

1703645_MatthewLewington_A CCA7002_DatabaseSystems_L7 _Report

by Matthew Dene Lewington

FILE	4069_MATTHEW_DENE_LEWINGTON_1703645_MATTHEWLEWINGTON_A CA7002_DATABASESYSTEMS_L7_REPORT_315194_535654274.DOCX (424.43K)		
TIME SUBMITTED	13-JAN-2021 10:22AM (UTC+0000)	WORD COUNT	3547
SUBMISSION ID	141886662	CHARACTER COUNT	19122



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

Abertawe Arts Centre Case Study Scenario

**Database Systems
(ACCA7002)**

Matthew Lewington (1703645)
MSc Data Science and Analytics

Wales Institute for Science and Art (WISA)
University of Wales Trinity Saint David

January 15, 2021

ABSTRACT

SQL vs NoSQL is a debated topic, but both have their own place (depending on requirements) within a database design. The Abertawe Arts case study scenario has been successfully implemented and all the requirements fulfilled. This document outlines all the steps taken to complete the case study scenario.

Table of Contents

Table of Figures	iv
Table of Tables	v
1 Introduction	1
2 SQL vs. NoSQL	2
2.1 SQL.....	2
2.1.1 Structure	2
2.1.2 Constraints	3
2.1.3 Normalisation.....	3
2.1.4 ACID Properties.....	4
2.1.5 Scalability	4
2.2 NoSQL	5
2.2.1 Structure	5
2.2.2 BASE Properties.....	6
2.2.3 Scalability	6
2.3 Comparison.....	7
3 Performance Tuning	8
3.1 Index 1 – IX_Customer_PaymentInfo	8
3.2 Index 2 – IX_Customer_Address	8
3.3 Index 3 – IX_ProductionCompany_Contact	8
4 Transactions	9
4.1 BookVenue Transaction.....	9
4.2 BookCustomerViewing Transaction	9
5 Distributed Database Solution.....	11
5.1 Customer	11
5.2 ViewingBooking	11
5.3 TicketCost	12
5.4 Show	12

5.5	Venue.....	12
5.6	ProductionCompany.....	12
5.7	VenueBooking.....	13
5.8	VenuePrice	13
5.9	Stall.....	13
5.10	Seat	13
6	Conclusion	14
7	References.....	15
	Appendix A Abertawe Arts Entity Relationship Diagram	17

Table of Figures

Figure 2-1 Entity relationship diagram (ERD) example [2, p. 64]	3
Figure 2-2 Process of normalisation.....	4
Figure 4-1 BookVenue transaction.....	9
Figure 4-2 BookCustomerViewing transaction	10
Figure A-1 Abertawe Arts ERD	17

Table of Tables

Table 1 Common NoSQL Database Technologies	6
--	---

1 Introduction

This report has been produced to accompany the SQL files submitted as part of task one for the Abertawe Arts Centre case study scenario. Provided herein will be a discussion on SQL vs. NoSQL to provide clarification for the Abertawe Arts Centre with advantages and disadvantages of each, accompanied by a recommendation for the given scenario. This report will also contain:

- An outline of the performance tuning executed to enhance the database queries,
- A description of the transaction written as a solution to the problems faced with the current booking system
- A proposal for the distributed database design showing a suitable fragmentation schema and fragmentation reconstruction

2 SQL vs. NoSQL

This section will discuss SQL vs. NoSQL covering advantages and disadvantages of each, and which may be better suited to the Abertawe Arts Centre scenario.

2.1 SQL

Structured Query Language (SQL) is the language which is considered the standard amongst relational databases such as Oracle, MSSQL, MySQL and PostgreSQL [1]. SQL satisfies the requirements of allowing a user to complete tasks such as creating a database and relational structures, and perform basic data management tasks such as insertion, modification, deletion and querying of data (commonly known as CRUD operations for Create, Read, Update and Delete) [2, pp. 191–193], [3, p. 15]. These tasks are fulfilled by the ISO 9075 standard [2, p. 193], [4] which describe two key components:

- **Data Definition Language (DDL)** – Defining database structure and controlling access to the data,
- **Data Manipulation Language (DML)** – Retrieving and updating data.

