

ART GALLERIES DATABASE

Subject area analysis: structural and functional analysis.....	1
ERDPlus schema.....	3
Relational schema.....	3
Diagram in DB.....	5
Example of filling the table with data.....	6
Additional requirements to DB.....	6
Queries.....	8
Function.....	11
Trigger.....	14
Editable view.....	17
Report View.....	24

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.

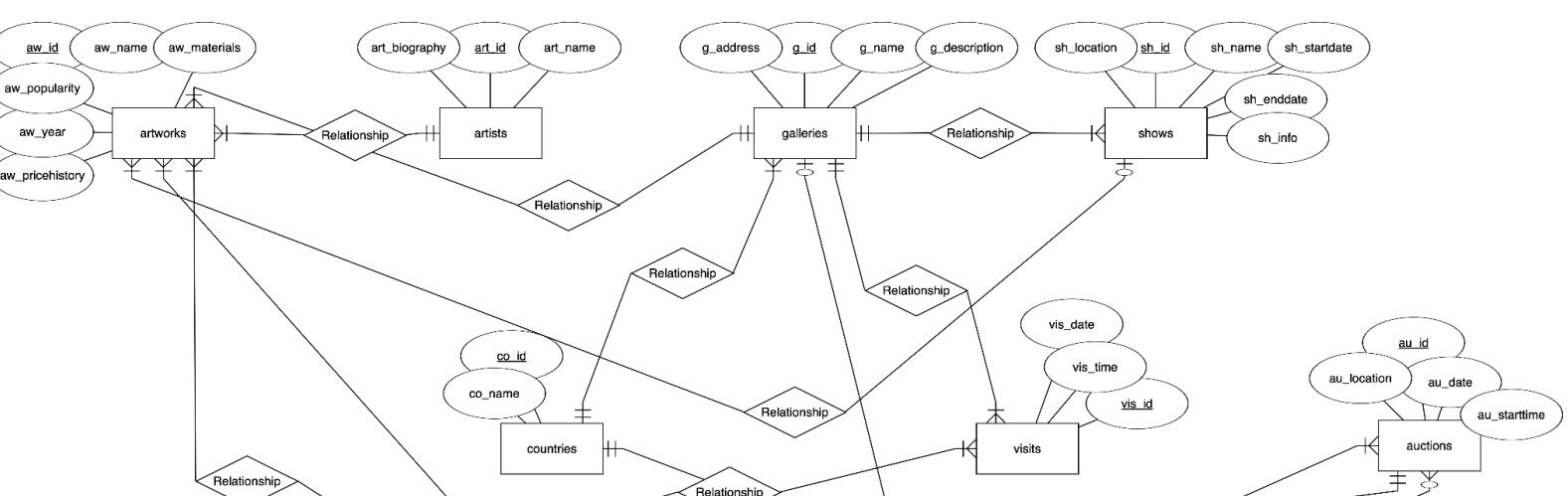
- Detailed Artwork Information;

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.

