibition where the artwork is moved.
- movement_date: The timestamp when the artwork was moved (defaulted to the current timestamp).

– Creating trigger and trigger function

```
CREATE OR REPLACE FUNCTION log_artwork_movement()
RETURNS TRIGGER AS $$
BEGIN
   INSERT INTO artwork_movements(aw_id, previous_sh_id, new_sh_id)
   VALUES (OLD.aw_id, OLD.sh_id, NEW.sh_id);
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER artwork_movement_trigger
AFTER UPDATE OF sh_id ON public.artworks
FOR EACH ROW
WHEN (OLD.sh_id IS DISTINCT FROM NEW.sh_id)
EXECUTE FUNCTION log_artwork_movement();
```



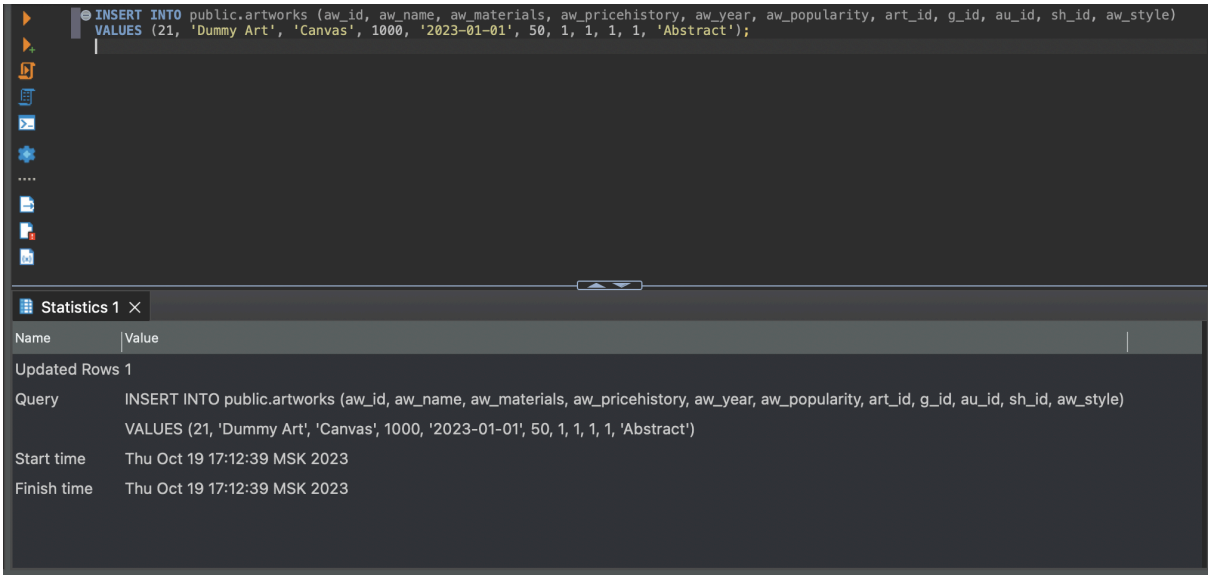
What happens:

- The log_artwork_movement function is a PL/pgSQL function that inserts a record into artwork_movements every time an artwork's sh_id (show/exhibition ID) is updated.
- The artwork_movement_trigger is an AFTER UPDATE trigger on the artworks table which fires whenever the sh_id column is updated and the old value is different from the new value. It calls the log_artwork_movement function to perform the logging.
- With this trigger in place, every time an artwork is moved to a different exhibition, a record will be stored in the artwork_movements table capturing this movement along with the timestamp of the movement.

Example:

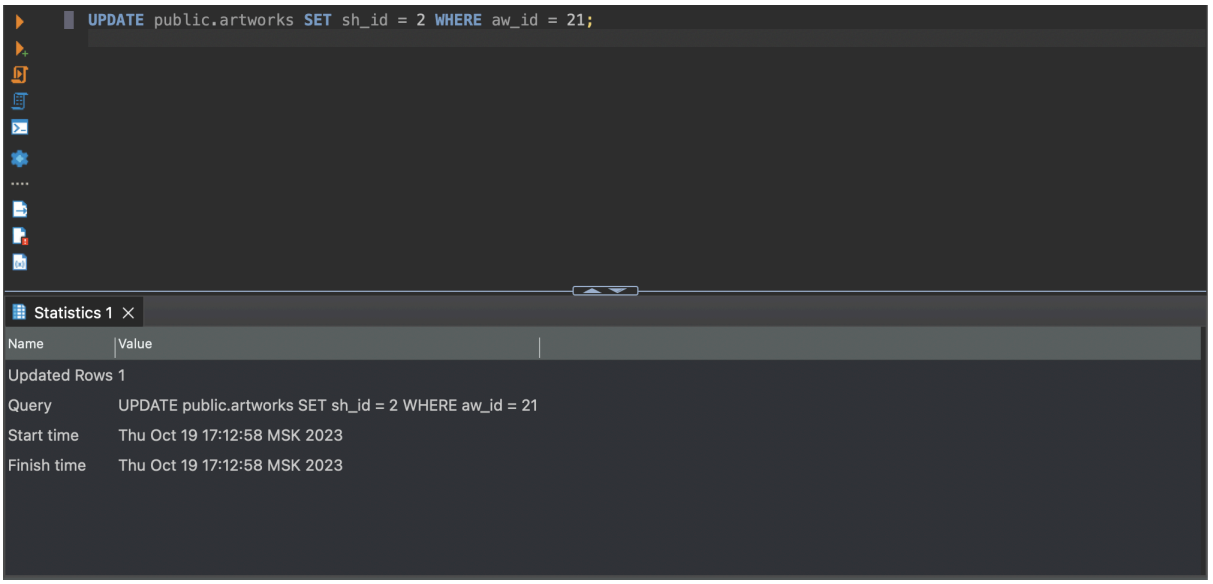
1. First, insert a dummy artwork with an exhibition (sh_id):