2.1.1 Structure

The structure, or schema, of a relational database requires that the database is recognized by the user as tables [2, p. 152], as shown in Figure 2-1 with a set of tables which are Branch, Staff, PropertyForRent, Client, Lease and PrivateOwner. Each of these tables contain fields, or attributes, which are likely the unique identifiers for each table and potentially a primary key, or part of a composite key.

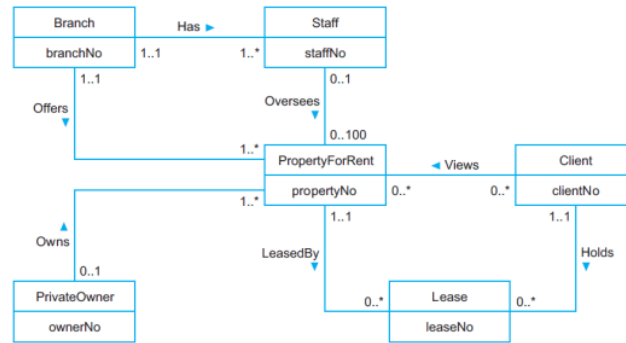


Figure 2-1 Entity relationship diagram (ERD) example [2, p. 64]

Along with the table design of the database logical structure, the whole idea of an RDBMS is a concept called relations, which are used to connect information between tables in the database [2, pp. 152–153]. This is shown again in Figure 2-1 as the links between tables.

2.1.2 Constraints

Following on from relations, constraints enforce integrity and consistency of the database and are also used to identify relationships between tuples [2, pp. 161–163], [5, p. 8]. There are two key rules for the relational model which are:

- **Entity Integrity** – No attribute of a primary key can be a null value,
- **Referential Integrity** – Foreign keys must match a key from the attribute of the originating table.

2.1.3 Normalisation

A key feature of relational databases is normalisation. Normalisation is the elimination of repeating data by producing suitable relations which support the data requirements of an organisation [2, pp. 451–452], which along with the database structure, can directly affect query performance optimisation if implemented correctly [6].

There are several stages of normalisation (Figure 2-2) from unnormalized form, however the minimum aimed for is at least three:

- **First Normal Form (1NF)** – Eliminates repeating groups,
- **Second Normal Form (2NF)** – Eliminates partial dependencies,
- **Third Normal Form (3NF)** – Eliminates transitive dependencies.

The prior phase(s) of normalisation must be satisfied before moving on as shown in Figure 2-2.



Figure 2-2 Process of normalisation

2.1.4 ACID Properties

ACID (Atomicity, Consistency, Isolation, Durability) properties are an important factor of relational database transactions, guaranteeing their reliability [2, p. 671], [7], [8].

- **Atomicity** – “All or nothing”. Either the entire transaction happens, or should one part fail, the transaction will either not go ahead or will be reversed.
- **Consistency** – The database integrity is preserved by the transaction ensuring the state of the database is consistent and not invalid.
- **Isolation** – Transactions execute independently of each other to ensure any data changed during one transaction does not affect another transaction running in parallel.
- **Durability** – Upon a transaction completing, the effects are instantly and permanently recorded within the database.

2.1.5 Scalability

Scalability is a general term, however in databases it is the ability to handle an increased and varied workload in terms of transactions and stored data. There are two types of scalability [8], [9]:

- **Horizontal** – Increasing the number of nodes (machines) in the resource cluster.
- **Vertical** – Increasing the power of the node by installing a better CPU or additional RAM.

Relational databases utilise vertical scalability as it can prove challenging to scale horizontally [10, p. 25]. As the volume of the data increases, the capacity

of the hard drive and processing power can be increased. This can prove to be an expensive investment, particularly in the event of a hardware failure [8].