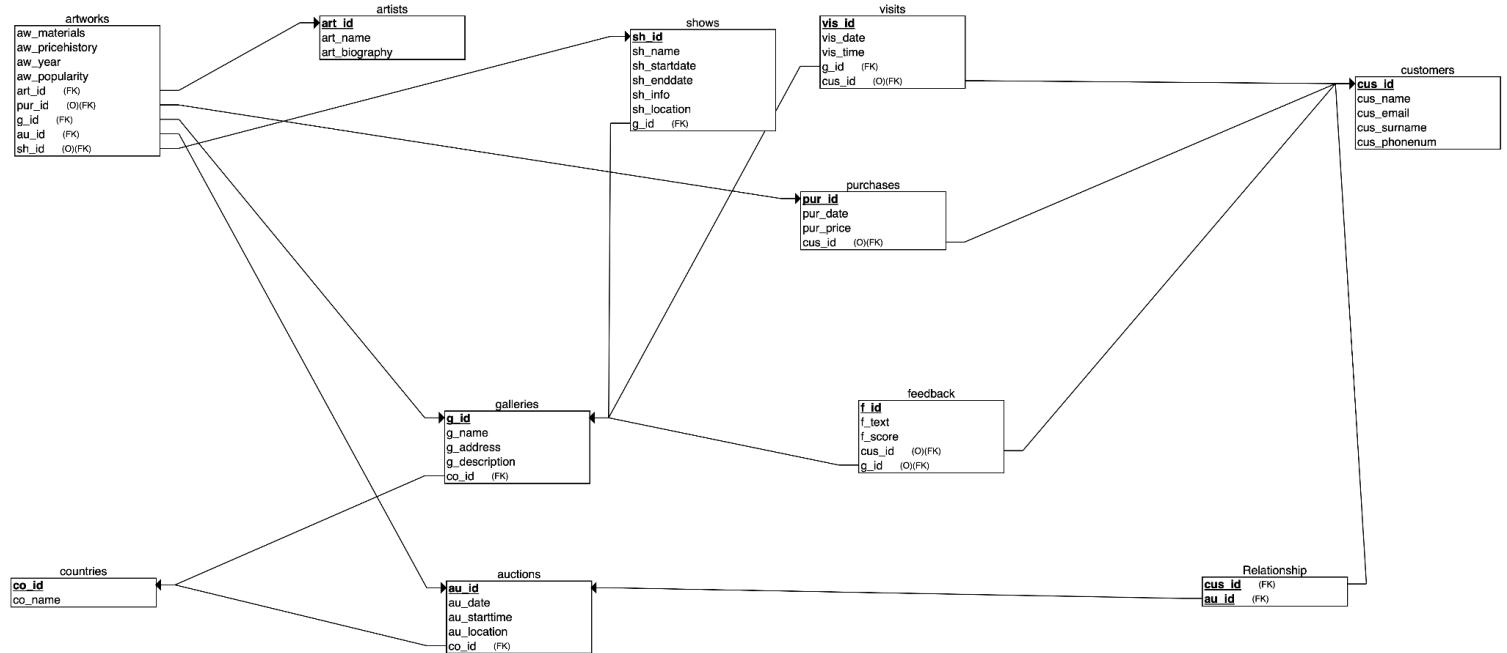
- Management of Exhibitions and Galleries.

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:

Таблица	1 ² cus_id	2 ³ cus_name	3 ⁴ cus_email	4 ⁵ cus_surname	5 ⁶ cus_phonenum
Текст	981	John	aoadljq@mail.ru	Adams	54 699 529 232
	982	Grace	sftqkknu@mail.ru	Hill	23 406 085 481
	983	Anthony	pirmmveh@mail.ru	Foster	44 145 916 708
	984	Victoria	qnwvajzw@mail.ru	Stewart	93 748 158 145
	985	Caleb	wchojdxb@mail.ru	Powell	20 258 702 283
	986	Evelyn	cygssnoh@mail.ru	Cox	54 882 347 528
	987	Samuel	gejhgaot@mail.ru	Ramirez	48 924 224 868
	988	Harper	xzuptyfe@mail.ru	Evans	14 543 418 386
	989	Daniel	wkqevrxe@mail.ru	Reed	60 677 508 244
	990	Ella	qqowvysb@mail.ru	Price	20 326 087 062
	991	Dylan	nuibhtnb@mail.ru	Cooper	77 163 758 460
	992	Natalie	cjzyibn@mail.ru	Ward	96 415 496 986
	993	Matthew	eulnmtzn@mail.ru	Coleman	19 279 732 686
	994	Scarlett	pdvpirk@mail.ru	Morris	28 144 751 624
	995	Christopher	jklxbsgc@mail.ru	Hayes	15 853 573 002
	996	Harper	fbseorib@mail.ru	Parker	62 683 625 045
	997	Joseph	fibuurtp@mail.ru	Butler	99 863 322 416
	998	Madison	eloyfayz@mail.ru	Price	79 026 668 404
	999	William	bxyawoip@mail.ru	Turner	35 637 394 896
	1 000	Mia	rmxtzsys@mail.ru	Hayes	20 510 580 324