INSERT INTO public.artworks (aw_id, aw_name, aw_materials, aw_pricehistory, aw_year, aw_popularity, art_id, g_id, au_id, sh_id, aw_style)
VALUES (21, 'Dummy Art', 'Canvas', 1000, '2023-01-01', 50, 1, 1, 1, 1, 'Abstract');



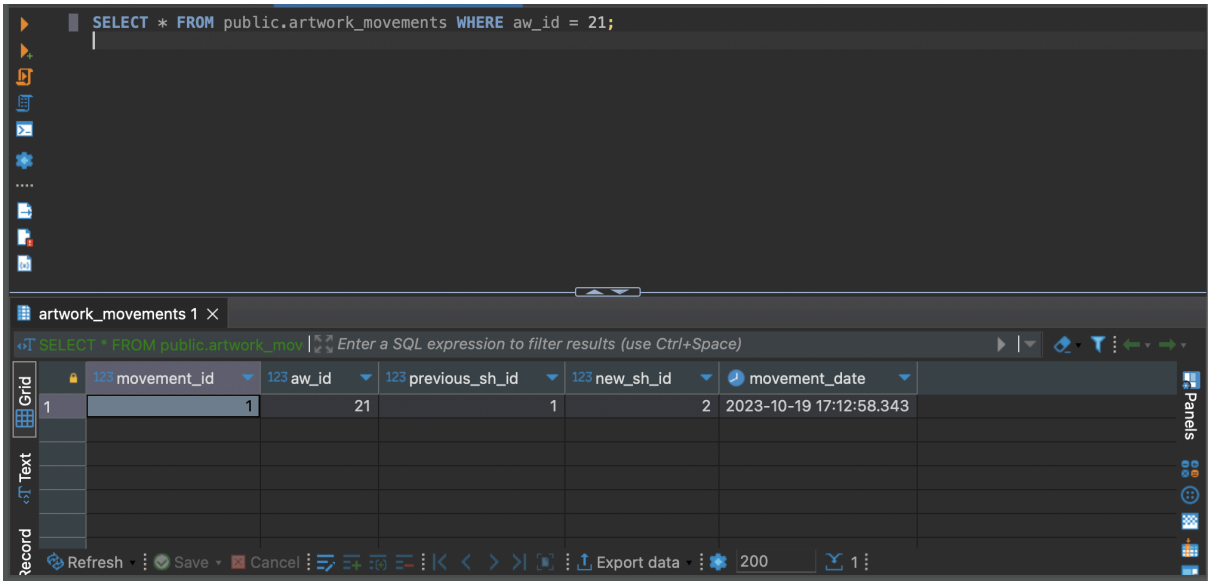
2. Now, simulate the movement of the artwork to another exhibition (e.g., from sh_id = 1 to sh_id = 2):

UPDATE public.artworks SET sh_id = 2 WHERE aw_id = 21;



3. Now, let's check the artwork_movements table to see the logged movement:

SELECT * FROM public.artwork_movements WHERE aw_id = 21;



4. Clean up

```
DELETE FROM public.artworks WHERE aw_id = 21;
DELETE FROM public.artwork_movements WHERE aw_id = 21;
```

Views:

Create two views. The first view will be an editable view that shows business data (like project's attribute, employee's contacts, etc.) and allows users or programs to change it. The second view should be used as a data source for your report in Excel or another visualization tool. Pay attention to details like formatting of your report.

Editable view:

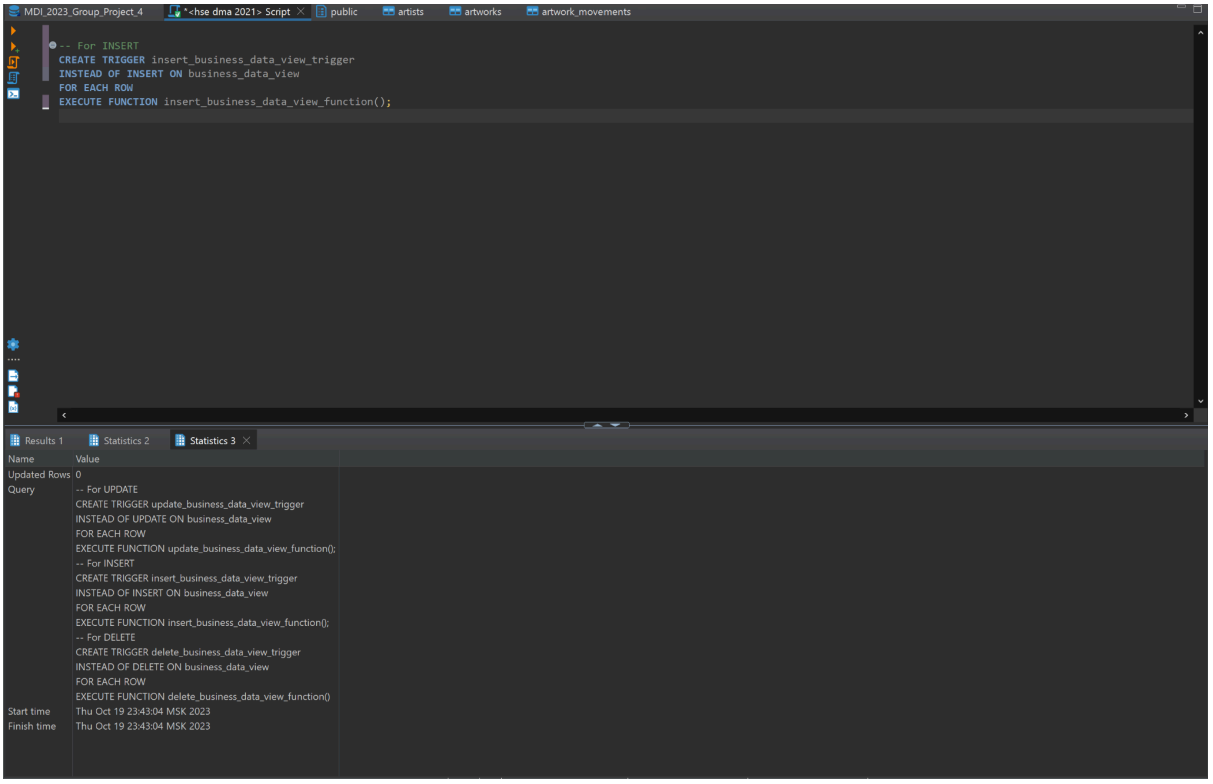
We created a view that consolidates data from the artworks, artists, and customers tables. This view will provide details on artworks, who created them, and who purchased them:

```
CREATE VIEW business_data_view AS
SELECT
    aw.aw_id,
    aw.aw_name,
    aw.aw_materials,
    aw.aw_pricehistory,
    aw.aw_year,
    aw.aw_popularity,
    aw.aw_style,
    art.art_name AS artist_name,
    art.art_biography AS artist_biography,
    cus.cus_name || ' ' || cus.cus_surname AS customer_fullname,
    cus.cus_email,
    cus.cus_phonenum
FROM public.artworks aw
LEFT JOIN public.artists art ON aw.art_id = art.art_id
LEFT JOIN public.purchases pur ON aw.pur_id = pur.pur_id
LEFT JOIN public.customers cus ON pur.cus_id = cus.cus_id;
```