2.2 NoSQL

Not only SQL, also known as Not Just SQL or No SQL, (NoSQL) databases are a newer generation of database management system that is not modelled on the conventional RDBMS in that they have no pre-defined schema and records can have various fields. Attributes can be dynamically added without having to have a database developer change the design of the database [10, p. 27].

NoSQL databases offer the ability to solve a variety of potential data management problems as the diversity of the many subtypes of NoSQL databases can be suited better to specific datasets than others [11]. These databases are designed to handle large volumes of data, varieties of data types and structures, and the swiftness of data operations that are of new business requirements [12, p. 11].

2.2.1 Structure

Though NoSQL databases have no concrete structure, there are four key types of NoSQL database [8], [10, p. 32], [13, p. 4] and several common technologies build on them (Table 1):

- **Key-value database** – Similar to a hash table, keys are stored in an index. This then points to the data.
- **Column-oriented database** – Shares some resemblances with both RDBMS as data is organised into columns, and key-value databases as each row contains a primary, or row key.
- **Document-based database** – The most used model. Can contain multiple key-value pairs. Can handle all types of data including structured and unstructured data.
- **Graph database** – Data is stored and represented as graphs, and like graph theory, graphs are sets of nodes (entities) and edges (relations between nodes).

Common NoSQL Database Technologies	
Key-value Database	DynamoDB [14], Redis [15], Couchbase [16]
Column-oriented Database	Cassandra [17]
Document-based Database	MongoDB [18], CouchDB [19]
Graph Database	Neo4j [20]

Table 1 Common NoSQL Database Technologies

2.2.2 BASE Properties

NoSQL databases rely on BASE properties which are more flexible than the RDBMS ACID properties [8], [10, pp. 43–44], [11]:

- **Basically Available** – In the event of a partial failure, the database ensures to attempt to respond to queries.
- **Soft State** – Though NoSQL does not require data to be refreshed, the system may update without user intervention to provide eventual consistency.
- **Eventual Consistency** – The system may temporarily be inconsistent in the case of data being kept across multiple servers, however eventually, replication will ensure all copies are consistent.

2.2.3 Scalability

NoSQL databases can successfully integrate both horizontal and vertical scaling. In the situation where horizontal scaling is utilised, instead of solely increasing the power of a node, additional nodes are added for data storage and processing power. This can be a cheaper and more flexible solution than vertical scaling as servers can be added or removed as required [10, p. 23].

This scaling option can also provide relief during disaster recovery as instead of the single node going down and the data being inaccessible, when one node fails or is taken down for maintenance then the other nodes within the cluster are able to have that nodes workload diverted to it and they are able to carry out the necessary operations [10, p. 27].

2.3 Comparison

Both types of database have their own rightful place within an organisation. The benefits of NoSQL over a traditional RDBMS range from being able to support a wide variety of data models and being able to take advantage of scaling easily to being more flexible and potentially faster or more efficient. On the other hand, the RDBMS is simple to design, maintain and use and a strong characteristic of an RDBMS are the ACID properties [21], [22].

The NoSQL databases provide an upper hand in the world of Big Data and cloud computing, however with a requirement for the Abertawe Arts Centre, the conventional relational database should prove to be a better choice as they are not working with the amounts of data usually seen with NoSQL databases, and the structure and attributes of the schema are all required – nothing should be missing, and nothing more should be required - the rigidity of this schema will prove useful as information will not be missed or null.

The integrity enforcement is also key as the data must be linked correctly to ensure bookings are completed correctly, allocating viewings in the correct venue, and confirming payment details. Again, the ACID properties play a large roll in ensuring that transactions are completed correctly and efficiently with any potential problems causing a rollback.