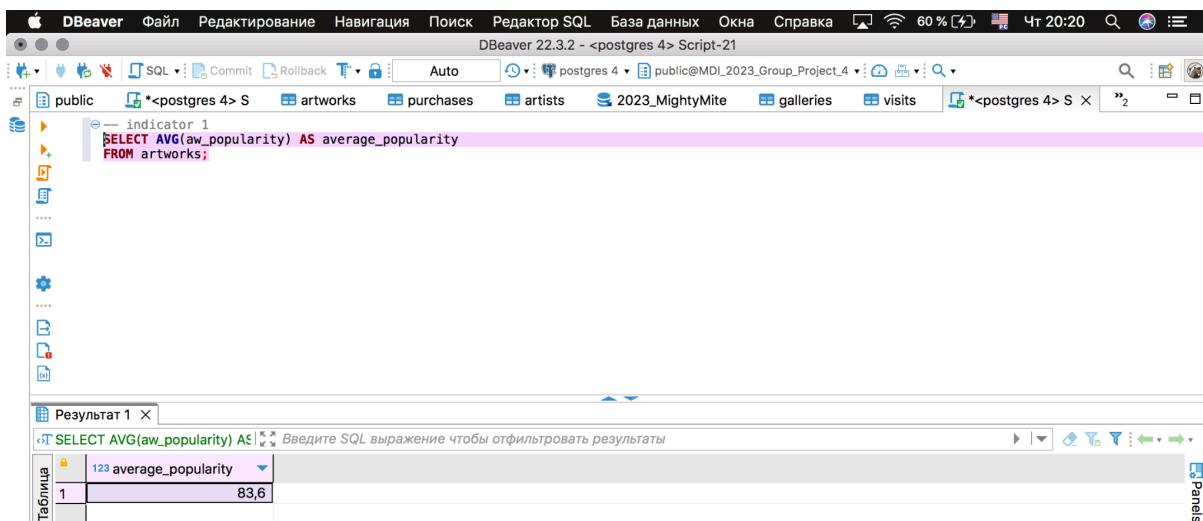
Additional requirements to DB:

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;



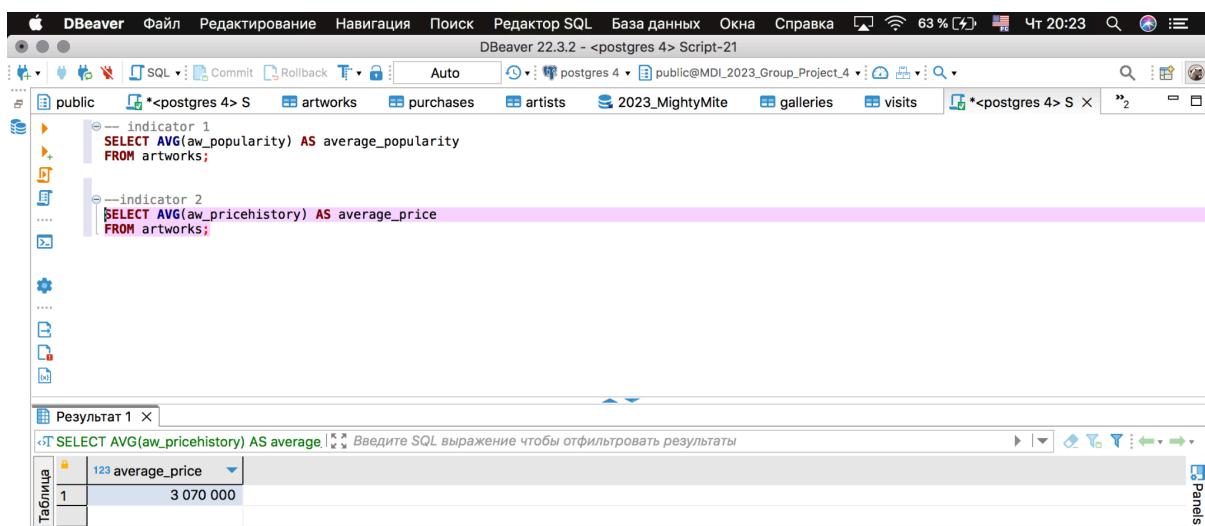
The screenshot shows the DBeaver interface with the following details:

- Toolbar:** DBeaver 22.3.2 - <postgres 4> Script-21, Auto mode, connected to postgres 4.
- Sidebar:** Shows the schema tree with public, artworks, purchases, artists, 2023_MightyMite, galleries, visits, and a script named indicator 1.
- SQL Editor:** Contains the SQL query: `SELECT AVG(aw_popularity) AS average_popularity FROM artworks;`.
- Result Table:** A single row labeled "average_popularity" with value 83,6.