- **INSERT:**

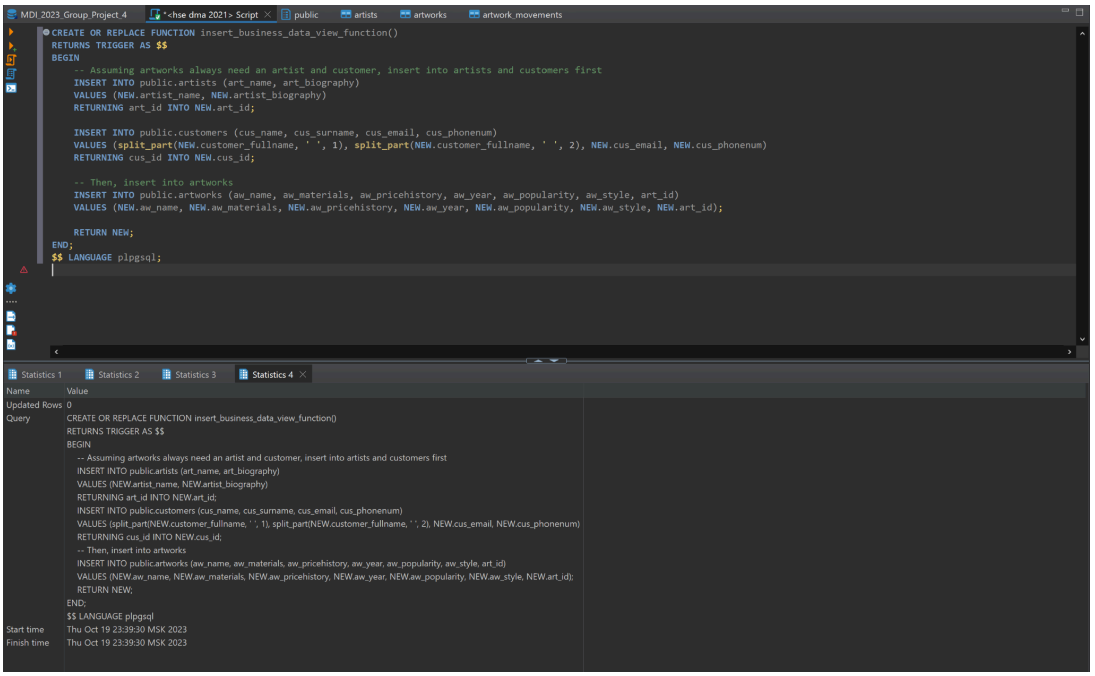
Trigger for INSERT:

-- For INSERT
CREATE TRIGGER insert_business_data_view_trigger
INSTEAD OF INSERT ON business_data_view
FOR EACH ROW
EXECUTE FUNCTION insert_business_data_view_function();



Function:

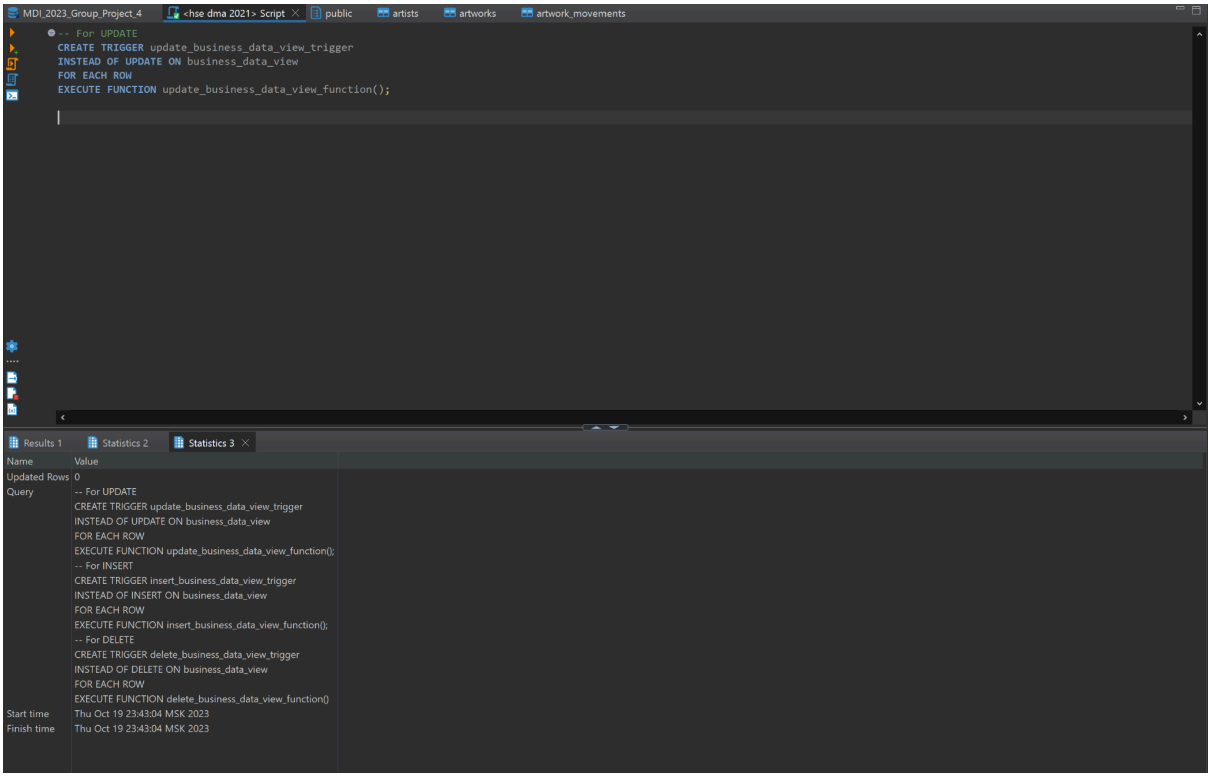
```
CREATE OR REPLACE FUNCTION insert_business_data_view_function()
RETURNS TRIGGER AS $$
BEGIN
    -- Assuming artworks always need an artist and customer, insert into artists and customers first
    INSERT INTO public.artists (art_name, art_biography)
    VALUES (NEW.artist_name, NEW.artist_biography)
    RETURNING art_id INTO NEW.art_id;

    INSERT INTO public.customers (cus_name, cus_surname, cus_email, cus_phonenum)
    VALUES (split_part(NEW.customer_fullname, ' ', 1), split_part(NEW.customer_fullname, ' ', 2), NEW.cus_email, NEW.cus_phonenum)
    RETURNING cus_id INTO NEW.cus_id;

    -- Then, insert into artworks
    INSERT INTO public.artworks (aw_name, aw_materials, aw_pricehistory, aw_year, aw_popularity, aw_style, art_id)
    VALUES (NEW.aw_name, NEW.aw_materials, NEW.aw_pricehistory, NEW.aw_year, NEW.aw_popularity, NEW.aw_style, NEW.art_id);

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```



- **UPDATE**

Trigger for UPDATE:

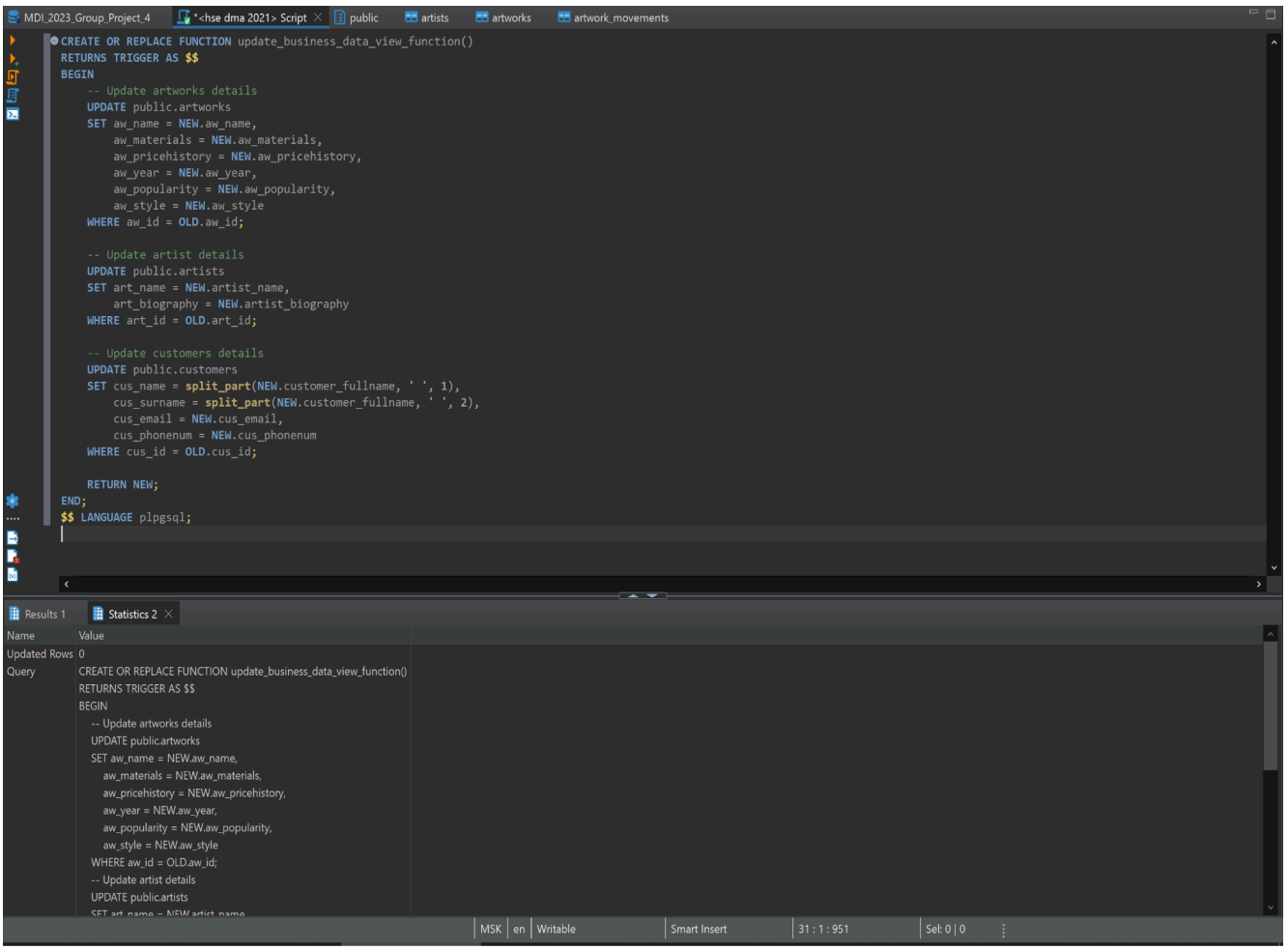
```
-- For UPDATE
CREATE TRIGGER update_business_data_view_trigger
INSTEAD OF UPDATE ON business_data_view
FOR EACH ROW
EXECUTE FUNCTION update_business_data_view_function();
```



Function:

```
CREATE OR REPLACE FUNCTION update_business_data_view_function()
RETURNS TRIGGER AS $$
BEGIN
    -- Update artworks details
    UPDATE public.artworks
    SET aw_name = NEW.aw_name,
        aw_materials = NEW.aw_materials,
        aw_pricehistory = NEW.aw_pricehistory,
        aw_year = NEW.aw_year,
        aw_popularity = NEW.aw_popularity,
        aw_style = NEW.aw_style
    WHERE aw_id = OLD.aw_id;
```

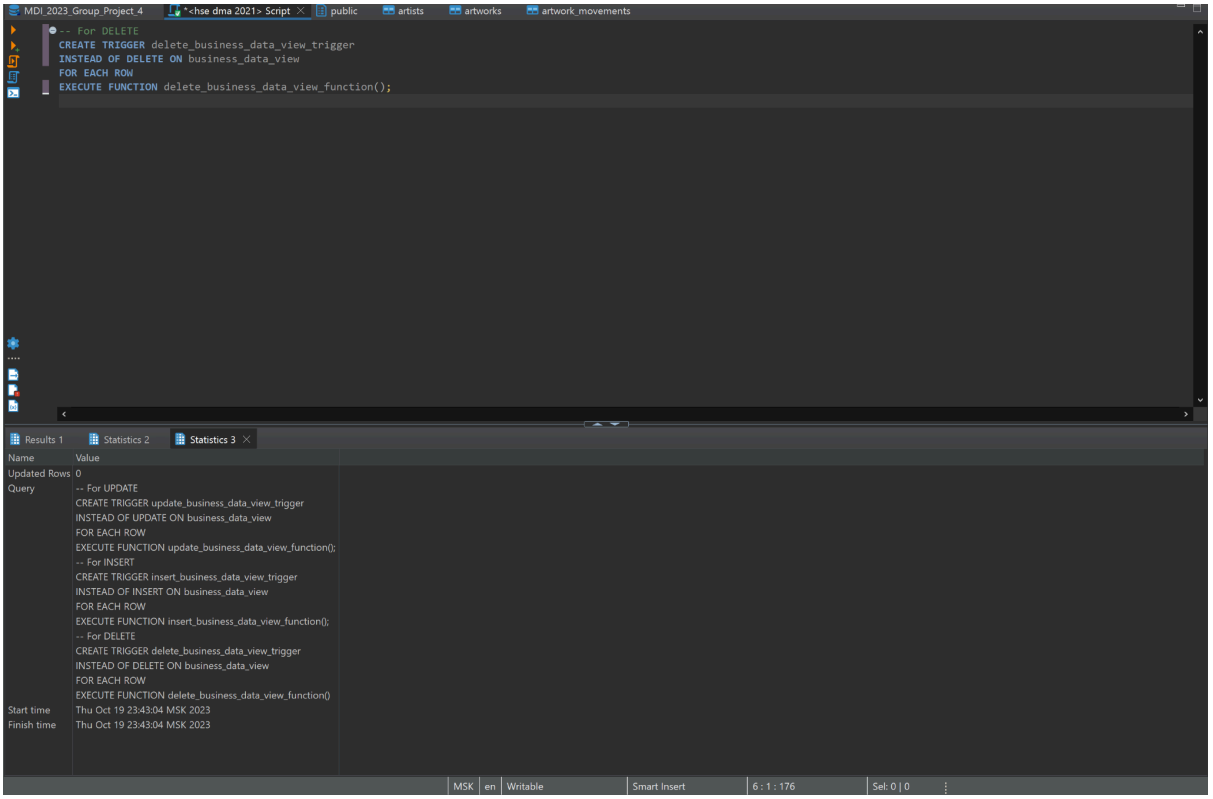
```
    -- Update artist details
    UPDATE public.artists
    SET art_name = NEW.artist_name,
        art_biography = NEW.artist_biography
    WHERE art_id = OLD.art_id;

    -- Update customers details
    UPDATE public.customers
    SET cus_name = split_part(NEW.customer_fullname, ' ', 1),
        cus_surname = split_part(NEW.customer_fullname, ' ', 2),
        cus_email = NEW.cus_email,
        cus_phonenum = NEW.cus_phonenum
    WHERE cus_id = OLD.cus_id;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```



DELETE:

Trigger for DELETE:

```
-- For DELETE
CREATE TRIGGER delete_business_data_view_trigger
INSTEAD OF DELETE ON business_data_view
FOR EACH ROW
EXECUTE FUNCTION delete_business_data_view_function();
```