Though operational transactions have been proven to be quicker on a NoSQL database in specific scenarios [23]–[25], there will not be thousands of transactions taking place at the same time on the Abertawe Arts Centre system therefore it is not deemed acceptable to sacrifice the improved security and integrity for performance.

3 Performance Tuning

A method of improving database performance is by adding secondary indexes for queries that are run frequently. Three secondary indexes were added to the database to assist.

3.1 Index 1 – IX_Customer_PaymentInfo

This index was created to increase the performance of any payment transactions. PaymentCard and PaymentCardCVC columns would only be updated if a card is replaced, or if the customer is wanting to use a different payment card which (is assumed) would not be a regular occurrence.

3.2 Index 2 – IX_Customer_Address

This index was produced to cater for sending tickets or booking confirmation out to customers. It is not stated within the brief how the customer would receive these, if at all. The assumption has been made that the ticket office will create a ticket and either send a paper copy to the customer in the post, hence the address being selected on the index, or via e-mail which is why the e-mail is also included.

It may be better to have an index for each one; CustomerID and address details, then CustomerID and e-mail. This would further increase the performance by only including relevant details for the preferred contact method.



3.3 Index 3 – IX_ProductionCompany_Contact

This index was added to assist with querying the production company contact details to allow Abertawe Arts Centre to swiftly communicate with the production company for reasons such as any unforeseen circumstances potentially affecting the booking, for payment information, etc...

It is assumed that there would potentially be an account manager that would deal with these queries.

4 Transactions

The two problems raised by Abertawe Arts were addressed by using the transactions outlined here.

4.1 BookVenue Transaction

It is assumed that either the dates are null values, or the transaction is failing at some point. As the BeginDateTime and EndDateTime attributes specifically state NOT NULL as a requirement it will not be a null value causing a problem. A "Try...Catch" statement was produced that attempts to insert the given values for the VenueBooking and commit the transaction. If the transaction fails it will rollback and not commit (Figure 4-1). This correlates with the ACID Properties discussed earlier in this report.

```
BEGIN TRANSACTION BookVenue
DECLARE @DateBegin SMALLDATETIME = '2021-05-01 00:01:00';
DECLARE @DateEnd SMALLDATETIME = '2021-05-02 12:30:00';

BEGIN TRY
    INSERT INTO VenueBooking (VenueBookingID, ShowID, VenueID, BeginDateTime, EndDateTime)
    VALUES ('00009', 'S1003', 'AA01', @DateBegin, @DateEnd);
    COMMIT TRANSACTION BookVenue;
END TRY

BEGIN CATCH
    ROLLBACK TRANSACTION BookVenue
    RAISERROR ('Rolling back - transaction failed', 16, 1);
END CATCH;
```

Figure 4-1 BookVenue transaction

4.2 BookCustomerViewing Transaction

Assuming this problem is customers can book tickets to events that have already passed and are no longer registered as events (the brief does not state that there are non-events such as rehearsals, so it is not expected to be this), a check was put in place to ensure that the booking date/time was greater than the transaction date/time (Figure 4-2). The VenueBooking table needed to be joined to the query, as when the production company book a venue, they provide the available dates, and they are added to the VenueBooking table.



```

BEGIN TRANSACTION BookCustomerViewing
DECLARE @ShowID VARCHAR(5) = 'S1002';

BEGIN TRY
    IF (SELECT COUNT(*) FROM Show s RIGHT JOIN VenueBooking vb
        ON s.ShowID = vb.ShowID
        WHERE vb.BeginDateTime > GETDATE() AND s.ShowID = @ShowID) > 0
        INSERT INTO ViewingBooking (BookingID, CustomerID, ShowID, VenueID, SeatsRequired)
            VALUES ('000092', 'CU0002', @ShowID, 'AA01', 1);
        COMMIT TRANSACTION BookCustomerViewing;
    END TRY

    BEGIN CATCH
        ROLLBACK TRANSACTION BookCustomerViewing
        RAISERROR ('Rolling back. Event has passed', 16, 1);
    END CATCH

```

Figure 4-2 BookCustomerViewing transaction

5 Distributed Database Solution

Each table will be required to be a part of the distributed database solution; this section will outline the specific fragmentation strategy (horizontal, vertical, hybrid or derived) and the reconstruction of the global relations from the fragments. Due to the small size of the organisation and the fact that it is a single site, it is assumed that records may be split between the relevant functions (scheduling, marketing, and sales) and possibly venues (main auditorium, Dylan 1, and Burton 2).

