Online Retailing and Shopping: An Academical Simulation to Databases

Michal Kvet, Jozef Stasko, Yu-Lin Wang, António Lima
University of Zilina
Zilina, Slovakia
Michal.Kvet@fri.uniza.sk

Teodora Gavrilović
University of Belgrade
Belgrade, Serbia
tg20213005@student.fon.bg.ac.rs

***Abstract*—The amount of data to be handled, modeled, and stored is still rising. Information systems need to cover the complexity by reaching performance. Thus, data retrieval must be properly operated, delimited by several aspects, from the hardware and data storage architecture to data modeling, normalization, indexes, and partitioning techniques. Using one shopping environment, continuous availability must be done to limit financial losses. Moreover, it would cause significant reliability issues. This paper is a result of the University of Belgrade, Serbia, and the UNIZA – University of Zilina, Slovakia cooperation. The overall management deals with relational data modeling. The main emphasis is done on the processing efficiency and overall performance reflected by the normalization, indexing, partitioning and impact of the encryption. The main aim of this paper is to propose robust techniques and discusssion for the data model definition and management.**

I. Introduction

The following article is created in the context of the course of Advanced Database Management from the Faculty of Management and Informatics, UNIZA – the University of Zilina and Database II course on Information System Module from the Faculty of Organizational Sciences, University of Belgrade. The proposed paper is part of the semestral work dealing with performance evaluation and strategy management. The aim is to create a methodology of performance definition and management in a complex online retailing system environment.

We were proposed by our teachers to create and develop a database about any theme we wanted. The group decided to pick the online shopping topic, as this area seemed quite challenging and increased their businesses with the current pandemic. According to the analysis of order data, we can understand more about our customers and find potential business.

As a way of developing our knowledge about the Database topic, we used some of the theories that we learned during our course and practiced during the realization of this project. We were challenged to use some advanced database techniques that contributed to the development of this article [4] [5] [6] [7].

After creating and designing the relational model for Online retailing with different tables and attributes, we developed this model in Oracle [1] [2] [3], where we created the classes and then uploaded the data. This last process was done using Procedures, a subroutine that allows the data insertion in a simplified way [8] [9]. With this principle, they have defined 9 tables that will contain different attributes and registrations. The tables created are *Customer, Image, Invoice, OrderMain, OrderProduct, OrderStatus, PaymentType, Product, and PromoCode*. With this information, it's possible to simulate a full buying process and generate different registrations to the database.

The following step was to generate our data. It was possible with procedures created for all tables with automatic data generation. The only table that didn't have these random values was the table Image, in which the information as web scrapped from a supermarket Tesco Database. With the way we're generating data, a procedure was also developed where we could update instantly our table *OrderMain* with the price of each order. There was the consideration of transforming URL data from VARCHAR to BLOB. For the table Customer, the passwords were encrypted via the DBMS_CRYPTO package.

The model development and implementation also included the task of developing Partitioning for this model was considered because of the size of the database. Our table *OrderProduct* contains a lot of big amounts of data, so this type of division was necessary to make our database more threaded and distributed. For our index's creation and processes, there were 5 indexes called: *ExpensiveOrder*, two *NumberOfOrder* (depending on the following tables: *OrderMain* and *OrderStatus*, *PromoCode,* and *PromoCodeExpiration*.

Our number of indexes depends on the queries that we developed for the simulation process of our database. These queries will be part of our process for inspecting and understanding how the behavior of the model is developed. They consist of: Number of orders in the first quarters of years where order status is paid; The customer with the most expensive order (the biggest amount) grouped by payment type; Top 10 most expensive products; Top 10 orders with the smallest amount to be paid; The order with the most products (overall products not product types); Customer with the most promo codes that give more than 50% discount; Promo codes grouped by % (<20, >20&<40, >40); All promo codes which will be expired by February 2022; Percentage of orders where order status is returned and Top 10 product with most orders

Based on the model review done, we also made some changes to the queries so that we could understand new database distributions. This process was important was critical to understand that we were only working with databases where there was a maximum of 10 products in an order.

With these queries, we could easily understand and compare how a system of online shopping should work. This part is when we enter the model review, where we try to test our database and understand the results. The generated type of data also tried

to diversify and be more accurate. It was in this part that we tested and understood the database, compared the difference (after our partitioning was done). For example, we could understand that our question number 10 could not improve that much was subjected to the partitioning process.