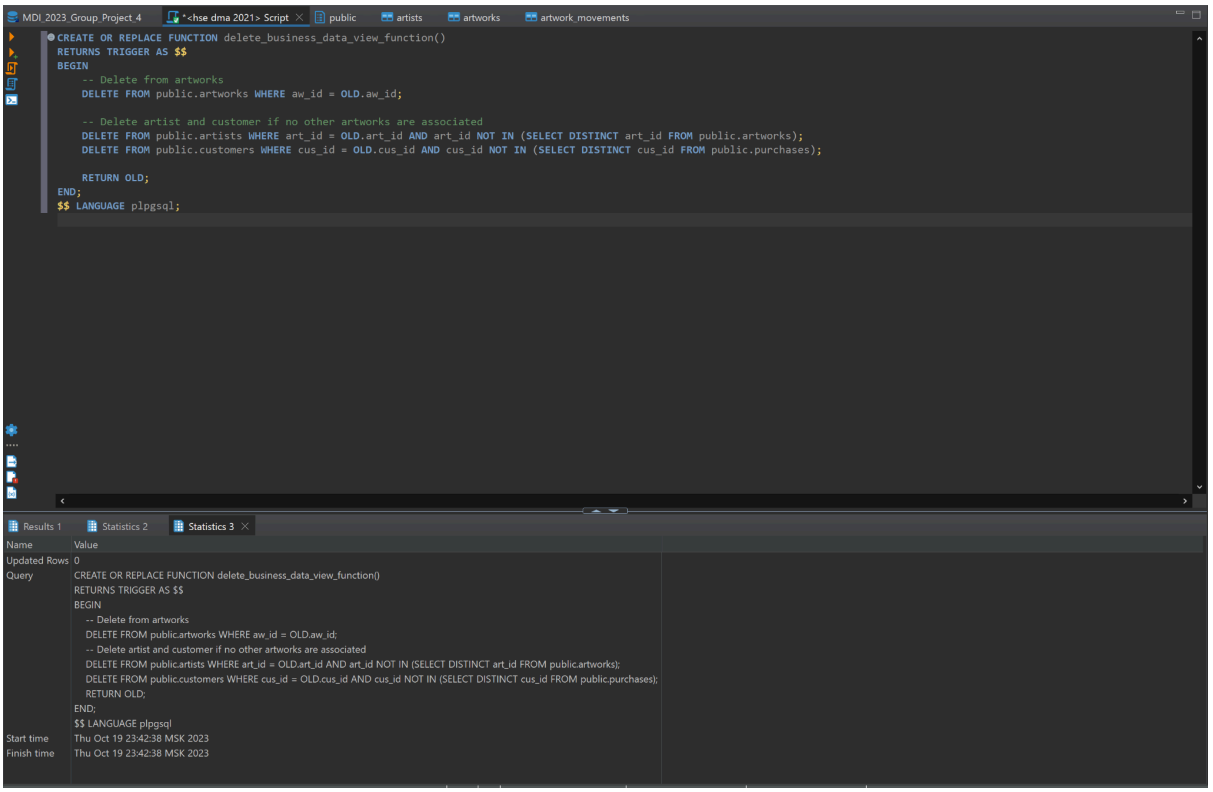


;

Function:

```
CREATE OR REPLACE FUNCTION delete_business_data_view_function()
RETURNS TRIGGER AS $$
BEGIN
    -- Delete from artworks
    DELETE FROM public.artworks WHERE aw_id = OLD.aw_id;
```

```
    -- Delete artist and customer if no other artworks are associated
    DELETE FROM public.artists WHERE art_id = OLD.art_id AND art_id NOT IN
(SELECT DISTINCT art_id FROM public.artworks);
    DELETE FROM public.customers WHERE cus_id = OLD.cus_id AND cus_id NOT
IN (SELECT DISTINCT cus_id FROM public.purchases);
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;
```



Example

```
INSERT INTO business_data_view

(aw_name, aw_materials, aw_pricehistory, aw_year, aw_popularity, aw_style,
artist_name, artist_biography, customer_fullname, cus_email, cus_phonenum)

VALUES

('Mystic Landscape', 'Oil on Canvas', 5000, '2023-01-01', 90, 'Impressionism', 'Jane
Doe', 'A contemporary artist known for her vibrant colors.', 'John Smith',
'johnsmith@email.com', 1234567890);



UPDATE business_data_view

SET

    aw_materials = 'Acrylic on Canvas',

    aw_pricehistory = 5500,

    artist_biography = 'A renowned contemporary artist known for her diverse
techniques.'

WHERE

    aw_name = 'Mystic Landscape' AND artist_name = 'Jane Doe' AND
customer_fullname = 'John Smith';

DELETE FROM business_data_view

WHERE
```

```
   aw_name = 'Mystic Landscape' AND artist_name = 'Jane Doe' AND
customer_fullname = 'John Smith'
```

Report View:

Create an automated report in Excel with the following fields:

- Medium

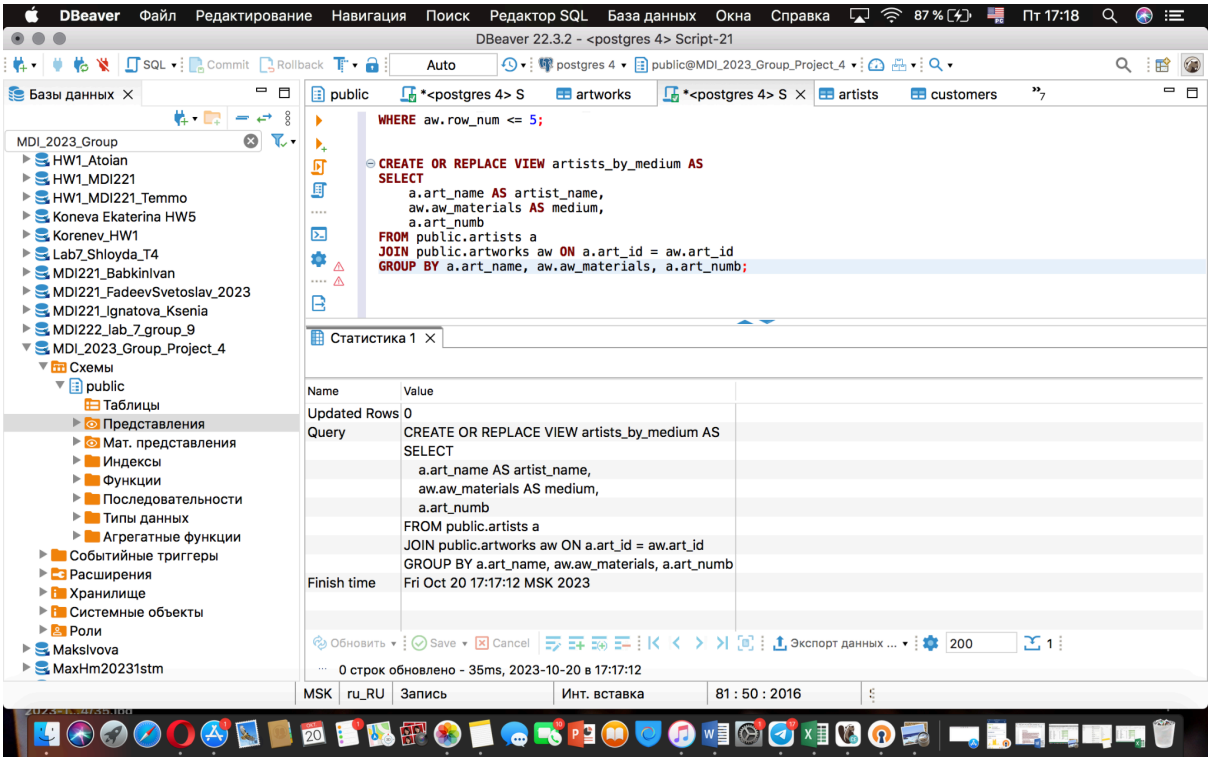
- Artist

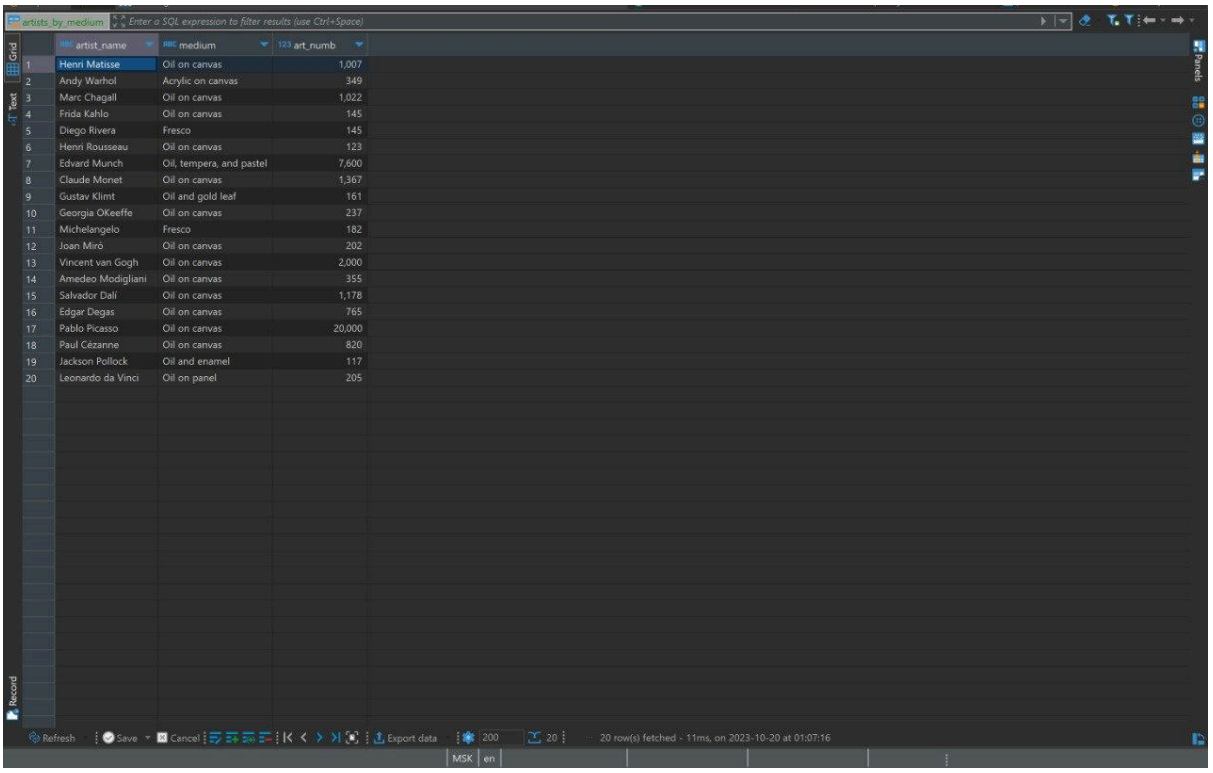
- Number of artworks

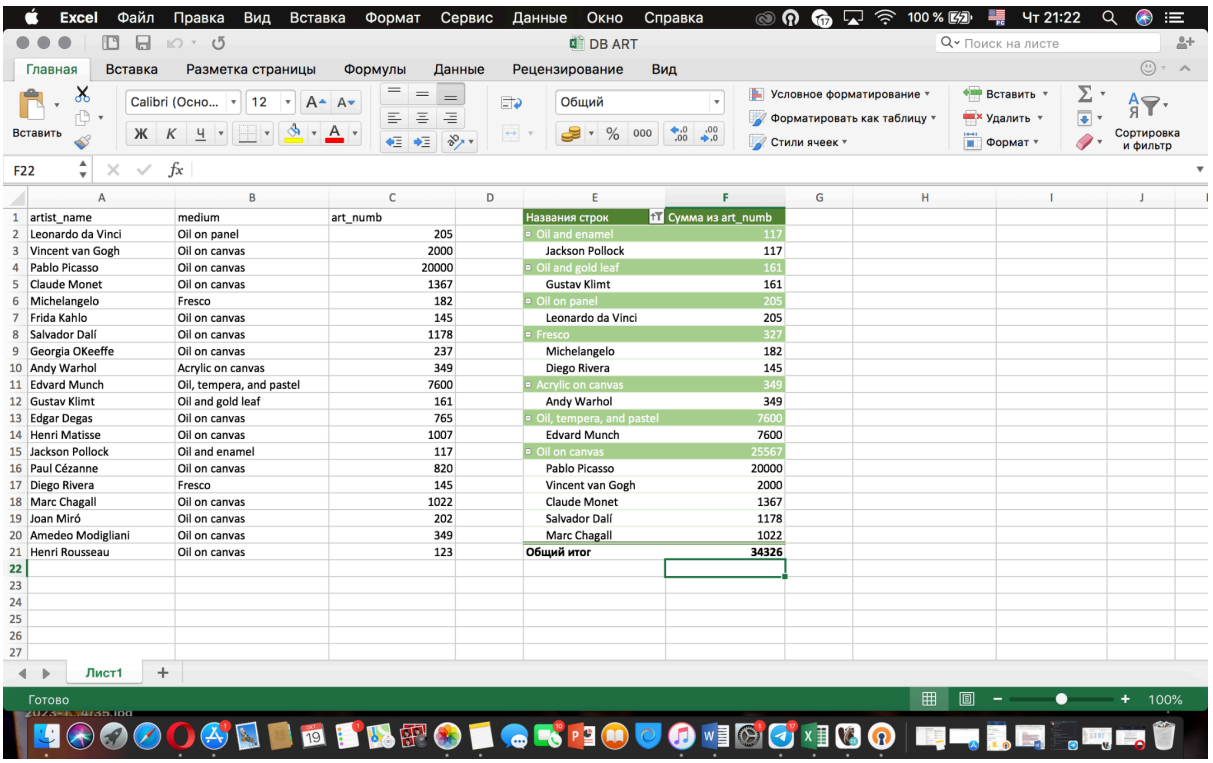
Show only artists that are in the top-5 by number of artworks in each art medium.