5.1 Customer

For the Customer table, it is thought that the best approach would be to apply vertical fragmentation for the marketing team to store the customers address details to allow for them to continue to send out relevant information, and then for the payment information to be stored with the sales teams.

C_1 : Π CustomerID, FirstName, LastName, DateOfBirth, Address, City, PostCode, PhoneNumber, Email (**Customer**)

C_2 : Π CustomerID, PaymentCard, PaymentCardCVC (**Customer**)

Reconstruction: $C_1 \bowtie C_2$

5.2 ViewingBooking

It is thought that the ViewingBooking table would benefit to being split based on the venue and stored in the venue, but this could also then be stored at the sales / ticket office.

ViB_1 : $ViB \triangleright_{VenueID} V_1$ (V_1 is a horizontal fragment from Venue, VenueID = 'AA01')

ViB_2 : $ViB \triangleright_{VenueID} V_2$ (V_2 is a horizontal fragment from Venue, VenueID = 'DY01')

ViB_3 : $ViB \triangleright_{VenueID} V_3$ (V_3 is a horizontal fragment from Venue, VenueID = 'BT01')

Reconstruction: $(ViB_1 \cup ViB_2 \cup ViB_3)$

5.3 TicketCost

Due to the small size of this table, it should be possible to replicate this across all sites as a backup, however it will primarily be required in the sales team.

5.4 Show

The decision was made to store these records locally to each of the venue's, whichever may be hosting the shows. This may assist with ticket checking or assisting staff but may need to be revised pending Abertawe Arts processes / layout.

$S_1: S \triangleright_{\text{VenueID}} \text{VeB}_1$ (VeB_1 is a horizontal fragment from VenueBooking, VenueID = 'AA01')

$S_2: S \triangleright_{\text{VenueID}} \text{VeB}_2$ (VeB_2 is a horizontal fragment from VenueBooking, VenueID = 'DY01')

$S_3: S \triangleright_{\text{VenueID}} \text{VeB}_3$ (VeB_3 is a horizontal fragment from VenueBooking, VenueID = 'BT01')

Reconstruction: $(S_1 \cup S_2 \cup S_3)$

5.5 Venue

Due to the small number of records, it should be reasonable to duplicate this table across each of the sites. Horizontal fragmentation should also be applied.

$V_1: \sigma_{\text{VenueID}} = \text{'AA01'}$

$V_2: \sigma_{\text{VenueID}} = \text{'DY01'}$

$V_3: \sigma_{\text{VenueID}} = \text{'BT01'}$

Reconstruction: $(V_1 \cup V_2 \cup V_3)$

5.6 ProductionCompany

Like the Customer table, ProductionCompany could follow the same vertical fragmentation and be split across schedule and sales, however no payment information has been included in the database as it is not known how a company would pay for their rental. As no payment information has been

included, it may be better offering no fragmentation in its current design state and replicating across marketing and scheduling.

5.7 VenueBooking

It may be beneficial to fragment the VenueBooking table horizontally and store this per venue. The scheduling team will need the most access to this table.

VeB₁: $\sigma_{\text{VenueID}} = \text{'AA01'}$

VeB₂: $\sigma_{\text{VenueID}} = \text{'DY01'}$

VeB₃: $\sigma_{\text{VenueID}} = \text{'BT01'}$

Reconstruction: $(VB_1 \cup VB_2 \cup VB_3)$

5.8 VenuePrice

Due to the small size of this, it may be a better idea to not fragment this table and replicate it across all sites – the primary site(s) would likely be sales or scheduling.

