



Swansea Art Emporium Design & Implementation

**By
Jay Chaturbhai Kakadiya**

Submitted to the University of Wales Trinity Saint David

**Module: Database Systems
Lecturer: Dr Suresh Shanmugasundaram**

Word Count: 3216

Abstract

The goal of this project is to implement Swansea Art Emporium's strategic plan to create a state-of-the-art online art marketplace. Known for its contributions to Swansea's art scene, the traditional art gallery hopes to expand its influence and modernize its operations by launching a digital portal. The task at hand involves the smooth integration of a robust data infrastructure and the selection of the best database technology to facilitate the intended online art marketplace. To properly manage artists, artworks, categories, customers, orders, and feedback, the project entails thorough consultation. Furthermore, the database solution needs to support situations of concurrent usage and require stored procedures, triggers, transactions, and functions to be implemented in order to facilitate effective database administration. In order to visualise the relationships between entities in a database, it also comes with an entity relationship diagram (ERD).

Table of Contents

1. Introduction	5
2. Evaluation of Database Design and Implementation	6
2.1 List of Entities	6
2.2 Entity Relationship Diagram (ERD)	7
2.3 Constraints.....	12
3. Performance Tuning	15
4. Transactions	17
5. Database Objects	19
5.1 Function	19
5.2 Trigger with Auditing Features.....	19
5.3 Stored Procedure.....	20
6 Conclusion	22
7 Bibliography.....	23
8 Appendix.....	24

List of Tables

Table 1: Description of the ERD	8
--	----------

List of Figures

Figure 1: Entity Relationship Diagram	7
Figure 2: Audit Table and Audit Function	20
Figure 3: Triggers and Testing	20

1. Introduction

Swansea Art Emporium, a prestigious gallery deeply ingrained in Swansea culture, is undergoing a metamorphosis to adjust to the changing landscape of the art world. The art gallery is dedicated to developing a cutting-edge online art marketplace because it recognizes the possibilities of having an online presence. With the help of this project, art enthusiasts will have an immersive platform to see, purchase, and commission works of art from skilled artists all around the United Kingdom. The primary task involves building a strong data infrastructure and choosing a suitable database to coordinate the many elements of the online art market effectively.

To tackle this difficulty, Swansea Art Emporium is looking for advice on how to handle the intricacies of managing artists, organizing artwork and categories, interacting with customers, completing orders, and compiling feedback. The effective execution of the web application depends on the careful planning and management of a database that can accommodate multiple concurrent use cases. To accomplish this, the project includes the development of stored procedures, transactions, triggers, and functions to provide effective database administration and thorough monitoring of all system activity.

2. Evaluation of Database Design and Implementation

This chapter covers our database design including its Entity-Relationship Diagram (ERD), which shows the relationships between different entities.

2.1 List of Entities

The database includes the following 23 tables:

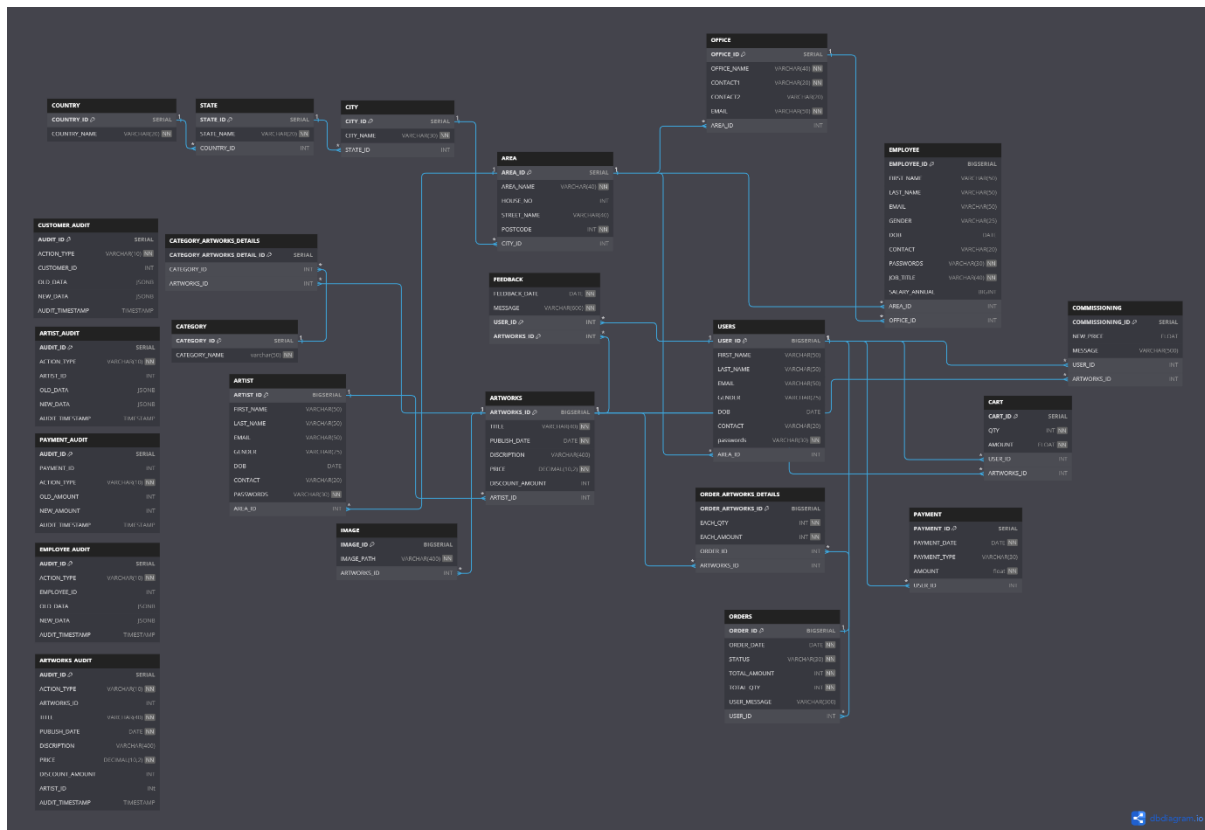
1. Area
2. Artist
3. Artist_audit
4. Artworks
5. Artworks_audit
6. Cart
7. Category
8. Category_artworks_details
9. city
10. Commissioning
11. Country
12. Customer_audit
13. Employee
14. Employee_audit
15. Feedback
16. Image
17. Office
18. Order_artworks_details
19. Orders
20. Payment
21. Payment_audit
22. State
23. Users

2.2 Entity Relationship Diagram (ERD)

ERD is a graphic depiction of the connections between entities in a database. ERDs are frequently used in database design to represent and comprehend a database's structure, showing the relationships between various elements.

Figure 1 shows the ERD of Swansea Art Emporium database.

Figure 1: Entity Relationship Diagram



Description of ERD:

Table 1 shows all the details of entities available in the above figure.

Table 1: Description of the ERD

Table Name	Columns	Constraints	Description
AREA	AREA_ID	PRIMARY KEY	The area table stores the details of the area. One city has many areas, so the id of the city is foreign key.
	AREA_NAME	NOT NULL	
	HOUSE_NO		
	STREET_NAME		
	POSTCODE	NOT NULL	
	CITY_ID	FOREIGN KEY	
ARTIST	ARTIST_ID	PRIMARY KEY	It stores the details of the artist and one artist can create many artworks.
	FIRST_NAME		
	LAST_NAME		
	EMAIL	NOT NULL, CHECK	
	GENDER		
	DOB		
	CONTACT	CHECK	
	PASSWORD	NOT NULL, CHECK	
	AREA_ID	FOREIGN KEY	
ARTIST_AUDIT	AUDIT_ID	PRIMARY KEY	For audit backup, we created an artist_audit table. It will store all the changes happen in artist table.
	ACTION_TYPE	NOT NULL	
	ARTIST_ID	NOT NULL	
	OLD_DATA		
	NEW_DATA		
	AUDIT_TIMESTAMP	DEFAULT	
ARTWORKS	ARTWORKS_ID	PRIMARY KEY	It stores artwork details. A single artist creates a piece of art. A piece of art can fit into more than one category.
	TITLE	NOT NULL	
	DESCRIPTION		
	DISCOUNT_AMOUNT		
	PUBLISH_DATE	NOT NULL	
	PRICE	NOT NULL	
	ARTIST_ID	FOREIGN KEY	
ARTWORKS_AUDIT	AUDIT_ID	PRIMARY KEY	All the changes happen in artwork will stored
	ACTION_TYPE	NOT NULL	