The costs of the different questions helped us identify how the information that we have could affect the normality of our database. It is possible so to understand which of the queries also has the lowest cost and requires less processing. For example, questions 2 and 6 assume their role as the cheapest. These process executions are easier when compared with our other questions, also because of the amount of data he's iterating over to give the result.

There is a limitation where the database doesn't reflect the spent *PromoCodes*. The only performance that was totally measured is related to the queries that were made, their cardinality, and costs. Although, it was also considered the chance of more statistical approaches to the database.

One of our last processes was reviewing what was done, what we learned, and what possible aspects of improving on developing this type of database, presenting all this reflection on this academic paper.

II. Model development

A. Model design

Our first operational process was the creation and design of our database model. In this part we started by deciding the number of tables, what attributes they would contain, and the relationship between them (by also reviewing similar databases that would exist online).

Our number of tables consist of 9, being *Customer, Image, Invoice, OrderMain, OrderProduct, OrderStatus, PaymentType, Product and PromoCode*.

1. The table ***Customer*** is composed by the attributes: *id_customer* (primary key - PK), *email*, *first_name*, last_*name* and *pass*.
2. The table ***Image*** is composed of the attributes: *id_image*, *created*, *content* and *other_data*.
3. The table ***Invoice*** contains the attributes: id_*invoice*, *doc* and *id_order* (foreign key - FK) to the table *Order*).
4. The table ***OrderMain*** is composed of the attributes: *id_order* (PK), *order_time*, *amount*, *id_os* (FK to the table *OrderStatus*, *id_payment* (FK to the table *PaymentType* and *id_customer* (FK to the table *Customer*).
5. The table ***OrderProduct*** has the following attributes: *id* (PK), *id_product* (FK to the table *Product*), *id_order* (FK to table *OrderMain*), and *quantity.*
6. The table ***OrderStatus*** is composed of the attributes: *id_os* (PK) and *name.*
7. The table ***PaymentType*** has the attributes: *id_payment* (PK) and *name*.
8. The table ***Product*** contains the attributes: *id_product* (PK), *name*, *price*, and *id_image* (FK to the table Image).
9. The table ***PromoCode*** has the attributes: *id* (PK), *name*, *percentage*, *expiration*, *id_customer* (FK to the table *Customer*), and *spent.*

B. Model insertion and implementation

(1)Insert data

Related to the model insertion and implementation, after creating the tables in Oracle SQL Server, the next step was data insertion. For this step, we created multiple Procedures, a subroutine that allows the data insertion in a simplified way. Before we ran the procedures, we had an insertion query to generate 100 products to the table *Product* with random prices. For the *Image* table, we inserted the information via a .csv file in that we used 100 URLs.

The first procedure is called *insert_into_customer* and is related to the table *Customer*, and we generated 10000 users, with sequential id, random email, first name, and last name, and then the password. The password was encrypted by using a library from Oracle named *DBMS_CRYPTO*, and the function *hash*. The package *DBMS_CRYPTO* is the correct package to generate hashes. It is not granted to the public by default. You will have to grant it specifically (GRANT EXECUTE ON SYS.DBMS_CRYPTO TO user1). The result of this function is of datatype RAW. You can store it in a RAW column or convert it to VARCHAR2 using the RAWTOHEX or UTL_ENCODE.BASE64_ENCODE functions.

The HASH function is overloaded to accept three datatypes as input: RAW, CLOB and BLOB. Due to the rules of implicit conversion, if you use a VARCHAR2 as input, Oracle will try to convert it to RAW and will most likely fail since this conversion only works with hexadecimal strings. If you use VARCHAR2 then, you need to convert the input to a binary datatype or a CLOB.

The following procedure is named *insert_into_promoCode* for the table *PromoCode* and generated 100000 promo codes with sequential id, random customer, name, expiration date, and percentage. This expiration date will explain if the promo code is still available for being used or not. The next procedure is named *insert_invoice* for the table *Invoice* and this will insert crossed data from *Customer* and *PaymentType* in the *doc* attribute.

