

# 2006611-ZhenzhongYin

*by* Zhenzhong Yin

---

|                |                                                                         |                 |       |
|----------------|-------------------------------------------------------------------------|-----------------|-------|
| FILE           | 25261_ZHENZHONG_YIN_2006611-ZHENZHONGYIN_315194_232366856.DOCX (52.01K) |                 |       |
| TIME SUBMITTED | 15-JAN-2021 08:07PM (UTC+0000)                                          | WORD COUNT      | 2378  |
| SUBMISSION ID  | 142315756                                                               | CHARACTER COUNT | 12746 |

## Abstract

This paper designs and optimizes the database of Abertawe Arts Centre's venue reservation and performance ticket demand. In addition, the difference between SQL and NoSQL is discussed, and a distributed database scheme is designed.

## 1 introduction

Today's world is an internet world full of data. With the development of information technology and the continuous growth of data, the storage and management of data are becoming more and more important. Therefore, for a company, the use of database can achieve data sharing and centralized control, reduce data redundancy, increase data security and reliability. Based on the analysis of Abertawe Arts Centre's demands, this paper analyzes the two main tasks of booking three newly built venues and selling tickets for performances. Then E-R diagram is used to design the logical structure, SQL is used to establish the database and insert the data. The use of indexes and transactions addresses the issues raised by the company and makes the database more efficient. In addition, this article discusses the SQL versus NoSQL debate. Finally, a distributed database solution is designed for this task.

## 2 Design of database

In order to design a reasonable database, we should first analyze the needs of the Abertawe Art Center. Simply speaking, the Abertawe Art Center has three sites where the professional and amateur performing arts company could book for the performance. Each venue has seats at different prices for each performance, and customers could buy seats for the performance. We can find that there are four entities that exist in this case, which are 'companies', 'venues', 'customers' and 'seats'. And there are two main relationships, which are holding performances and buying tickets. Then I adopted the bottom-up conceptual structure design method, and got the E-R diagrams of the two main relationships as shown in Figure 1(a) and Figure 1(b).

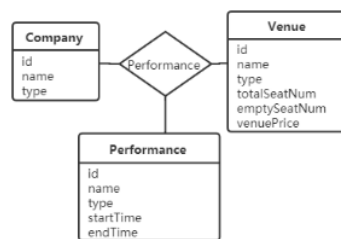


Figure 1(a) Performance relationship

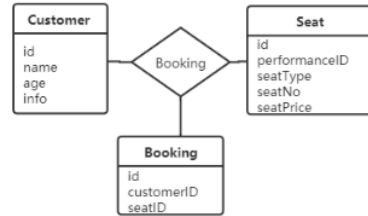


Figure 1(b) Booking relationship

After converting the E-R diagram into the relationship model and making some optimization, the data model has satisfied the 3NF.

Company ( id, name, type )

Venue ( id, name, type, totalSeatNum, emptySeatNum, venuePrice )

Performance ( id, name, type, companyID, venueID, startTime, endTime )

Seat ( id, performanceID, seatType, seatNo, seatPrice )

Customer ( id, name, age, info )

Booking ( id, customerID, seatID )

In relational database, constraint is a kind of restriction on the data in a table, which can help the database administrator manage the database better and ensure the correctness and validity of the data in the database. In these fields, the id in each table represents the primary key. In the Performance table, companyID and venueID are foreign keys, which respectively represent the company and venue where the performance will be held. In the Seat table, performanceID is a foreign key, and each performance is associated with a batch of the seat record. The reason why the Seat table is associated with the Performance table is that, considering the needs of the organizer of each performance, all the seats in the venue may not be used. Furthermore, sometimes some of the seats may be damaged. In the Booking table, customerID and seatID are foreign keys, respectively representing the customer and the seat he has reserved. Since a customer may buy more than one performance, this design is reasonable. In addition, all the necessary fields have NOT NULL constraints, and some have default values.

With the operation of the Abertawe Art Center, more and more companies will reserve venues and more and more audiences will come to watch performances. It is necessary to build an index to improve the efficiency of data query. However, the establishment of an index will not only bring more storage costs, but also will increase the time of insert, update, and delete operations. So which table needs to build index and which column should be indexed is the key problem we have to consider. In the company table and the Customer table, information about companies and customers is relatively fixed, and needs to be added, deleted and modified frequently, so it is not suitable for indexing. In the Venue table, the number of remaining seats needs to be constantly updated, which is not suitable for indexing.