Table Name	Columns	Constraints	Description
	ARTWORKS_ID	NOT NULL	automatically in this audit table.
	TITLE	NOT NULL	
	PUBLISH_DATE	NOT NULL	
	DESCRIPTION	NOT NULL	
	PRICE	NOT NULL	
	DISCOUNT_AMOUNT		
	ARTIST_ID	NOT NULL	
	AUDIT_TIMESTAMP	DEFAULT	
CART	CART_ID	PRIMARY KEY	The cart stores artworks, quantity, and amount until the user places that order.
	QTY	NOT NULL	
	AMOUNT	NOT NULL	
	USER_ID	FOREIGN KEY	
	ARTWORKS_ID	FOREIGN KEY	
CATEGORY	CATEGORY_ID	PRIMARY KEY	Artworks are categorized according to their kind. In one category there are many different artworks.
	CATEGORY_NAME	NOT NULL	
CATEGORY_ARTWORKS_DETAILS	CATEGORY_ARTWORKS_DETAILS_ID	PRIMARY KEY	A category can contain numerous works of art, and a category itself may contain many works of art.
	CATEGORY_ID	FOREIGN KEY	
	ARTWORKS_ID	FOREIGN KEY	
CITY	CITY_ID	PRIMARY KEY	The city table stores details of the city and its particular state ID.
	CITY_NAME	NOT NULL	
	STATE_ID	FOREIGN KEY	
COMMISSIONING	COMMISSIONING_ID	PRIMARY KEY	Users can commission a particular artwork and send a message to the artist through artwork ID.
	NEW_PRICE		
	MESSAGE		
	USER_ID	FOREIGN KEY	
	ARTWORKS_ID	FOREIGN KEY	
COUNTRY	COUNTRY_ID	PRIMARY KEY	This table stores the name of the countries and their unique id. One country has many states.
	COUNTRY_NAME	NOT NULL	
CUSTOMER_AUDIT	AUDIT_ID	PRIMARY KEY	This table stores any changes that happened in the customer table.
	ACTION_TYPE	NOT NULL	
	CUSTOMER_ID	NOT NULL	

Table Name	Columns	Constraints	Description
	OLD_DATA		
	NEW_DATA		
	AUDIT_TIMESTAMP	DEFAULT	
EMPLOYEE	EMPLOYEE_ID	PRIMARY KEY	The employee table stores all the details of the employee such as name, email, mobile, job title, salary, etc. In one office there are many employees.
	FIRST_NAME		
	LAST_NAME		
	EMAIL	NOT NULL, CHECK	
	GENDER		
	DOB		
	CONTACT	NOT NULL, CHECK	
	PASSWORD	NOT NULL, CHECK	
	JOB_TITLE	NOT NULL	
	SALARY_ANNUAL	DEFAULT	
	AREA_ID	FOREIGN KEY	
	OFFICE_ID	FOREIGN KEY	
EMPLOYEE_AUDIT	AUDIT_ID	PRIMARY KEY	Any modifications made to the employee table are stored in this audit table.
	ACTION_TYPE	NOT NULL	
	EMPLOYEE_ID	NOT NULL	
	OLD_DATA		
	NEW_DATA		
	AUDIT_TIMESTAMP	DEFAULT	
FEEDBACK	FEEDBACK_DATE	NOT NULL	Customers can give feedback on artwork. This table contains a <u>composite primary key</u> .
	MESSAGE	NOT NULL	
	USER_ID	PRIMARY KEY, FOREIGN KEY	
	ARTWORKS_ID	PRIMARY KEY, FOREIGN KEY	
IMAGE	IMAGE_ID	PRIMARY KEY	It stores the image ID, path, and ID of related artwork.
	IMAGE_PATH	NOT NULL, CHECK	
	ARTWORKS_ID	FOREIGN KEY	
OFFICE	OFFICE_ID	PRIMARY KEY	One company has many offices in different locations.
	OFFICE_NAME	NOT NULL	
	CONTACT1	NOT NULL, CHECK	
	CONTACT2	CHECK	
	EMAIL	NOT NULL, CHECK	
	AREA_ID	FOREIGN KEY	