The next procedure is called *insert_into_order* to the table *OrderMain* and will generate 10000 orders with sequential id and different random attributes like, the order status, payment type, the customer and the date. This procedure also inserts information in the *Invoice* table. The following procedure is *insert_into_OrderProduct* and will insert 100000 random values from the table *OrderMain* to the table *OrderProduct*, by generating random values to the *order_id*, *product_id* and the own id.

The last procedure is to *update_OrderMain* and has the purpose of updating the amount for the table *OrderMain*, based on the information from the *OrderProduct* table (information from *Product* table).

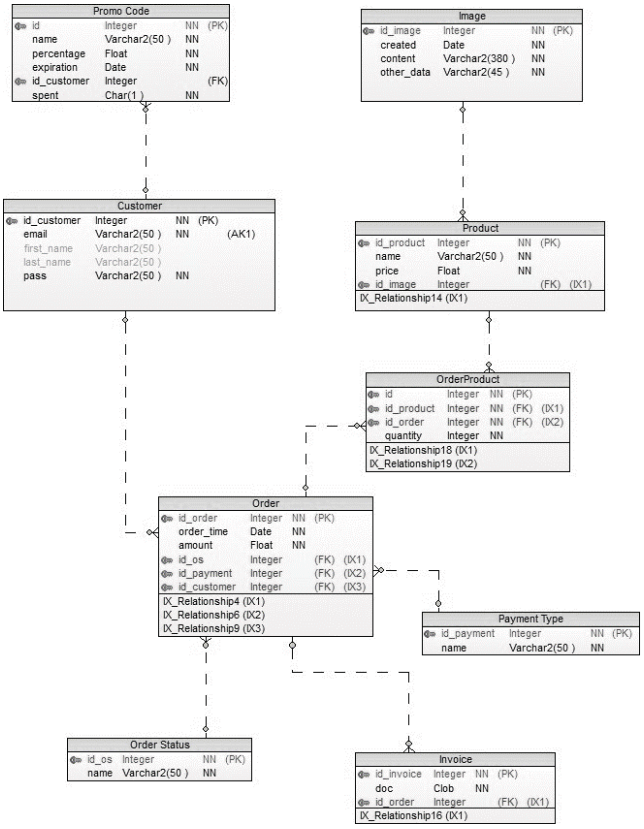

Fig. 1. Relational Model

(2) Partitioning

The Partitioning Concept, as its name suggests, is based on dividing data into smaller portions (partitions) to optimize the database. It allows tables and indexes to be divided, enabling these database objects to be managed and accessed at a finer level of granularity. Oracle provides a rich variety of partitioning strategies and extensions to address different requirements.

Given that the database on which this paper is based is characterized by tables of relatively low complexity and a large number of records, Horizontal Partitioning stands out as a good approach. This scenario involves creating subtables within a defined range. The goal is to increase the efficiency of data access.

The table of greatest interest is the OrderProduct which contains 100,000 records. This amount of data makes it an excellent candidate for partitioning. As the table has only the quantity attribute, in addition to the keys, partitions are created above this field. It is necessary to determine the type of criteria for hierarchical partitioning. The Quantity attribute is an integer value, in the range 1 to 10. Therefore, we conclude that it is appropriate to apply the Range criterion. The number of partitions was chosen to be three:

- Small (27957 records)
- Medium (44525 records)
- Big (27518 records)

The logical aspect of the model and data structures were analyzed, in order to determine the boundary for each partition. Since it is a question of the quantity of products purchased, it was considered that more than 3 products go beyond the scope of a small purchase. Similarly, over 7 copies of the same product are a large number. Statistics of the generated data also support this choice. It can be seen that the first and third partitions have approximately the same number of records, while the second contains about 60% more. By choosing the boundaries differently, the partitions would not be logical wholes and would be less uniform.

It is important to remember that in this case, we are talking about generated data, which, although created as a simulation of a real system, cannot truly mimic it. Therefore, when using real data, it would be necessary to analyze the quantity distribution and change the range if required.

Fig. 2 shows the execution plan before creating the partitions. Partitioning efficiency is monitored based on queries:

```
select *
from orderproduct
 where quantity>5 OR quantity<3;
```

It can be seen that the cost is 105 units, the cardinality is very high, and the execution time is 0.094 sec. Fig. 3 shows the plan after partitions. Unfortunately, it can be seen that the cost is much higher with partitions. On the other hand, the cardinalities that indicate the number of joined rows are much smaller, and the execution time itself is shorter. This shows that this partitioning is not suitable for this type of query. A space in which partitions would prove useful would be complex queries that are limited to values from only one partition, i.e., directly access it.