5.9 Stall

Because of the lesser volume of the Stall size, it again may be a better idea not to fragment this table and replicate it across all sites.

5.10 Seat

Finally, again because of the size of the Seat table, it could be an idea to not fragment this table and replicate it across all sites.



6 Conclusion

SQL vs. NoSQL is a highly debated topic, but each of the technologies are relevant depending on a business' circumstances. Guidance has been provided to the Abertawe Arts business on which would be more beneficial to their circumstances.

All the work completed for the Abertawe Arts has been documented and completed successfully – the performance tuning has been outlined, and the transactions to ensure correctness have been provided along with a proposal for the database distribution.

7 References

- [1] W. Truskowski, R. Klewek, and M. Skublewska-Paszkowska, "Comparison of MySQL, MSSQL, PostgreSQL, Oracle databases performance, including virtualization," *J. Comput. Sci. Inst.*, no. 16, pp. 279–284, 2020.
- [2] T. M. Connolly, *Database systems : a practical approach to design, implementation and management*, Sixth edit. Harlow : Pearson Education Limited, 2015.
- [3] M. J. Hernandez, *Database design for mere mortals : a hands-on guide to relational database design*, Third edit. Upper Saddle River, NJ: Upper Saddle River, NJ : Addison-Wesley, 2013.
- [4] "ISO - ISO/IEC 9075-1:2016 - Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework)." <https://www.iso.org/standard/63555.html> (accessed Dec. 08, 2020).
- [5] G. Harrison, *Next Generation Databases*. 2015.
- [6] F. Mithani, S. Machchhar, and F. Jasdanwala, "A Novel Approach For SQL Query Optimization," in *2016 IEEE International Conference on Computational Intelligence and Computing Research (ICIC)*, Dec. 2016, pp. 1–4.
- [7] S. Palanisamy and P. SuvithaVani, "A survey on RDBMS and NoSQL Databases MySQL vs MongoDB," in *2020 International Conference on Computer Communication and Informatics (ICCCI)*, Jan. 2020, pp. 1–7.
- [8] K. Sahatqija, J. Ajdari, X. Zenuni, B. Raufi, and F. Ismaili, "Comparison between relational and NOSQL databases," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, May 2018, pp. 0216–0221.
- [9] D. Seybold, S. Volpert, S. Wesner, A. Bauer, N. Herbst, and J. Domaschka, "Kaa: Evaluating Elasticity of Cloud-Hosted DBMS," in *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Dec. 2019, pp. 54–61.
- [10] D. Sullivan, *NoSQL for Mere Mortals*, 1st ed. Addison-Wesley, 2015.
- [11] M. Plechawska-Wójcik and D. Rykowski, "Comparison of Relational, Document and Graph Databases in the Context of the Web Application Development," 2016, pp. 3–13.