Table Name	Columns	Constraints	Description
ORDER_ARTWORKS_DETAILS	ORDER_ARTWORKS_DETAILS_ID	PRIMARY KEY	In one order there can be many artworks and one artwork can be available in many orders. So, this table stores each qty and amount along with the order ID and artwork ID.
	EACH_QTY	NOT NULL	
	EACH_AMOUNT	NOT NULL	
	ORDER_ID	FOREIGN KEY	
	ARTWORKS_ID	FOREIGN KEY	
ORDERS	ORDER_ID	PRIMARY KEY	It will store the order ID, date, status, total amount, total quantity, user message, and user ID.
	ORDER_DATE	NOT NULL	
	STATUS	NOT NULL	
	TOTAL_AMOUNT	NOT NULL	
	TOTAL_QTY	NOT NULL	
	USER_MESSAGE		
	USER_ID	FOREIGN KEY	
PAYMENT	PAYMENT_ID	PRIMARY KEY	This table stores all the details of payments made by users.
	PAYMENT_DATE	NOT NULL	
	PAYMENT_TYPE	CHECK	
	AMOUNT	NOT NULL	
	USER_ID	FOREIGN KEY	
PAYMENT_AUDIT	AUDIT_ID	PRIMARY KEY	All the actions performed in the payment table are stored in this audit table.
	PAYMENT_ID	NOT NULL	
	ACTION_TYPE	NOT NULL	
	OLD_AMOUNT		
	NEW_AMOUNT		
	AUDIT_TIMESTAMP	DEFAULT	
STATE	STATE_ID	PRIMARY KEY	It stores all the states with its particular country ID. One country has many states and one state has many city.
	STATE_NAME	NOT NULL	
	COUNTRY_ID	FOREIGN KEY	
USERS	USER_ID	PRIMARY KEY	All the details of the customers are stored in this table.
	FIRST_NAME		
	LAST_NAME		
	EMAIL	NOT NULL, CHECK	
	GENDER		
	DOB		

Table Name	Columns	Constraints	Description
	CONTACT	CHECK	
	PASSWORD	NOT NULL, CHECK	
	AREA_ID	FOREIGN KEY	

2.3 Constraints

Constraints are used to enforce rules on the data in a database, ensuring data integrity and consistency. These constraints specify guidelines and requirements that the data must follow, avoiding the addition of inaccurate or inconsistent data.

➤ Primary Key

This guarantees that a selected primary key uniquely identifies every record in a table. It prevents duplicate entries, promoting data accuracy and speedy data retrieval.

We use the primary key in every table available in this database [2].

Example:

```
--CREATING COUNTRY TABLE
CREATE TABLE COUNTRY(
COUNTRY_ID SERIAL PRIMARY KEY,
COUNTRY_NAME VARCHAR(20) NOT NULL
);
```

In the above example, we create a country table, and **COUNTRY_ID** is the Primary key.

➤ Foreign Key

This constraint establishes relationships between tables by linking a field in one table to the primary key in another. Foreign key constraints maintain referential integrity, enforcing the connection between related data and preventing or cascading changes that could compromise relationships [1].

Example:

```
--CREATING STATE TABLE
CREATE TABLE STATE(
STATE_ID SERIAL PRIMARY KEY,
STATE_NAME VARCHAR(20) NOT NULL,
COUNTRY_ID INT REFERENCES COUNTRY(COUNTRY_ID));
```

Here, **COUNTRY_ID** in the **STATE** table is a foreign key that references the **COUNTRY_ID** in the **COUNTRY** table.

➤ Check

A check constraint is a kind of constraint that lets us set a requirement that needs to be met for every table entry.

Each Password field is subject to the Check() constraint. A minimum of eight characters, one capital and one lowercase letter, one digit, one special character (@, #, \$, &, etc.), and no spaces are required in every password [3].

```
--Check Constraint for password
ALTER TABLE USERS ADD CONSTRAINT CHK_PASS CHECK (passwords like '%[0-9]%' or
passwords like '%[A-Z]%' or passwords like '%[!@#$$a^&*()-_+=.,;:''\"`~]%' or
length(passwords) >= 8)
--NO SPACE IN PASSWORD
ALTER TABLE USERS ADD CONSTRAINT chkNoSpaces
CHECK (PASSWORDS NOT LIKE '% %');
```

Every Email and contact fields also have check constraint.

```
--CHECK EMAIL
ALTER TABLE USERS ADD CHECK(EMAIL LIKE '%_@_%._%');
SELECT *FROM USERS;
--CHECK CONTACT
ALTER TABLE USERS ADD CHECK(CONTACT LIKE '+[0-9]%' OR CONTACT LIKE'% %'),
```

➤ Not Null

These ensure that essential information is always there by stating that a given column cannot contain null (missing or undefined) values.

Example:

```
--CREATING CITY TABLE
CREATE TABLE CITY(
CITY_ID SERIAL PRIMARY KEY,
CITY_NAME VARCHAR(30) NOT NULL,
STATE_ID INT REFERENCES STATE(STATE_ID)
);
```

In this example, **CITY_NAME** cannot have NULL values.