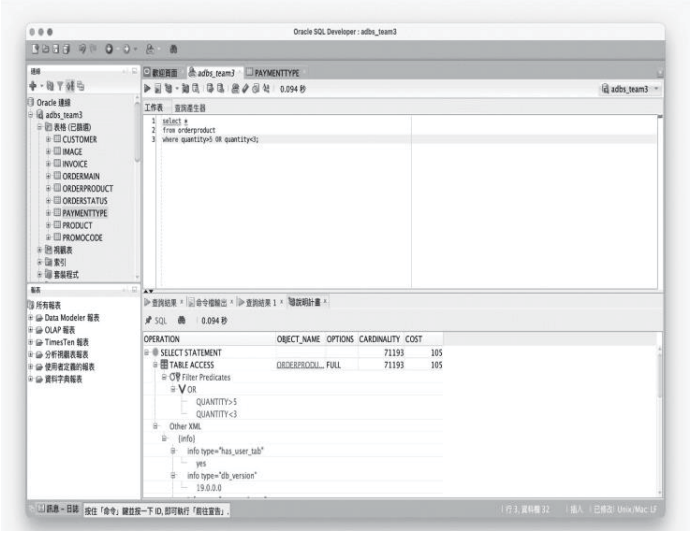

Fig. 2. Question 11 without Partitioning

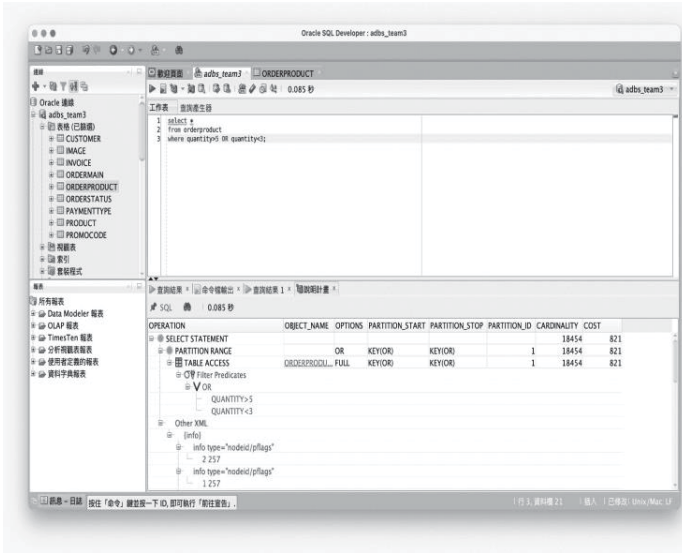

Fig. 3. Question 11 with Partitioning

As can be seen below, accessing data through a partition is much more efficient than directly applying the conditions in the query. It was tested on queries:

```
select *
 from OrderMain
  join OrderProduct
    partition(medium) using(id_order);

select *
 from OrderMain
  join OrderProduct using(id_order)
   where OrderProduct.quantity >3
     and OrderProduct.quantity <8 ;
```

OBJECT_NAME	OPTIONS	CARDINALITY	COST
		94239	619
		94239	619

RODUCT.ID_ORDER

ORDERMAIN	FULL	10000	13
	ITERATOR	94239	605
ORDERPRODUCT	FULL	94239	605

Fig. 4. Question 12 without Partitioning

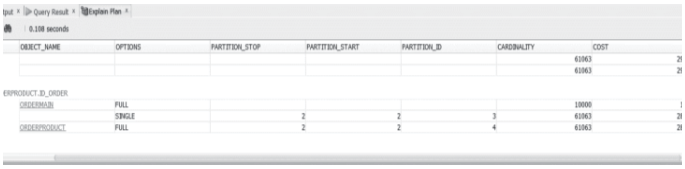

Fig. 5. Question 12 with Partitioning

The conclusion we came to through the partition testing process is that the existence of many records is not a necessary measure of the convenience of partitioning. It is essential to observe the process from the business side as well. Next, analyze what queries are actually used and the distribution of real data.