The first index selected to be established is customerID and seatID in the Booking table. For customers and ticket sales personnel, it is often needed to check the customer's corresponding seats,

especially before the start of the show. In addition, when modifying the primary keys of the Customer table and the Seat table, the cascading modification can locate the two foreign keys through the index, instead of scanning the entire table, which greatly improves the efficiency. The other two indexes I selected are seatNo and seatPrice in the Seat table. The system will check the seat number and seat price frequently during the transaction. Therefore, in order to increase the transaction speed and reduce the response time when customers buy the tickets or the salesmen operate, seatNo and seatPrice are selected to build an index. The indexing operations are as follows.

```
CREATE NONCLUSTERED INDEX bookingInfo on Booking(customerID,seatID);
```

```
CREATE NONCLUSTERED INDEX seatNo on Seat(seatNo);
```

```
CREATE NONCLUSTERED INDEX seatPrice on Seat(seatPrice);
```

Transaction is an important concept in database. When a transaction is submitted to a DBMS, the DBMS needs to ensure that all operations in the transaction are successfully completed and the results are permanently saved in the database. If some operations in the transaction are not successfully completed, all operations in the transaction need to be rolled back to the state before the transaction is executed. When a company books a venue, the date as a non-empty field which must be recorded, so there will not be a case of no registration. There may be a time conflict that needs to be changed, and you can undo the transaction by rollback the operation. In our database design, all the records in the seat table are already registered, and the customer corresponds to the act in the Booking table. Assuming that there are unregistered tickets now, the unregistered seats can be detected by the same transaction, and rollback can be performed. The sample code of using transactions is as follows.

```
BEGIN TRANSACTION addNewPerformance;
```

```
BEGIN TRY
```

```
INSERT INTO Performance(id, name, type, companyID, venueID, startTime, endTime)
```

```
VALUES (9,'Le Petit Prince','plays',2,2,'2021-1-10 15:30:00', '2021-1-10 18:00:00');
```

```
COMMIT TRANSACTION addNewPerformance;
```

```
END TRY
```

```
BEGIN CATCH
```

```
ROLLBACK TRANSACTION addNewPerformance;
```

```
RAISERROR('Rolling back, an error occurred!', 16, 1);
```

```
END CATCH;
```

```
ROLLBACK;
```

When an error is encountered during the execution of a transaction, the ROLLBACK statement is used to roll the transaction back to the starting point or to a specified hold point. At the same time, all data changes made since the beginning of the transaction or to a savepoint are cleared, and the resources controlled by the transaction are freed. Therefore, this statement also marks the end of the transaction.

### **3 Debate of SQL and NoSQL**

With the rapid development of the Internet, traditional relational databases are unable to cope with super-large scale, super-large traffic, and high concurrency data. At this time, Not Only SQL has developed rapidly. As Not Only SQL (NoSQL) becomes more and more popular today, this section explains the difference between SQL and NoSQL to Abertawe Arts Center upon request. (1) Different storage methods. SQL data is stored in tables, so that rows and columns are easily associated with each other and stored together, making it easy to extract data. Data of NoSQL can be stored in documents, key-value pairs, graphics, etc., which has higher flexibility and scalability. (2) SQL corresponds to structured data. The table structure must be defined before inserting data, which is very reliable and stable. While NoSQL is based on a dynamic structure, and data can be added at any time and anywhere. (3) Relational databases use structured query language to operate the database, while NoSQL operates data in blocks, using an unstructured query language (UnQL). (4) There is no concept of transaction in NoSQL, and the operation of each data set is atomic. (5) The relational database is scaled vertically, which means that if you want to improve the processing power, you must use a faster computer. While the NoSQL database is scaled horizontally, its storage is naturally distributed, and the load can be shared by adding more ordinary database servers to the resource pool.

For the problems raised by Abertawe Arts Center, the data for booking venues and buying tickets is usually structured, and structured queries, such as JOIN, are needed. In addition, the scale and growth rate of data can be expected. But if the data reaches massive amounts, the efficiency of obtaining data by key will be higher. If the company wants to extract customer characteristics from these data, mining valuable information for user portraits, etc., using NoSQL is appropriate. From a cost point of view, most relational databases are expensive and costly, while NoSQL databases are usually free. Therefore, in my opinion, as the number of audiences and partner companies increases, the combination of SQL and NoSQL will make more sense.

### **4 Distributed Database Design**

The design of distributed database can improve system efficiency, increase system reliability and availability. For the design of distributed databases, a top-down design process is also adopted, which requires several steps such as requirement analysis, conceptual design, E-R representation, conversion to relational mode, distributed design, data fragmentation and distribution, and physical design. By slicing global data, you can reduce network transmission, increase processing efficiency of local transactions, and improve data availability and query efficiency. After slicing global data, we can reduce network transmission, increase processing efficiency of local transactions, and improve the availability and query of data efficiency. As the system of Abertawe Arts Center also faces the problem of how to carry out distributed design, this section proposes the following scheme.