We will use this view to find the top 5 number of artworks

```
CREATE OR REPLACE VIEW artists_by_medium AS
SELECT
   a.art_name AS artist_name,
   aw.aw_materials AS medium,
   a.art_numb
FROM public.artists a
JOIN public.artworks aw ON a.art_id = aw.art_id
GROUP BY a.art_name, aw.aw_materials, a.art_numb;
```


DBeaver Файл Редактирование Навигация Поиск Редактор SQL База данных Окна Справка Пт 17:18
DBeaver 22.3.2 - <postgres 4> Script-21
Базы данных
MDI_2023_Group
HW1_Atoian
HW1_MDI221
HW1_MDI221_Temmo
Koneva Ekaterina HW5
Korenev_HW1
Lab7_Shloyda_T4
MDI221_BabkinIvan
MDI221_FadeevSvetoslav_2023
MDI221_Ignatova_Ksenia
MDI222_lab_7_group_9
MDI_2023_Group_Project_4
Схемы
public
Таблицы
Представления
Мат. представления
Индексы
Функции
Последовательности
Типы данных
Агрегатные функции
Событийные триггеры
Расширения
Хранилище
Системные объекты
Роли
MakslvovaMaxHm20231stm
public
*<postgres 4> S
artworks
*<postgres 4> S
artists
customers
WHERE aw.row_num <= 5;
CREATE OR REPLACE VIEW artists_by_medium AS
SELECT
a.art_name AS artist_name,
aw.aw_materials AS medium,
a.art_numb
FROM public.artists a
JOIN public.artworks aw ON a.art_id = aw.art_id
GROUP BY a.art_name, aw.aw_materials, a.art_numb;
Статистика 1
Name
Value
Updated Rows
0
Query
CREATE OR REPLACE VIEW artists_by_medium AS
SELECT
a.art_name AS artist_name,
aw.aw_materials AS medium,
a.art_numb
FROM public.artists a
JOIN public.artworks aw ON a.art_id = aw.art_id
GROUP BY a.art_name, aw.aw_materials, a.art_numb
Finish time
Fri Oct 20 17:17:12 MSK 2023
Обновить
Save
Cancel
Экспорт данных ...
200
0 строк обновлено - 35ms, 2023-10-20 в 17:17:12
MSK
ru_RU
Запись
Инт. вставка
81 : 50 : 2016


artists_by_medium
Enter a SQL expression to filter results (use Ctrl+Space)
artist_name
medium
art_numb
Henri Matisse Oil on canvas 1,007
Andy Warhol Acrylic on canvas 349
Marc Chagall Oil on canvas 1,022
Frida Kahlo Oil on canvas 145
Diego Rivera Fresco 145
Henri Rousseau Oil on canvas 123
Edvard Munch Oil, tempera, and pastel 7,600
Claude Monet Oil on canvas 1,367
Gustav Klimt Oil and gold leaf 161
Georgia OKeeffe Oil on canvas 237
Michelangelo Fresco 182
Joan Miró Oil on canvas 202
Vincent van Gogh Oil on canvas 2,000
Amedeo Modigliani Oil on canvas 355
Salvador Dalí Oil on canvas 1,178
Edgar Degas Oil on canvas 765
Pablo Picasso Oil on canvas 20,000
Paul Cézanne Oil on canvas 820
Jackson Pollock Oil and enamel 117
Leonardo da Vinci Oil on panel 205
Refresh
Save
Cancel
Export data
200
20 row(s) fetched - 11ms, on 2023-10-20 at 01:07:16
MSK
en



	A	B	C	D	E	F	G	H	I	J
1	artist_name	medium	art_numb		Названия строк	Сумма из art_numb				
2	Leonardo da Vinci	Oil on panel	205		Oil and enamel	117				
3	Vincent van Gogh	Oil on canvas	2000		Jackson Pollock	117				
4	Pablo Picasso	Oil on canvas	20000		Oil and gold leaf	161				
5	Claude Monet	Oil on canvas	1367		Gustav Klimt	161				
6	Michelangelo	Fresco	182		Oil on panel	205				
7	Frida Kahlo	Oil on canvas	145		Leonardo da Vinci	205				
8	Salvador Dalí	Oil on canvas	1178		Fresco	327				
9	Georgia OKeeffe	Oil on canvas	237		Michelangelo	182				
10	Andy Warhol	Acrylic on canvas	349		Diego Rivera	145				
11	Edvard Munch	Oil, tempera, and pastel	7600		Acrylic on canvas	349				
12	Gustav Klimt	Oil and gold leaf	161		Andy Warhol	349				
13	Edgar Degas	Oil on canvas	765		Oil, tempera, and pastel	7600				
14	Henri Matisse	Oil on canvas	1007		Edvard Munch	7600				
15	Jackson Pollock	Oil and enamel	117		Oil on canvas	25567				
16	Paul Cézanne	Oil on canvas	820		Pablo Picasso	20000				
17	Diego Rivera	Fresco	145		Vincent van Gogh	2000				
18	Marc Chagall	Oil on canvas	1022		Claude Monet	1367				
19	Joan Miró	Oil on canvas	202		Salvador Dalí	1178				
20	Amedeo Modigliani	Oil on canvas	349		Marc Chagall	1022				
21	Henri Rousseau	Oil on canvas	123		**Общий итог**	**34326**				
22										
23										
24										
25										
26										
27										