(3)Index

How to set Indexes in the database depends on queries which will be detailed described in the next section (C. Model Review). Here, we focus on how to decide the index and how is the result.

About Question 1, we can find the query spent works on finding first quarter from attribute *ordertime* in Table *OrderMain* and find the payment type from attribute *name* in Table *PaymentType.* Thus, create index in these two attributes to shorten query time. Compare to Figure 6 and Fig. 7, can see that the running time is 0.105 seconds less and the works query cost from 17 reduced to 16.

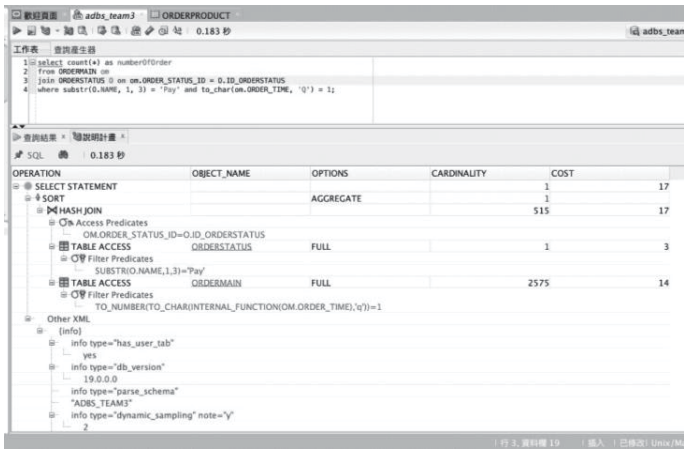

Fig. 6. Question 1

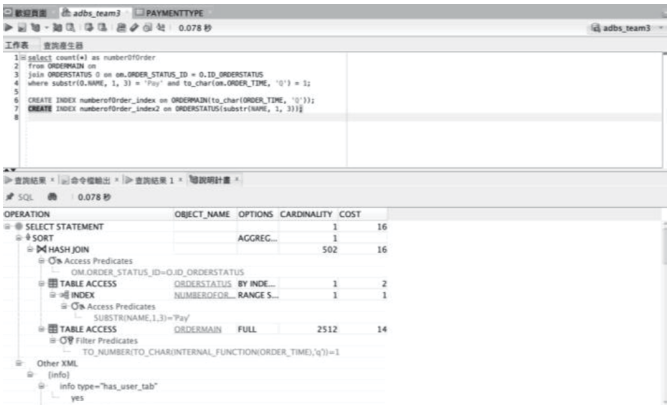

Fig. 7. Question 1 Index

In Question 2, after creating the indexes for attribute *email* in Table *Customer* and attribute *name* in Table *Product*, we can see the processing time from 0.064 seconds (Fig. 8) decrease to 0.036 seconds (Fig. 9). In addition, there is a remarkable point that cost from 626 (Fig. 8) reduced to 3 (Fig. 9). It means that the indexes are really useful.

