



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

School of Applied Computing

ACCA7002

Report on Database Implementation:

Swansea Art Emporium (SWE)

By

Chisom Vanessa Nwankwo

2227183

TABLE OF CONTENTS

ABSTRACT.....	iii
CHAPTER ONE: THE DATASET.....	1
1.1. Creating the Dataset	1
1.1.1. Creating and Normalising the Tables.....	1
CHAPTER TWO: DATABASE FEATURES.....	2
2.1. TASK 1: Database Creation	2
2.1.1. Schema of the Swansea Art Emporium Database.....	2
2.2. Performance Tuning Features of the Database.....	3
2.2.1. Triggers with Auditing.....	3
2.2.2. Stored Procedures	3
2.2.3. Transactions	3
2.2.4. Functions.....	4
2.2.5. Indexing	4
CHAPTER THREE: WRITTEN ELEMENTS.....	5
3.1. Performance Tuning	5
3.1.1. Index on customerID in the Orders Table:.....	5
3.1.2. Index on artworkID column in the OrderDetails table:	5
3.1.3. Index on customerID in commission table:	5
3.2. Transaction	6
3.3. Database Evaluation.....	7
3.3.1. ERD Evaluation for Swansea Art Emporium	7
3.3.2. Database Deployment Considerations	8
REFERENCES	10

ABSTRACT

The "Swansea Art Emporium" report provides a detailed analysis and implementation strategy for a robust and efficient database system tailored for an art gallery. It covers the process of dataset creation, normalisation, and the establishment of a structured database schema, ensuring data integrity and consistency. Features like triggers, stored procedures, and indexing are discussed for performance tuning and security. The report evaluates the system's design through an Entity-Relationship Diagram, addressing deployment considerations and management strategies. Ultimately, it offers comprehensive insights into optimising a database for the dynamic needs of an art emporium, emphasising performance, integrity, and practical management.

CHAPTER ONE: THE DATASET

1.1. Creating the Dataset

The data contained in the Swansea Art Emporium Database is a combination of six data sources: [1], [2], [3], [4], [5], [6], which were cleaned and organised in Microsoft Excel and imported into pgAdmin4 as CSV files to ensure a robust and detailed database. Each column of the merged dataset was filtered to prepare the data, with columns containing unnecessary information and rows with important missing data removed. Empty rows and columns were filled in based on similar data from previous rows and columns. The column names and datatypes were organised to ensure consistency and prevent errors during importation. Specific columns in the dataset were also randomly generated with the RAND() and RANDBETWEEN() functions in Excel.

1.1.1. Creating and Normalising the Tables

The organised data was split into worksheets, converted into individual CSV files, and then tables for the database. Following the principles of normalisation to ensure that the tables were in 3NF, the worksheets were separated, with each worksheet containing its data, eliminating transitive dependencies, and generating ID columns for each worksheet in Excel to serve as primary key columns. Foreign key columns were also created with the aid of XLOOKUP() and VLOOKUP() functions after identifying relationships between the entities.

CHAPTER TWO: DATABASE FEATURES

2.1. TASK 1: Database Creation

2.1.1. Schema of the Swansea Art Emporium Database

Artist Table PK: artistID Non-key Attributes: role, name, birthYear, deathYear, nationality, gender, ULAN_URL, wikidata_URL

Artwork Table PK: artworkID FK: artistID (References Artist) Non-key Attributes: Is_Highlight, galleryNumber, departmentID, price, accessionYear, title, beginDate, endDate, medium, dimension, creditLine, rights_and_reproduction, imageResource, repository, tags

Customer Table PK: customerID Non-key Attributes: firstName, lastName, city, country, phoneNumber, email, registrationDate

Department Table PK: departmentID Non-key Attributes: name

Category Table PK: categoryID Non-key Attributes: name

ArtworkCategory Table (Junction Table for Many-to-Many relationship between Artwork and Category) PK: Composite (artworkID, categoryID) FK: artworkID (References Artwork), categoryID (References Category)

Orders Table PK: orderID FK: customerID (References Customer) Non-key Attributes: orderDate, orderPriority, deliveryStatus

OrderDetails Table (Junction Table for Many-to-Many relationship between Orders and Artwork) PK: Composite (orderID, artworkID) FK: orderID (References Orders), artworkID (References Artwork) Non-key Attributes: quantity, amount

Gallery Table PK: galleryNumber Non-key Attributes: name

Feedback Table PK: feedbackID FK: artworkID (References Artwork), customerID (References Customer) Non-key Attributes: feedbackDate, rating, comment

Commission Table PK: commissionID FK: customerID (References Customer) Non-key Attributes: title, requirements, budget, status

2.2. Performance Tuning Features of the Database

Essential features of the Swansea Art Emporium database for performance enhancement are functions, triggers with an auditing feature, stored procedures, transactions (rollback and commit), and indexes.