➤ Default

A database management system (DBMS) feature called a Default Constraint lets you set a default value for a column in a table. If no explicit value is entered when a new row is inserted, this default value is automatically set to the column.

Example:

```
CREATE TABLE ARTWORKS_AUDIT (  
    AUDIT_ID SERIAL PRIMARY KEY,  
    ACTION_TYPE VARCHAR(10) NOT NULL,  
    ARTWORKS_ID INT NOT NULL,  
    TITLE VARCHAR(40) NOT NULL,  
    PUBLISH_DATE DATE NOT NULL,  
    DISCRIPTION VARCHAR(400),  
    PRICE DECIMAL(10, 2) NOT NULL,  
    DISCOUNT_AMOUNT INT,  
    ARTIST_ID INT NOT NULL,  
    AUDIT_TIMESTAMP TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

In this example, **AUDIT_TIMESTAMP** column by default saves the current time.

3. Performance Tuning

In the context of database management and software systems, performance tuning is the act of enhancing a system's functionality to increase its scalability, efficiency, and responsiveness. Enhancing the system's capacity to manage a heavier demand, deliver quicker reaction times, and make better use of its resources is the aim.

Key aspects of performance tuning:

Database Optimisation: Creating or adjusting indexes, fine-tuning database parameters, and query optimisation are common steps in performance optimisation for database management systems (DBMS). Performance of queries is enhanced by well-designed database schemas and normalised structures.

Indexing Techniques: A well-designed index can greatly improve query performance. Performance tuning requires selecting the appropriate columns to index and updating indexes on a regular basis [4-5].

Determining the most important indexes for the tables in the Swansea Art Emporium database would depend on the specific queries and operations frequently performed on the data. Below are some potential indexes based on common use cases:

1. Artworks table:

Artworks_id Primary Key Index: As the main index, it is essential for quickly obtaining certain artworks.

Artist_id index : If there are a lot of inquiries based on artist information.

```
-- Artworks Table
CREATE INDEX idx_artworks_artworkid ON Artworks(artworks_id);
CREATE INDEX idx_artworks_artistid ON Artworks(artist_id);
```

2. Cart Table:

Composite Index on User_id and Artworks_id: Useful for quickly retrieving the contents of a user's shopping cart.

Index on Cart_id (Primary Key): Essential for efficient retrieval of specific cart details.

```
-- Cart Table
CREATE INDEX idx_cart_userid_artworkid ON Cart(UserID, artworks_id);
CREATE INDEX idx_cart_cartid ON Cart(cart_id);
```

3. Category_artworks_details Table:

Composite Index on Category_id and Artworks_id: Important for efficiently fetching artworks belonging to a specific category.

```
CREATE INDEX idx_cat_art_details_categoryid_artworkid ON
Category_artworks_details(Category_id, artworks_id);
```

4. Orders Table:

Composite Index on User_id and Order_id: Useful for quickly retrieving the contents of a user's order.

Primary Key Index on Order_id: Vital for quick access to specific orders.

Index on User_id: Useful for retrieving all orders placed by a particular user.

```
---- Create index for orders table----
CREATE INDEX idx_order_user ON orders (order_id,user_id);
CREATE INDEX idx_orders_orderid ON Orders(order_id);
CREATE INDEX idx_orders_userid ON Orders(user_id);
```

5. Payment Table:

Composite Index on User_id and Payment_id: Useful for quickly retrieving the contents of a user's payment.

Primary Key Index on Payment_id: Essential for fast retrieval of payment details.

Index on User_id: Important for linking payments to specific users.

```
CREATE INDEX idx_payment_user ON PAYMENT (payment_id,user_id,);
CREATE INDEX idx_payment_paymentid ON Payment(payment_id);
CREATE INDEX idx_payment_paymentid ON Payment(user_id);
```

6. Users Table:

Primary Key Index on UserID: Critical for quick access to user details.

Index on Email (assuming uniqueness): Useful for fast user authentication based on email.