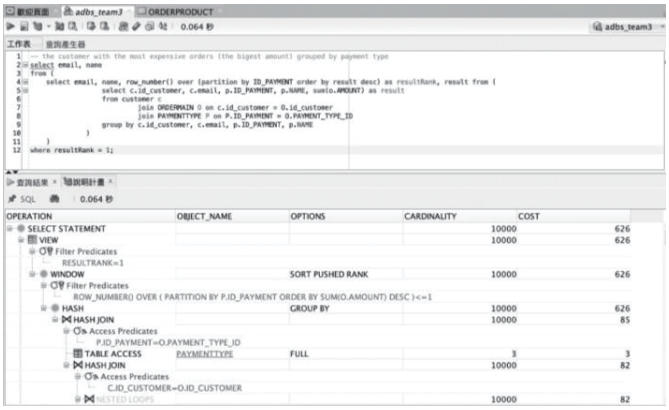

Fig. 8. Question 2

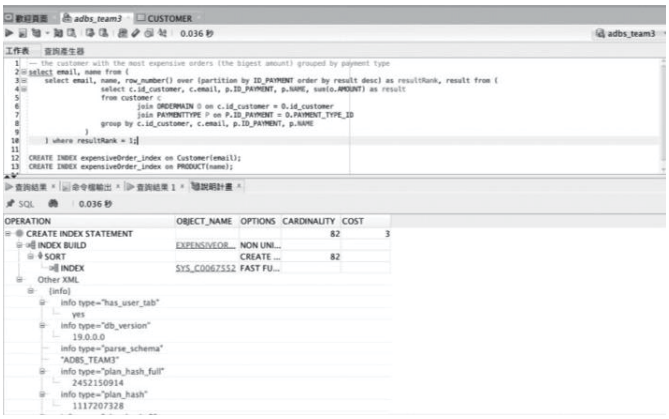

Fig. 9. Question 2 Index

The index about attribute *percentage* in Table *promoCode* helps Question 6(Fig. 10, Fig. 11) and Question 7(Fig. 12, Fig. 13). Especially in Question 6, this index largely improves the processing time and the number of plans.