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

```
SELECT AVG(aw_pricehistory) AS average_price
```

FROM artworks;



The screenshot shows the DBeaver interface with the following details:

- Toolbar:** DBeaver 22.3.2 - <postgres 4> Script-21, Auto mode, connected to postgres 4.
- Sidebar:** Shows the schema tree with public, artworks, purchases, artists, 2023_MightyMite, galleries, visits, and two scripts named indicator 1 and indicator 2.
- SQL Editor:** Contains the SQL query: `SELECT AVG(aw_pricehistory) AS average_price FROM artworks;`.
- Result Table:** A single row labeled "average_price" with value 3 070 000.

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

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

```
FROM artworks;
```

The screenshot shows the DBeaver interface with the following details:

- Toolbar:** Includes File, Редактирование (Edit), Навигация (Navigation), Поиск (Search), Редактор SQL (SQL Editor), База данных (Database), Окна (Windows), Справка (Help), and system icons for battery level (81%), signal strength, and date/time (Чт 20:39).
- SQL Editor:** Shows three SQL statements under a script named "Script-21".
 - Indicator 1: `SELECT AVG(aw_popularity) AS average_popularity FROM artworks;`
 - Indicator 2: `SELECT AVG(aw_pricehistory) AS average_price FROM artworks;`
 - Indicator 3: `SELECT AVG(EXTRACT(YEAR FROM aw_year)) AS avg_year FROM artworks;` (highlighted in pink)
- Results Panel:** Labeled "Результат 1" (Result 1). It contains a table with one row:

Таблица	avg_year
1	1878

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');
```

aw_id	aw_name	aw_materials	aw_pricehistory	aw_popularity	art_id	pic
7	The Persistence of Oil on canvas	3 500 000	1931-01-01	87	7	7

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();
```

The screenshot shows the pgAdmin interface with two SQL scripts open. The top script contains the code for creating a function named 'log_artwork_movement'. The bottom script contains the code for creating a trigger named 'artwork_movement_trigger' that triggers after an update on the 'sh_id' column of the 'public.artworks' table, executing the 'log_artwork_movement' function when the old and new values are distinct. Below the scripts, a 'Statistics 1' tab is visible, showing 'Updated Rows 0' and a query history with the same trigger creation command. At the bottom, system information like 'Start time' and 'Thu Oct 19 17:08:07 MSK 2023' is displayed.

```
MDI_2023_Group_Project_4 <hse dma 2021> Script-1 public

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();

Statistics 1
Name | Value
Updated Rows 0
Query 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()
Start time Thu Oct 19 17:08:07 MSK 2023
```

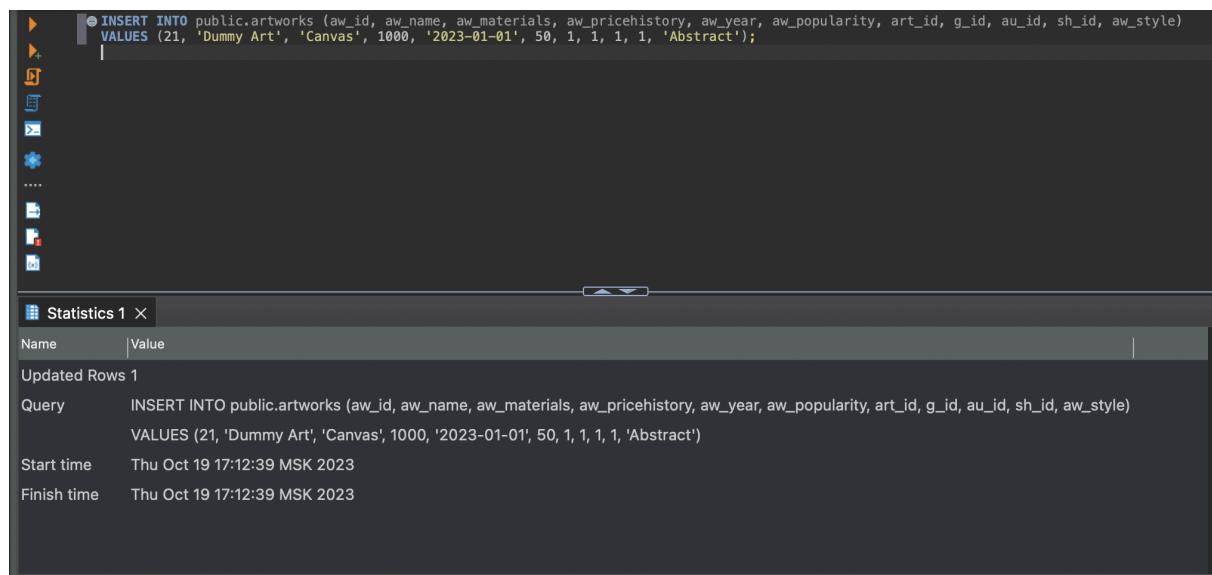
What happens:

- 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');
```



The screenshot shows the pgAdmin interface with a query editor and a statistics panel. The query editor contains the following SQL code:

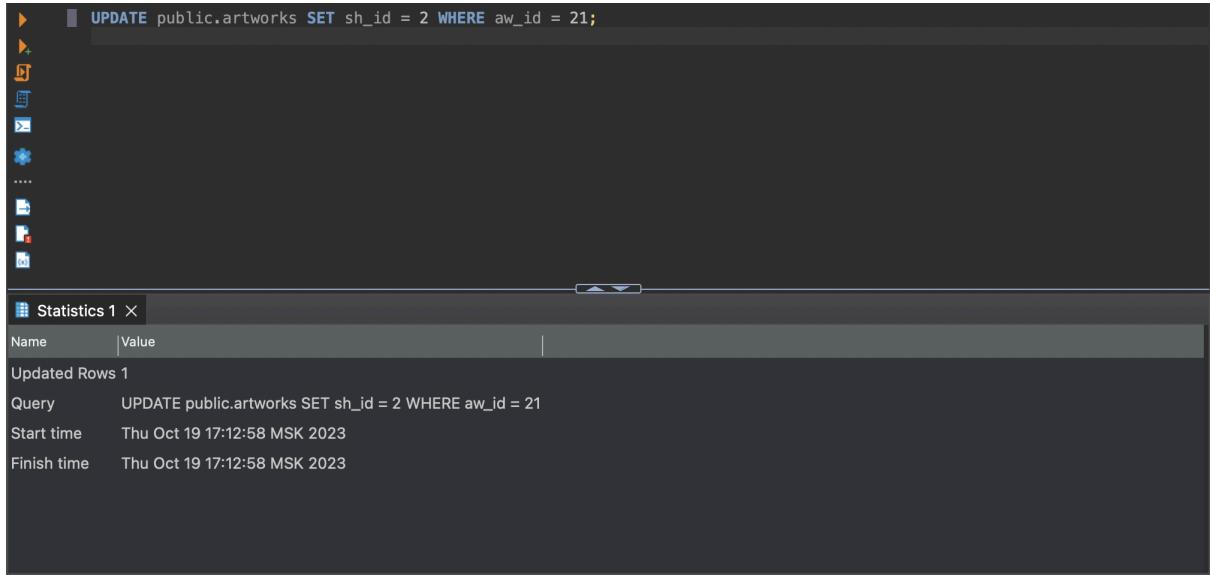
```
INSERT INTO public.artworks (aw_id, aw_name, aw_materials, aw_pricehistory, aw_year, aw_popularity, art_id, g_id, au_id, sh_id, aw_style)
VALUES (21, 'Dummy Art', 'Canvas', 1000, '2023-01-01', 50, 1, 1, 1, 1, 'Abstract');
```

The statistics panel below shows the following information:

Name	Value
Updated Rows	1
Query	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')
Start time	Thu Oct 19 17:12:39 MSK 2023
Finish time	Thu Oct 19 17:12:39 MSK 2023

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

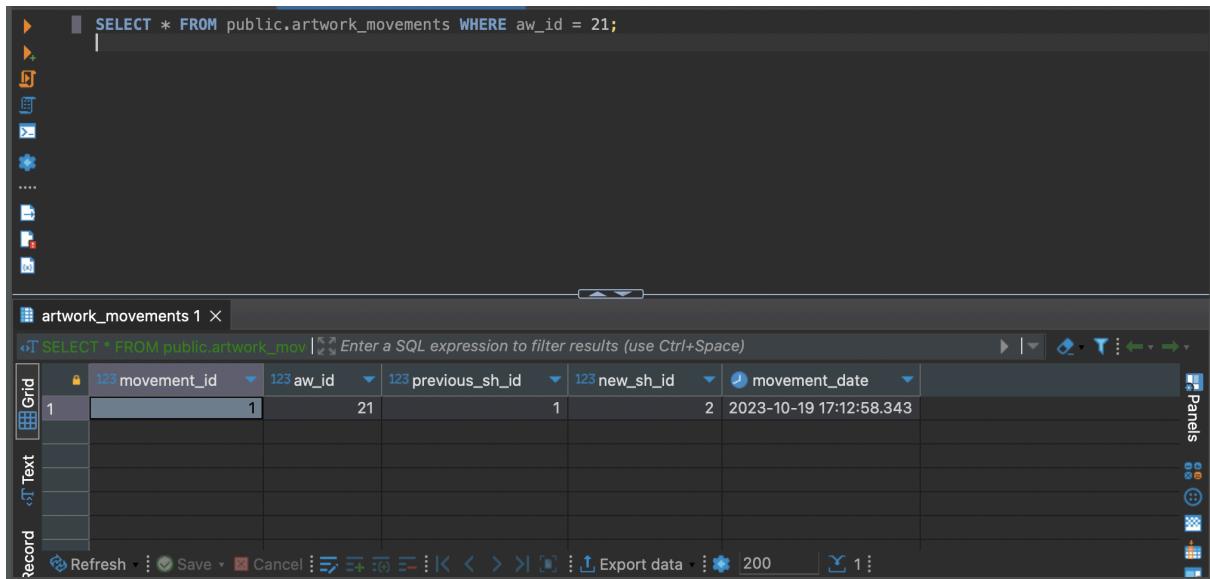
```
UPDATE public.artworks SET sh_id = 2 WHERE aw_id = 21;
```



The screenshot shows a PostgreSQL client interface. In the main query editor window, the command `UPDATE public.artworks SET sh_id = 2 WHERE aw_id = 21;` is entered. Below the editor, a "Statistics" panel displays the results of the query: "Updated Rows 1". Underneath the statistics, it shows the query itself and its execution details: "Start time" and "Finish time" both listed as "Thu Oct 19 17:12:58 MSK 2023".

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