```
---- Hash index on user id and email for quick customer lookups
CREATE INDEX idx_customers_id ON users USING HASH (user_id);
CREATE INDEX idx_customers_email ON users USING HASH (email);
```

Determining which index is more important depends on the specific queries that are more frequent and critical for the application. For instance, if the application frequently retrieves artworks based on categories, the composite index on CategoryID and ArtworkID in the Category_artworks_details table might be crucial. Similarly, the primary key index on OrderID in the Orders table would be vital if order-related queries are common.

Ultimately, the importance of an index is subjective and context-dependent. Regularly monitoring query performance and adjusting indexes based on actual usage patterns is a good practice for ongoing performance optimization.

4. Transactions

A logical unit of work consisting of one or more database operations or actions is called a transaction. To guarantee the consistency, integrity, and dependability of the database, these processes are bundled together. Transactions provide a way to guarantee that a series of related operations either all succeed or, if an error occurs, are entirely rolled back to their previous state, maintaining the database's integrity.

The properties of the transactions include Atomicity, Consistency, Isolation, and Durability (ACID).

Atomicity: A transaction is handled as a single, indivisible unit of work because to its characteristics. Either every operation in the transaction is successfully finished, or none of it is applied to the database. The entire transaction is rolled back if any portion of it fails [5].

Consistency: Transactions bring the database from one consistent state to another. Before and after the transaction, the database needs to meet a number of integrity requirements. A transaction is rolled back if it breaks any integrity restrictions [5].

Isolation: This attribute makes sure that a transaction's intermediate state is hidden from other transactions until it is committed. By preventing interference, isolation enables concurrent transactions to function independently of one another [5].

Durability: A transaction's consequences are irreversible once it is completed and survive even system breakdowns and power outages. Even in the case of a system failure, the modifications made by a committed transaction are kept in the database [5].

Applying transactions to the Swansea Art Emporium database involves wrapping a series of SQL statements within a transaction block. In PostgreSQL, transactions are initiated with the **BEGIN** statement and can be concluded with either **COMMIT** or **ROLLBACK** depending on whether the operations should be permanently saved or discarded.

In this database, we applied transactions for payment as it is an important part of the payment process.

```
----- *** Apply Transactions on PAYMENT
LANGUAGE plpgsql
AS $$
BEGIN
    -- Start the transaction
    BEGIN
        -- Update the payment record
        UPDATE PAYMENT
        SET AMOUNT = p_amount,
            PAYMENT_DATE = p_payment_date,
            PAYMENT_TYPE = p_payment_type,
            USER_ID = p_user_id
        WHERE PAYMENT_ID = p_payment_id;
```

```
-- Insert audit record for the update
INSERT INTO PAYMENT_AUDIT (PAYMENT_ID, ACTION_TYPE, OLD_AMOUNT,
NEW_AMOUNT)
VALUES (p_payment_id, 'UPDATE', (SELECT AMOUNT FROM PAYMENT WHERE
PAYMENT_ID = p_payment_id), p_amount);

-- Commit the transaction
COMMIT;

EXCEPTION
-- Handle exceptions and rollback the transaction if an error occurs
WHEN OTHERS THEN
    RAISE NOTICE 'Transaction failed: %', SQLERRM;
    ROLLBACK;
    RETURN;

END;
END;
$$;
```

In the above example, we can only see the queries of transactions, but we applied triggers and function with auditing features.

5. Database Objects

Functions, triggers, and stored procedures are database objects that can be utilized in the context of transactions, but they themselves are not transactions.

5.1 Function

Functions in a database can be a component of a transaction and are usually created to carry out a particular activity or computation. We created functions on **commissioning**, **artist**, **users**, **payment**, **employee**, and **artworks** tables.

The below function checks if an artist has a commission request or not. It will give us either 'TRUE' or 'False' [6][5].

```
----- ***** Function to Check if an Artist Has Commission Requests ***** -----
-----
CREATE FUNCTION HasCommissionRequests(artist_id INT)
RETURNS BOOLEAN
LANGUAGE PLPGSQL
AS
$$
    DECLARE
        commissionRequestsExist BOOLEAN;
BEGIN
    SELECT COUNT(*) > 0
    INTO commissionRequestsExist
    FROM commissioning
    WHERE artist_id = artist_Id;

    RETURN commissionRequestsExist;
END;
$$
```

5.2 Trigger with Auditing Features

A database component known as a trigger with auditing functionality combines audit trails with triggers to record and capture modifications made to particular tables. The major objective is to keep track of data alterations for auditing, compliance, and change history purposes.

We created triggers with auditing features on **artworks**, **employee**, **users**, **artist**, and **payment** tables. Below Figure 2 and Figure 3 shows the example of triggers with auditing function we apply on the **artworks** table [5].

Figure 2: Audit Table and Audit Function