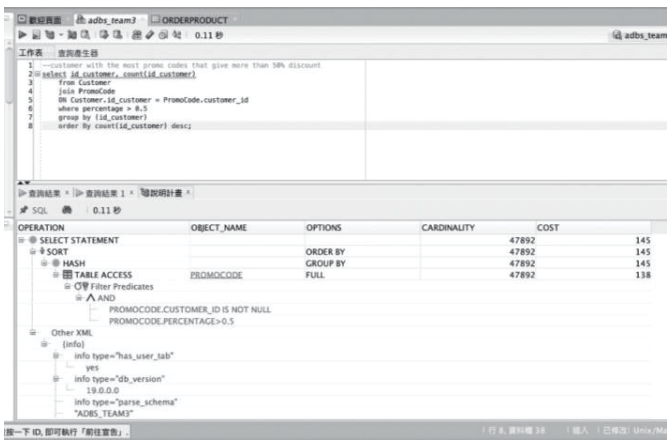

Fig. 10. Question 6

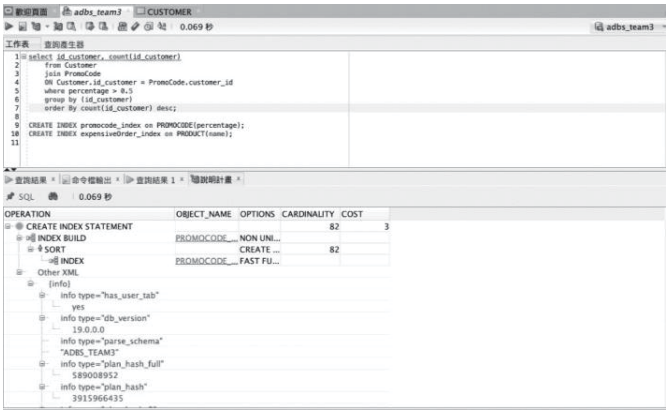

Fig. 11. Question 6 Index

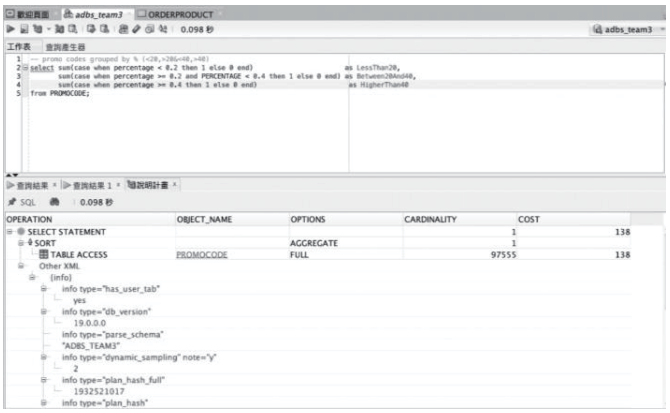

Fig. 12. Question 7

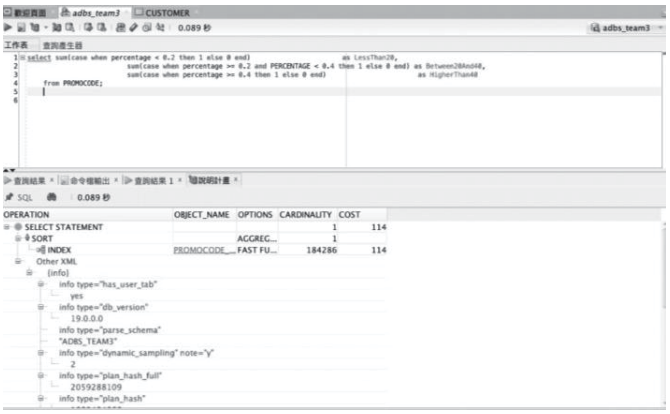

Fig. 13. Question 7 Index

Created an index about attribute *expiration* in Table *promoCode* for Question 8. However, if compared to Figure 14, Fig. 15, and Fig. 16, we can find processing time may decrease, but the cost becomes worse. Even we use parallel to make multiple processes work simultaneously. It doesn't really help. Thus, this index is not useful for the efficiency of the query.

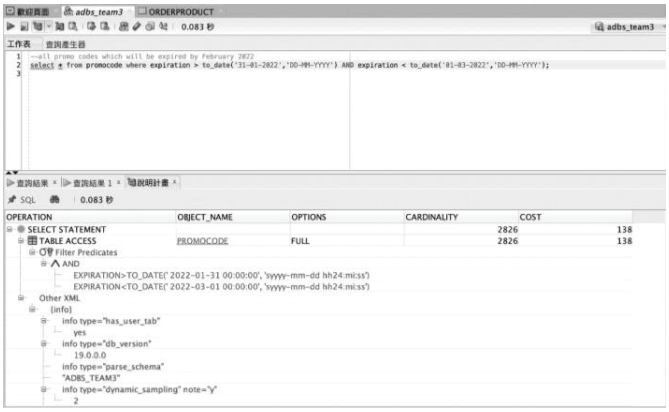

Fig. 14. Question 8

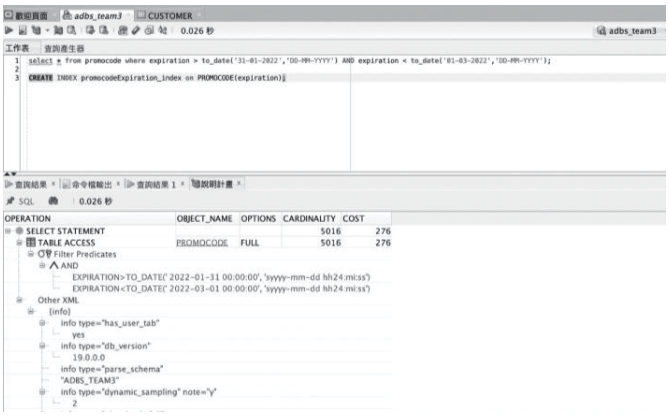

Fig. 15. Question 8 Index

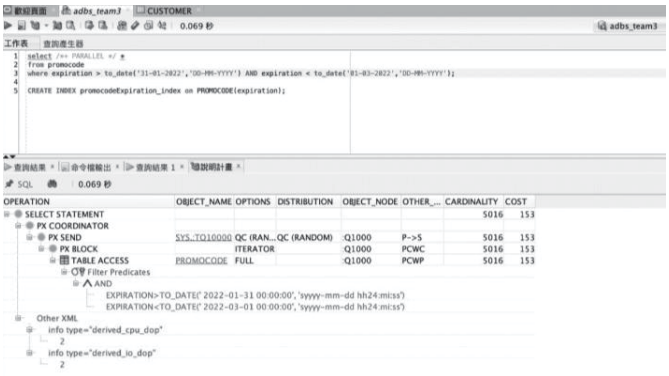

Fig. 16. Question 8 Index Parallel

C. Model review

To analyze the database, we simulate 10 questions that may be useful for business.

1) Number of orders in the first quarter of years that order status is paid;
2) The customer with the most expensive order (the biggest amount) is grouped by payment type;
3) Top 10 most expensive products;
4) Top 10 orders with the smallest amount to be paid
5) The order with the most products;
6) Customer with the most promo codes that give more than 50% discount;
7) Promo codes grouped by % (<20%, >20% & <40%, >40%);
8) All promo codes will be expired by February 2022;
9) Percentage of orders where order status is returned;
10) Top 10 products with most orders.