```
SELECT * FROM public.artwork_movements WHERE aw_id = 21;
```



The screenshot shows a PostgreSQL client interface. In the main query editor window, the command `SELECT * FROM public.artwork_movements WHERE aw_id = 21;` is entered. Below the editor, a "artwork_movements" result grid is displayed. The grid has columns: movement_id, aw_id, previous_sh_id, new_sh_id, and movement_date. A single row is visible, showing values: 1, 21, 1, 2, and 2023-10-19 17:12:58.343 respectively.

	movement_id	aw_id	previous_sh_id	new_sh_id	movement_date
1	1	21	1	2	2023-10-19 17:12:58.343

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();
```

The screenshot shows a database management interface with two main panes. The top pane displays a script editor with the following SQL code:

```
-- 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();
```

The bottom pane shows the execution results. It includes a table for statistics and a detailed log of the executed triggers:

Name	Value
Updated Rows	0

Query details:

```
-- 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();
```

```
-- 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();
```

```
-- 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()
```

Execution timeline:

Start time	Finish time
Thu Oct 19 23:43:04 MSK 2023	Thu Oct 19 23:43:04 MSK 2023

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;
```

The screenshot shows the pgAdmin 4 interface with two tabs visible: 'Script' and 'Statistics'. The 'Script' tab contains the PostgreSQL function code provided above. The 'Statistics' tab shows the execution results:

Name	Value
Updated Rows	0
Query	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;
Start time	Thu Oct 19 23:39:30 MSK 2023
Finish time	Thu Oct 19 23:39:30 MSK 2023

● 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();
```

The screenshot shows a database interface with a script editor and a results table. The script editor contains the SQL code for creating a trigger. The results table shows the trigger was created successfully with 0 updated rows. The query section also displays the trigger's definition.