```

-----+ ***** Create trigger with auditing function on ARTWORKS *****-----
----- 1.Create the audit_log_artworks table if it doesn't exist-----
CREATE TABLE ARTWORKS_AUDIT (
  AUDIT_ID SERIAL PRIMARY KEY,
  ACTION_TYPE VARCHAR(10) NOT NULL,
  ARTWORKS_ID INT NOT NULL,
  TITLE VARCHAR(40) NOT NULL,
  PUBLISH_DATE DATE NOT NULL,
  DISCRIPTION VARCHAR(400),
  PRICE DECIMAL(10, 2) NOT NULL,
  DISCOUNT_AMOUNT INT,
  ARTIST_ID INT NOT NULL,
  AUDIT_TIMESTAMP TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

----- 2.Create a trigger function for INSERT, UPDATE, and DELETE operations on artworks-----
CREATE OR REPLACE FUNCTION audit_artworks()
RETURNS TRIGGER AS $$
BEGIN
  IF TG_OP = 'INSERT' THEN
    INSERT INTO ARTWORKS_AUDIT (ACTION_TYPE, ARTWORKS_ID, TITLE, PUBLISH_DATE,DISCRIPTION,PRICE,DISCOUNT_AMOUNT,ARTIST_ID)
    VALUES ('INSERT', NEW.ARTWORKS_ID, NEW.TITLE, NEW.PUBLISH_DATE,NEW.DISCRIPTION,NEW.PRICE,NEW.DISCOUNT_AMOUNT,NEW.ARTIST_ID);
  ELSIF TG_OP = 'UPDATE' THEN
    INSERT INTO ARTWORKS_AUDIT (ACTION_TYPE, ARTWORKS_ID, TITLE, PUBLISH_DATE,DISCRIPTION,PRICE,DISCOUNT_AMOUNT,ARTIST_ID)
    VALUES ('UPDATE', NEW.ARTWORKS_ID, NEW.TITLE, NEW.PUBLISH_DATE,NEW.DISCRIPTION,NEW.PRICE,NEW.DISCOUNT_AMOUNT,NEW.ARTIST_ID);
  ELSIF TG_OP = 'DELETE' THEN
    INSERT INTO ARTWORKS_AUDIT (ACTION_TYPE, ARTWORKS_ID, TITLE, PUBLISH_DATE,DISCRIPTION,PRICE,DISCOUNT_AMOUNT,ARTIST_ID)
    VALUES ('DELETE', OLD.ARTWORKS_ID, OLD.TITLE, OLD.PUBLISH_DATE,OLD.DISCRIPTION,OLD.PRICE,OLD.DISCOUNT_AMOUNT,OLD.ARTIST_ID);
  END IF;
  RETURN NULL;
  -- Commit the transaction
  COMMIT;
END;
$$ LANGUAGE plpgsql;

```

Figure 3: Triggers and Testing

```

--create trigger for insert
CREATE TRIGGER artworks_audit_insert
AFTER INSERT ON ARTWORKS
FOR EACH ROW
EXECUTE FUNCTION audit_artworks();

--create trigger for update
CREATE TRIGGER artworks_audit_update
AFTER UPDATE ON ARTWORKS
FOR EACH ROW
EXECUTE FUNCTION audit_artworks();

--create trigger for delete
CREATE TRIGGER artworks_audit_delete
AFTER DELETE ON ARTWORKS
FOR EACH ROW
EXECUTE FUNCTION audit_artworks();

-- 3. Testing
✓ INSERT INTO ARTWORKS (PUBLISH_DATE,TITLE,DISCRIPTION,PRICE,DISCOUNT_AMOUNT,ARTIST_ID) VALUES
  ('12/12/2023', 'oil Painting', 'A beautiful painting', 5000.00, 50, 7);
select *from ARTWORKS_AUDIT

```

5.3 Stored Procedure

Stored procedures are sets of precompiled SQL statements that are stored in the database and can be executed by calling the procedure. Stored procedures can be designed to perform complex operations that may involve multiple SQL statements within a transaction [5].

We create a stored procedure to fetch the informations of all customers, employees, and artists.