2.2.1. Triggers with Auditing

Triggers in database implementation allow for database auditing [7], which is one of the issues in database security. These database audits are needed to detect errors and prevent loss of data. It is vital in database operations because it keeps track of actions to detect and prevent illegal access to the database. The first step in the audit process is logging activities, which is keeping a record of activities within the database to determine who accessed the system, when, how and why it was accessed. Simply put, the use of triggers enables the logging and auditing of database activities.

2.2.2. Stored Procedures

This refers to a group of SQL queries stored in a database catalogue that allows the queries to be recalled. Typically, database operations like insert, update and delete can use stored procedures. It optimises the performance of database queries and reduces the security gap at the interface level of a system [8]. Hasan and Abu Bakar [9], in their research, highlight the role of stored procedures in ensuring the completeness and integrity of data transformation, stating their importance in managing metadata with diverse database systems. The study by [10] also discusses its importance in maintaining business logic and the integrity of data during the migration of databases to PostgreSQL from ORACLE.

2.2.3. Transactions

Transactions in a database refer to one or a sequence of multiple operations performed as a single logical unit. Two important statements in database transactions include the ROLLBACK AND COMMIT statements. The code in a database transaction must be fully executed and committed or not executed at all, then rolled back to prevent partial execution of code, which might lead to errors. The COMMIT statement executes to make a transaction permanent. In contrast, a ROLLBACK statement undoes changes made in the event that a transaction does not execute successfully and restores the database to its previous state before the transaction. The primary purpose of database transactions is to handle related operations in a controlled and safe manner, ensuring data integrity, consistency, and reliability. The study by [11] emphasises the role of

transactions in enhancing performance and data security and proposes solutions to minimise transaction latency. Liu [12] presents a novel SNOW-optimal algorithm called LORA for distributed read atomic transactions in cloud databases. The study discusses the importance of database transactions in balancing strong data consistency and availability.

2.2.4. Functions

Functions in database programming are predefined or user-defined sequences of SQL code that perform a specific task, usually a computation or data manipulation task within a database. They are essential tools for more efficient, readable, and secure code. According to [13], the use of functions within a database environment is of high importance because it can aid several database operations like data definition through the creation and modification of the database schema, data retrieval using code for querying the database, data security by creating functions to implement access control mechanisms like authentication and authorisation. Functions can also be written to aid in insert, delete and update database operations.

2.2.5. Indexing

Indexing in database programming is a data structure (typically a B-tree or hash table) used to speed up the retrieval of records from a database table. When an index is created, the database goes through all the records in the table and builds a data structure that holds the indexed column's values and pointers to the actual records holding those values. This structure is then used to locate rows quickly: instead of scanning the entire table, the database follows the pointers in the index. A study by [14] reviews different distributed query optimisers and highlights the importance of indexing for query optimisation. The study states the necessity of effective indexing techniques in distributed systems where data retrieval and management are complex. Also, Gayathiri et al. [15] explore two techniques for Big Data retrieval, Hash Indexing and MapReduce, with NoSQL databases. The efficiency of these methods in handling large datasets is compared, and the results show that indexing, particularly hash-based methods, is vital for efficient data retrieval and management in big data environments.

CHAPTER THREE: WRITTEN ELEMENTS

3.1. Performance Tuning

Indexes to improve the performance of important / frequently executed data queries.

3.1.1. Index on customerID in the Orders Table:

The primary purpose of creating an index on the customerID column in the Orders table is to speed up the retrieval of data from the Orders table based on the CustomerID. For example, it is used to determine all orders made by a particular customer, mainly because it is likely to be executed frequently. Without indexes, query filtering by customerID would require a full table scan, which can be time-consuming, especially for large tables. The index allows the database to find the relevant rows more quickly. Also, if there are queries that join the Orders table with another table (like the Customer table) based on customerID, this index will make those join operations more efficient.

3.1.2. Index on artworkID column in the OrderDetails table:

Another important query is one that fetches the details of all orders for a specific artwork. For example, to determine an artwork that is most requested. Also, there will be times when joins will be performed on this table with other tables, like the 'Artwork' table. Without the index, this query might be slow, particularly as the size of the OrderDetails table grows. With the index, the database can quickly locate all orders related to a particular artwork, significantly improving the query's execution time.

3.1.3. Index on customerID in commission table:

When queries that filter or join on the customerID in the Commission table are executed, the index allows the database to locate the rows associated with a specific customer much faster than it would by scanning the entire table. Also, the table will likely be joined with the Customer table to retrieve a customer's details like phone number. Indexing on this column will help speed up the query process. It can also improve the performance of operations that involve cascading updates or deletes.