Name	Value
Updated Rows	0
Query	-- 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(); -- 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(); -- 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()
Start time	Thu Oct 19 23:43:04 MSK 2023
Finish time	Thu Oct 19 23:43:04 MSK 2023

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;

```

```

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;

```

Name	Value
Updated Rows	0
Query	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 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();
```

The screenshot shows a database interface with a script editor and a results table. The script editor contains the SQL code for creating a trigger. The results table shows the trigger definition and execution details.

Name	Value
Updated Rows	0
Query	<pre>-- 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(); -- 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(); -- 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()</pre>
Start time	Thu Oct 19 23:43:04 MSK 2023
Finish time	Thu Oct 19 23:43:04 MSK 2023

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;
```

The screenshot shows a PostgreSQL IDE interface with two main panes. The top pane displays the SQL code for creating a trigger function. The bottom pane shows the results of the query execution, including the number of updated rows (0), start and finish times, and statistics.

```
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;
```

Name	Value
Updated Rows	0
Query	<pre>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</pre>
Start time	Thu Oct 19 23:42:38 MSK 2023
Finish time	Thu Oct 19 23:42:38 MSK 2023

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;
```

Screenshot of DBBeaver 22.3.2 interface showing a PostgreSQL database connection.

Left Panel (Bases de datos):

- MDI_2023_Group
 - HW1_Atoian
 - HW1_MDI221
 - HW1_MDI221_Temmo
 - Koneva Ekaterina HW5
 - Korenev_HW1
 - Lab7_Shlyoda_T4
 - MDI221_Babbkinivan
 - 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
- Makslisova
- MaxHm20231stm

Central Panel (SQL Editor):

```

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_num
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_num;
  
```

Bottom Panel (Statistics):

Name	Value
Updated Rows	0
Query	<pre> CREATE OR REPLACE VIEW artists_by_medium AS SELECT a.art_name AS artist_name, aw.aw_materials AS medium, a.art_num 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_num;</pre>
Finish time	Fri Oct 20 17:17:12 MSK 2023

Other details: MSK | ru_RU | Запись | Инт. вставка | 81 : 50 : 2016 | 200 | 1 | ... 0 строк обновлено - 35ms, 2023-10-20 в 17:17:12

Screenshot of DBBeaver 22.3.2 interface showing the results of the query in the SQL editor.

Table: artists_by_medium

	artist.name	medium	art.num
1	Henri Matisse	Oil on canvas	1,007
2	Andy Warhol	Acrylic on canvas	349
3	Marc Chagall	Oil on canvas	1,022
4	Frida Kahlo	Oil on canvas	145
5	Diego Rivera	Fresco	145
6	Henri Rousseau	Oil on canvas	123
7	Edvard Munch	Oil, tempera, and pastel	7,600
8	Claude Monet	Oil on canvas	1,367
9	Gustav Klimt	Oil and gold leaf	161
10	Georgia O'Keeffe	Oil on canvas	237
11	Michelangelo	Fresco	182
12	Joan Miró	Oil on canvas	202
13	Vincent van Gogh	Oil on canvas	2,000
14	Amedeo Modigliani	Oil on canvas	355
15	Salvador Dalí	Oil on canvas	1,178
16	Edgar Degas	Oil on canvas	765
17	Pablo Picasso	Oil on canvas	20,000
18	Paul Cézanne	Oil on canvas	820
19	Jackson Pollock	Oil and enamel	117
20	Leonardo da Vinci	Oil on panel	205

Other details: Refresh | Save | Cancel | Export data | 200 | 20 | ... 20 row(s) fetched - 11ms, on 2023-10-20 at 01:07:16 | MSK | en | ...

Excel 100% 21:22

Поиск на листе

Главная Вставка Разметка страницы Формулы Данные Рецензирование Вид

Стили ячеек

Сортировка и фильтр

	A	B	C	D	E	F	G	H	I	J
1	artist_name	medium	art_numb		Название 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 O'Keeffe	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										

Лист1 Готово 100%