```

-----***** Create a stored procedure to fetch information of all customers,
artists, and employees ***** -----
CREATE OR REPLACE PROCEDURE get_all_users_info()
LANGUAGE plpgsql
AS $$
DECLARE
    customer_data RECORD;
    artist_data RECORD;
    employee_data RECORD;
BEGIN
    -- Fetch information for all customers
    FOR customer_data IN (SELECT * FROM users)
    LOOP
        -- Output or process customer data as needed
        RAISE NOTICE 'Customer ID: %, First Name: %, First Name: %, Email: %',
customer_data.user_id, customer_data.first_name, customer_data.last_name,
customer_data.email;
    END LOOP;

    -- Fetch information for all artists
    FOR artist_data IN (SELECT * FROM ARTIST)
    LOOP
        -- Output or process artist data as needed
        RAISE NOTICE 'Artist ID: %, First Name: %, Last Name: %',
artist_data.artist_id, artist_data.first_name, artist_data.last_name;
    END LOOP;

    -- Fetch information for all employees
    FOR employee_data IN (SELECT * FROM EMPLOYEE)
    LOOP
        -- Output or process employee data as needed
        RAISE NOTICE 'Employee ID: %, First Name: %, Last Name: %, Position:
%, Email: %', employee_data.employee_id, employee_data.first_name,
employee_data.last_name, employee_data.job_title, employee_data.email;
    END LOOP;
END;
$$;

-- Call the stored procedure
CALL get_all_users_info();

```

6 Conclusion

To sum up, the Swansea Art Emporium database offers an extensive and organised way to handle the complexities of running an art emporium. The Entity-Relationship Diagram (ERD) was used to support the evaluation of the database design, which showed a well-thought-out arrangement of tables and relationships that successfully captures the various facets of the art business.

The Entity Relationship Diagram (ERD) demonstrated the connections among many tables, including Area, Artist, Artworks, Cart, Category, Commissioning, Orders, and so on. Every table is essential to the database's general operation, maintaining data integrity, and enabling quick and easy access to data.

The importance of transactions in maintaining the consistency and integrity of data was covered. The atomicity, consistency, isolation, and durability of operations are guaranteed by the ACID qualities, which are sustained by transactions and are essential to preserving the database's dependability.

Furthermore, the application of database objects like stored procedures, triggers, and functions was investigated. These objects improve the usefulness of the database by encapsulating tasks, handling events, and automating business logic. Notably, auditing triggers were offered as a way to keep an audit trail and track modifications, which promoted compliance and responsibility.

In summary, the Swansea Art Emporium database, with its well-designed schema, performance enhancements, robust transaction handling, and strategic use of database objects, stands as a resilient and scalable solution tailored to meet the demands of a dynamic art emporium environment.

7 Bibliography

[1]Cristina Marinescu, “Discovering the Objectual Meaning of Foreign KeyConstraints in Enterprise Applications,” *researchgate.net*, Nov. 2007.

https://www.researchgate.net/publication/4297128_Discovering_the_Objectual_Meaning_of_Foreign_Key_Constraints_in_Enterprise_Applications#fullTextFileContent

[2]CHAITANYA SINGH, “DBMS unit 2 - NOTES DBMS,” *Studocu.com*, 2021.

<https://www.studocu.com/in/document/oriental-institute-of-science-and-technology/analog-and-digital-communication/dbms-unit-2-notes-dbms/38905150>

[3]Hardik Gupta, “Constraints in DBMS,” *www.tutorialspoint.com*, Sep. 12, 2023.

<https://www.tutorialspoint.com/constraints-in-dbms>

[4]E. Bertino *et al.*, *Indexing Techniques for Advanced Database Systems*. Springer Science & Business Media, 2012. Accessed: Dec. 20, 2023. [Online]. Available:

https://books.google.co.uk/books?hl=en&lr=&id=XqLhBwAAQBAJ&oi=fnd&pg=PA1&dq=indexing+techniques+in+sql&ots=xT_NIF7Af3&sig=q4_ZZnM6vxqyGn5i1f_gnkIO-Ak&redir_esc=y#v=onepage&q=indexing%20techniques%20in%20sql&f=false

[5]Tutorialspoint, “PostgreSQL Tutorial - Tutorialspoint,” *www.tutorialspoint.com*, 2023.

<https://www.tutorialspoint.com/postgresql/index.htm>

[6]sschung, “Views, Stored Procedures, Functions, and Triggers.”

<https://eecs.csuohio.edu/~sschung/CIS408/ViewSPTriggerMySQL.pdf> (accessed Dec. 20, 2023).

8 Appendix

Data Generator Tool:

Mockaroo:- <https://www.mockaroo.com/>

ChatGPT:- <https://chat.openai.com/>

Online Tutorial:

TutorialsPoint:- <https://www.tutorialspoint.com/postgresql/index.htm>

ERD URL:

<https://dbdiagram.io/d/Swansea-Art-Emporium-Design-6578743356d8064ca0e21995>