3.2. Transaction

Stored procedure transactions for insert, update, and delete operations like the ones created on Artwork, Artist, Customer and Commission tables, are crucial in the Swansea Art Emporium database to ensure the ACID (Atomicity, Consistency, Isolation, Durability) property of databases:

a. Ensuring Atomicity

A fundamental principle of database transactions is atomicity. This allows a series of operations within a transaction to be treated as a single unit, which succeeds or fails together. For example, the stored procedure for the Artist table ensures that an update to the table, which may be part of a larger transaction, will only be committed if all other operations are successfully committed.

b. Maintaining consistency of data

Errors that occur during any operation within the transactions can be rolled back to its initial state, thereby preventing partial updates that could lead to data inconsistencies.

c. Ensuring Isolation

Transactions provide isolation of a database operation from other concurrent database operations. Changes made in a transaction are invisible to other concurrent transactions until they are committed, preventing "dirty reads" and ensuring that each transaction sees a consistent data view.

d. Durability

Transactions ensure durability. This means that once a transaction is successfully committed, its changes are permanent, even in the event of a system failure. This durability is crucial for maintaining a reliable record of data changes.

In summary, the stored procedure transactions for the Artwork, Artist, Customer and Commission tables ensure that operations like adding new artwork, updating artists' details, updating customers' details, or deleting a commission are handled reliably, consistently, and without leaving the database in an inconsistent state. They help manage complex relationships and dependencies between these tables efficiently and securely. Using stored procedure transactions for insert, update, and delete operations in a database like that of Swansea Art Emporium, not only enhances

data integrity and performance but also plays a critical role in maintaining the overall health and reliability of the database system.

3.3. Database Evaluation

3.3.1. ERD Evaluation for Swansea Art Emporium

a. Entity Relationships

The ERD clearly defines the relationships between entities such as Artwork, Artist, Customer, Orders, etc. This includes correctly mapping one-to-many and many-to-many relationships using foreign keys and associative entities like ArtworkCategory for the many-to-many relationship between Artwork and Category and the OrderDetails for the many-to-many relationships between Orders and Artwork. Primary keys are correctly defined in all tables, ensuring unique record identification. Foreign keys are specified where relationships exist, which helps maintain referential integrity across the database.

b. Attribute Types

Data types are clearly indicated (int, string, float, Boolean, date), which helps understand the kind of data stored in each table.

c. Schema Clarity

Non-key attributes are well-defined in the ERD, providing a comprehensive understanding of the data stored in each table. The ERD includes relationship labels, e.g., "creates," "managed_by," and "categorized_as", which help in understanding the nature of the relationships. The cardinality is also depicted (one-to-one, one-to-many, many-to-many) using crow's feet notation.

d. Naming Conventions

The naming conventions are consistent with singular nouns for table names (except for Orders. 'Order' is a keyword in PostgreSQL), which is good practice.

e. Normalisation

The schema complies with the third normal form (3NF) normalisation technique, reducing redundancy and ensuring data integrity by ensuring that each piece of data is stored only once.

3.3.2. Database Deployment Considerations

Transaction isolation levels are crucial when deploying the database in a multi-user environment to prevent concurrency issues such as lost updates, uncommitted dependencies (dirty reads), and inconsistent analysis (non-repeatable reads or phantom reads).

a. Transaction Isolation Levels

- i. Read Uncommitted: This level should be avoided in a multi-user environment as it allows dirty reads.
- ii. Read Committed: This is the default level in PostgreSQL and prevents dirty reads but doesn't prevent non-repeatable or phantom reads.
- iii. Repeatable Read: Prevents dirty and non-repeatable but not phantom reads.
- iv. Serializable: This level can provide the strictest isolation by preventing dirty, non-repeatable, and phantom reads. It's suitable for situations where accuracy is critical, but it can lead to decreased performance due to increased locking.

b. Avoiding Concurrency Problems

- i. Lost Updates: This occurs when two concurrent transactions update the same data, and one overwrites the other. Transaction locks should be used (`SELECT FOR UPDATE`) to lock records during a transaction to prevent simultaneous transactions from overwriting each other's updates, or optimistic concurrency control should be implemented by checking a version number or timestamp before committing updates.
- ii. Uncommitted Dependencies: This happens when a transaction reads data from a concurrent uncommitted transaction. Isolation level of at least Read Committed should be used to avoid reading uncommitted data from other transactions.
- iii. Inconsistent Analysis: To prevent this, Repeatable Read or Serializable isolation level should be used to ensure a transaction can repeat the same read and get the same results, even if other transactions are modifying the data, depending on the criticality of the operations and the acceptable performance trade-offs.