About Question 1, we can understand 622 orders in the first quarter. If we also analyze other quarters, then we can find out when customers are especially willing to spend money, then set a development strategy for each quartal. For Question 2, saying who cost the most and how much they paid in each payment type. Additionally, we can find customers paid by PayPal cost more than credit cards and cash, showing people prefer paying directly from an account to others. Question 3 shows the company is offering up to 489 euros product.

Question 4 can find the 10 smallest order amounts are from 164 euros to 999 euros. These are the target customers of the company. Question 5 shows the order has 160 products. The total amount is 264 euros. However, in this ranking, since second orders all cost more than 5000 euros per order. It means the order that has the most products can be an exception if a company wants to set a strategy for these customers. Also, if we combine Question 4 and Question 5, we can find that customers in this selling system may have a high demand for products. In Question 6, we can see that the customer has 25 promo codes. However, he didn't use any one of them. Question 7 divided promo codes into three parts, 38,962 lower than 20%, 40,672 promo codes between 20% and 40% and 120,366 promo codes over 40%.

Question 8 shows 5,622 promo codes will expire in February. About 56% in all promo codes, and the percentage of being used promo code is low. Showing customers have great loyalty to the company. They keep ordering from here even without using promo code. In Question 9, there is 25.9% of orders be returned. That means it's better that the company checks the whole supply chain, finds the problems and tries to lower the percentage of it. The last Question shows that top 10 popular products being ordered about 1000-1100 times and the price of each product from 40 to 484 euros. This couldn't reflect the relation between customer preferences and product price, but there is a possibility between product categories and customer preferences.

III. Conclusion

This paper summarizes the process of creating and optimizing a database from the initial idea and model. Observed from the formal aspect, the paper deals with the creation of the model and its realization in the database schema. The data used in the project was generated by team members. The data also contains password encryption as an important part of business logic. The database was further optimized by using horizontal partitioning and applying indexes.

The eventual continuation of work on the project would include additional improvements to the database and creating a user application to which this system applies.

In addition to all the above, this work is a product of the work of an international team within the Erasmus project. For this reason, it is much more than a simulation of an actual business process. During the project, we had the opportunity to understand how to process the same teaching units at different faculties. We managed to apply the theoretically acquired knowledge. A project like this encourages independent research, too. As our initial knowledge differed, we were able to learn during the process, independently but also from each other.

Above all, the work on the project has somewhat managed to bring closer the real business environment that awaits us in the future. Therefore, the conclusion of this project is that it has brought progress in academic, professional, and human terms.

ACKNOWLEDGMENT

This publication was realized with the support of the project: Accelerating the transition towards Edu 4.0 in HEIs, TEACH4EDU4, funded under Erasmus+ programme - KA203 Strategic Partnerships for higher education.

REFERENCES

[1] Bryla, B.: Oracle Database 12c The Complete Reference, Oracle Press, 2013, ISBN – 978-0071801751

[2] Burleson, D. K.: Oracle High-Performance SQL Tuning, Oracle Press, 2001, ISBN - 9780072190588

[3] Delplanque, J., Etien, A., Anquetil, N., Auverlot, O.: Relational database schema evolution: An industrial case study, IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Spain, 2018, pp. 635-644

[4] Dudáš A., Škrinárová J., Kiss A.: On graph coloring analysis through visualization. In: Information and Digital Technologies 2021 : proceedings of the international conference. pp. 71-78. ISBN 978-1-6654-3692-2.

[5] Eisa, I., Salem, R., Abdelkader, H.: A fragmentation algorithm for storage management in cloud database environment, Proceedings of ICCES 2017 12th International Conference on Computer Engineering and Systems, Egypt, 2018

[6] Kvet, M. (2019). Complexity and Scenario Robust Service System Design. In Information and Digital Technologies 2019: conference proceedings, Žilina, 2019, ISBN 978-1-7281-1400-2, pp. 271-274.

[7] Kvet, M.: Managing, locating and evaluating undefined values in relational databases. 2020

[8] Steingartner, W., Eged, J., Radakovic, D., Novitzka, V.: Some innovations of teaching the course on Data structures and algorithms. In 15th International Scientific Conference on Informatics, 2019.

[9] Zhang, K., et al., 2019, Efficient public-key encryption with equality test in the standard model, In Theoretical Computer Science, 755, 65-80.