For the problem of venue reservation and ticket purchase, based on the previous database design, we first analyze the possible slicing methods of each entity. For performances, different

types of performances usually have different pricing and are in different venues. For example, ordinary band performances are usually in large venues and the prices are lower, while musicals usually use small venues and the prices are higher. Since large companies often hold performances and release performance information, the data of these companies can also be stored in a distributed way, so that these companies can release and update performance information in a time. For customer data, as the number of customers increases, customers of different ages prefer different types of performances. For example, most young people like to watch live while older people like classical music. There are many other horizontal slicing schemes to choose from. According to the previous analysis, my work takes the type of performance as an example, and the horizontal slicing scheme is as follows.

Given a performance relationship  $P \{ id, name, type, companyID, venueID, startTime, endTime \}$ , and a booking relationship  $B \{ id, customerID, seatID \}$ . We can slice according to the conditions. E1. all tuples satisfying  $(companyID = 1)$ . E2. all tuples satisfying  $(companyID = 2)$ . E3. all tuples satisfying  $(companyID < 1 \text{ AND } companyID < 2)$ . According to the definition of horizontal slice, this scheme satisfies  $E1 \cup E2 \cup E3 = P$ . What's more, the sliced data is disjoint, that is to say  $E1 \cap E2 = \emptyset$ ,  $E2 \cap E3 = \emptyset$ ,  $E1 \cap E3 = \emptyset$ . But for the Booking table, the fragmentation is not based on the attributes of the relationship  $P$  itself, but on the attributes associated with it. Taking the first fragmentation as an example, the predicate to derive the horizontal fragmentation can be expressed as  $B1 = \sigma_{companyID = 1}(P) \bowtie B \bowtie Seat$ .

Similarly, the distributed database can also be established based on vertical fragmentation. Generally, there are two application scenarios for vertical fragmentation. If a table has a lot of columns, and only a few columns are commonly used data, we need to split this table into a master table and a son table, so as to avoid too many operations on a wide table. Another application is to split the watch from the same machine to different machines before large-scale business expansion to reduce the pressure on a single machine. In our Abertawe Arts Center system, there are no table has a particularly large number of columns, so vertical fragmentation is not recommended in this case.

Through the design of the distributed database of Abertawe Arts Center, the communication cost of data transmission is reduced, and the response speed is faster under certain conditions. Moreover, compared with supporting a large-scale equipment, this design saves costs and has good scalability. However, it also brings security problems and communication failures, which need to be further resolved.

## 5 Conclusion

Based on the analysis of the requirements of Abertawe Arts Center, this paper analyzes the process of booking venues for the company and booking tickets for customers. Some entities and relations are abstracted out, and a complete database is designed. Then, through the use of indexes, the performance of the database is improved, and two problems raised by the company are solved

through the use of transactions. In the third part, I compare the differences between SQL and NoSQL, and give recommendations for this task. Finally, through the design of distributed database, a reasonable solution is proposed. After the study of this course, I have mastered a more comprehensive and profound knowledge of database. In the future, I will continue to pay attention to the development of database and continue to learn more new technologies.

## References

- [1] Powell, G. (2006). *Beginning database design*. John Wiley & Sons.
- [2] Bagui, S., & Earp, R. (2011). *Database design using entity-relationship diagrams*. Crc Press.
- [3] Bugiotti, F., Cabibbo, L., Atzeni, P., & Torlone, R. (2014, October). Database design for NoSQL systems. In *International Conference on Conceptual Modeling* (pp. 223-231). Springer, Cham.
- [4] Ceri, S., Pernici, B., & Wiederhold, G. (1987). Distributed database design methodologies. *Proceedings of the IEEE*, 75(5), 533-546.
- [5] Teorey, T. J. (1989). Distributed database design: a practical approach and example. *ACM SIGMOD Record*, 18(4), 23-39.
- [6] Stonebraker, M. (2010). SQL databases v. NoSQL databases. *Communications of the ACM*, 53(4), 10-11.
- [7] Li, Y., & Manoharan, S. (2013, August). A performance comparison of SQL and NoSQL databases. In *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)* (pp. 15-19). IEEE.

FINAL GRADE

73/100

GENERAL COMMENTS

Instructor

Good report structure and layout. Evidence of appropriate research and a competent level of understanding displayed throughout each task. Try to indicate you reference sources throughout the document. Your Sql works correctly on reading but you could have attached some example screenshot to you report to demonstrate the transaction for example. Appropriate test data shown. Good analysis of distributed database solution for the case study.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5



Comment 1

Good Analysis of the case study

PAGE 6