Indexes like those previously created are essential in improving query performance in a multi-user database. Queries should be optimised to use indexes effectively, especially in JOIN operations and WHERE clauses. Also, the indexes should be maintained with periodic reindexing, especially for tables that undergo frequent updates to prevent index bloat.

The schema for Swansea Art Emporium is structurally sound and follows relational database design principles. However, for transaction isolation levels when deploying in a multi-user environment, pay special attention to prevent concurrency issues. Regular monitoring and maintenance will help in avoiding performance degradation over time. Additionally, strategic indexing will be essential for maintaining optimal query performance.

REFERENCES

- [1] metmuseum, “openaccess/ at master · metmuseum/openaccess,” *GitHub*, 2016. <https://github.com/metmuseum/openaccess?search=1> (accessed Dec. 20, 2023).
- [2] A. Derungs, “1001 Memorable Ideas For Art Gallery Names Plus Naming Tips for 2023,” *Niche Pursuits*, Jul. 10, 2023. <https://www.nichepursuits.com/art-gallery-names/> (accessed Dec. 20, 2023).
- [3] “data.world,” *Data.world*, 2023. https://data.world/kgarrett/unwanted-calls/workspace/file?filename=Consumer_Complaints_Data_-_Unwanted_Calls.csv (accessed Dec. 20, 2023).
- [4] “Customer Testimonials | 1st Art Gallery,” *1st-art-gallery.com*, 2019. <https://www.1st-art-gallery.com/testimonials-artists.html> (accessed Dec. 20, 2023).
- [5] datablist, “GitHub - datablist/sample-csv-files,” *GitHub*, Nov. 18, 2022. <https://github.com/datablist/sample-csv-files> (accessed Dec. 20, 2023).
- [6] Mohammed, “Customer Order Data,” *Data.world*, May 16, 2017. <https://data.world/yasin/customer-order-data> (accessed Dec. 20, 2023).
- [7] P. Togatorop, H. Sitorus, R. Sirait, and T. Manurung, “Database Audit System Design and Implementation,” *Jurnal Mantik*, vol. 5, no. 4, pp. 2535–2541, 2022, Accessed: Dec. 20, 2023. [Online]. Available: www.iocscience.org/ejournal/index.php/mantik/index
- [8] N. Hartono and E. Erfina, “Comparison of Stored Procedures on Relational Database Management System,” *Tech-E*, vol. 4, no. 2, pp. 8–15, Mar. 2021, doi: <https://doi.org/10.31253/te.v4i2.529>.
- [9] F. F. Hasan and M. S. A. Bakar, “An approach for metadata extraction and transformation for various data sources using R programming language,” *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 26, no. 3, p. 1520, Jun. 2022, doi: <https://doi.org/10.11591/ijeecs.v26.i3.pp1520-1529>.
- [10] E. A. Kypako and V. Orlov, “Database Migration from ORACLE to PostgreSQL,” *Programming and Computer Software*, vol. 49, no. 5, pp. 455–463, Oct. 2023, doi: <https://doi.org/10.1134/s0361768823050055>.

- [11] Ana-Maria Dincă, Sabina-Daniela Axinte, and Ioan BACIVAROV, “Performance Enhancements for Database Transactions,” *International Journal of Information Security and Cybercrime*, vol. 11, no. 2, pp. 29–34, Dec. 2022, doi: <https://doi.org/10.19107/ijisc.2022.02.02>.
- [12] S. Liu, “All in One: Design, Verification, and Implementation of SNOW-optimal Read Atomic Transactions,” *ACM Transactions on Software Engineering and Methodology*, vol. 31, no. 3, pp. 1–44, Mar. 2022, doi: <https://doi.org/10.1145/3494517>.
- [13] D. Chopra, “What are Functions of DBMS?,” *Analytics Vidhya*, Jun. 06, 2023. <https://www.analyticsvidhya.com/blog/2023/06/what-are-functions-of-dbms/> (accessed Dec. 21, 2023).
- [14] M. Sharma, G. Singh, and R. Singh, “A review of different cost-based distributed query optimizers,” *Progress in Artificial Intelligence*, vol. 8, no. 1, pp. 45–62, Jun. 2018, doi: <https://doi.org/10.1007/s13748-018-0154-8>.
- [15] N.R. Gayathiri, D. David Jaspher, and A. M. Natarajan, “Big Data retrieval techniques based on Hash Indexing and MapReduce approach with NoSQL Database,” in *International Conference on Advances in Computing and Communication Engineering (ICACCE)*, IEEE, Apr. 2019. doi: <https://doi.org/10.1109/icacce46606.2019.9079964>.