- [12] C. Coronel and S. Morris, *Database Systems: Design, Implementation, & Management*, 12th ed. Course Technology, 2016.
- [13] D. Vohra, *NoSQL Web Development with Apache Cassandra*, 1st ed. Cengage Learning, 2015.
- [14] "DynamoDB." <https://aws.amazon.com/dynamodb/> (accessed Dec. 14, 2020).
- [15] "Redis." <https://redis.io/> (accessed Dec. 14, 2020).
- [16] "Couchbase." <https://www.couchbase.com/> (accessed Dec. 14, 2020).
- [17] "Cassandra." <https://cassandra.apache.org/> (accessed Dec. 14, 2020).
- [18] "MongoDB." <https://www.mongodb.com/> (accessed Dec. 14, 2020).
- [19] "CouchDB." <https://couchdb.apache.org/> (accessed Dec. 14, 2020).
- [20] "Neo4j." <https://neo4j.com/> (accessed Dec. 14, 2020).
- [21] T. N. Khasawneh, M. H. AL-Sahlee, and A. A. Safia, "SQL, NewSQL, and NOSQL Databases: A Comparative Survey," in *2020 11th International Conference on Information and Communication Systems (ICICS)*, Apr. 2020, pp. 013–021.
- [22] W. Ali, M. U. Shafique, M. A. Majeed, and A. Raza, "Comparison between SQL and NoSQL Databases and Their Relationship with Big Data Analytics," *Asian J. Res. Comput. Sci.*, pp. 1–10, Oct. 2019.
- [23] B. Sirish Shetty and K. Akshay, "Performance Analysis of Queries in RDBMS vs NoSQL," in *2019 2nd International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICT)*, Jul. 2019, pp. 1283–1286.
- [24] S. Rautmare and D. M. Bhalerao, "MySQL and NoSQL database comparison for IoT application," in *2016 IEEE International Conference on Advances in Computer Applications (ICACA)*, Oct. 2016, pp. 235–238.
- [25] C.-H. Lee and Z.-W. Shih, "A Comparison of NoSQL and SQL Databases Over the Hadoop and Spark Cloud Platforms Using Machine Learning Algorithms," in *2018 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW)*, May 2018, pp. 1–2.

Appendix A Abertawe Arts Entity Relationship Diagram

This appendix contains the ERD for the Abertawe Arts case study. Though the 1:1 / 1:many / etc... have not been included

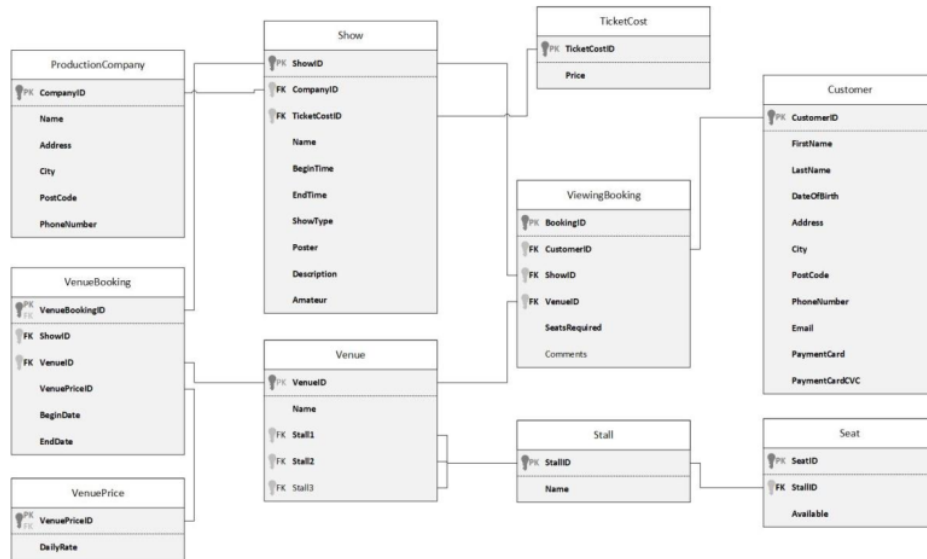


Figure A-1 Abertawe Arts ERD

FINAL GRADE

81 / 100

GENERAL COMMENTS

Instructor

An excellent report well presented. You have analysed and discussed sql nosql and reflected this back to the case study. Distributed database considering marketing and other departments demonstrated a sound understanding of this process.

You have implemented a grid type system for you seats i.e. all your seats are register with an ID and availability which is excellent but in you ERD schema you have not used this setup to its best advantage in the bookings transaction. Looking at you ERD for example if you have used an additional field customer id (null) in the seating plan then this could be linked to the actual seat /s they book.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12



Comment 1

Good clear analysis and reflection on case study



Comment 2

good



Comment 3

good interpretation and analysis



Comment 4

Excellent analysis clearly presented