Testing Report – Student #2

* **Group Number**: C1.034
* **Repository**: https://github.com/CarlosCerdaMorales/dp2-c1-034
* **Workgroup Members** **and Corporate Emails:**

Loubna Founoun Elaoud: [loufouela@alum.us.es](mailto:loufouela@alum.us.es)

Carlos Cerdá Morales: [carcermor@alum.us.es](mailto:carcermor@alum.us.es)

José Luis Moraza Vergara: [josmorver@alum.us.es](mailto:josmorver@alum.us.es)

Claudia Oviedo Govantes: [claovigov@alum.us.es](mailto:claovigov@alum.us.es)

Sergio Cantillo Blanco: [sercanbla@alum.us.es](mailto:sercanbla@alum.us.es)

* **Date**: Sunday 25th, May 2025

**Table Of Contents**

[1. Executive Summary 3](#_451ewtno8qvm)

[2. Revision Table 3](#_ov3mb6vwut6t)

[3. Introduction 3](#_r2bem3ub6oc7)

[4. Contents 3](#_nh08ol2sdh4r)

[4.1 Functional Testing 3](#_h69on2nqubkm)

[4.2 Performance Testing 3](#_xn4c25wc84mn)

[5. Conclusions 3](#_2l2wgk3amr06)

[6. Bibliography 3](#_a255vkbpqjez)

# Executive Summary

In this report, I present the functional and performance testing I carried out on my student. The objective was to ensure that all features behave as expected and to assess how quickly the system responds under normal conditions.

For the functional testing, I organized the test cases by feature. Each case targets a specific functionality and was essential in verifying that the application behaves correctly.

For the performance testing, I followed the methodology outlined in the session guide: I collected execution times from the .trace files and processed the data using Excel. I generated charts and calculated 95% confidence intervals to evaluate whether the system’s response times remain within acceptable limits. The tests were executed in two different configurations: one using the database without any additional indexing, and another with relevant indexes applied. I then performed a statistical comparison between both setups to determine the impact of indexing on performance.

In summary, this report reflects the testing I have conducted on the key features of the application, supported by performance data that provides a solid understanding of how the system behaves in real-world conditions.

# Revision Table

|  |  |  |
| --- | --- | --- |
| **Revision Number** | **Date** | **Description** |
| 1.0 | 25/05/2025 | Final version |

# Introduction

The purpose of this report is to document the testing I carried out on my project, covering both functionality and performance. My goal was to ensure that each feature works as intended and that the system can handle requests efficiently.

To achieve this, I first ran a set of functional tests grouped by feature, verifying that the application behaved as expected. Then, I focused on performance testing by analyzing how long the system took to respond to various requests. This helped me identify potential bottlenecks and compare performance across different configurations.

Overall, this testing phase gave me a clearer understanding of how the system behaves under real-world conditions.

# Contents

## 4.1 Functional testing

For these tests, I evaluated the functionalities that my project, Acme-ANS, provides to customers, specifically regarding their bookings and passengers. The tests are documented through trace files located in the /src/test/resources/customer/booking and /src/test/resources/customer/passenger directories. All test cases achieved 100% coverage, as shown in the image below.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto. These are the trace files that demonstrate the code coverage, included just below, and created following the recommendations provided in the course. I tested both .safe traces—covering positive and negative test cases—and .hack traces, which simulate attempts to hack the application. All hacking attempts were correctly handled and resulted in an "Access Unauthorised" response.

In the following sections, the specific test cases are specified and how each was addressed.

Tabla

El contenido generado por IA puede ser incorrecto.

|  |  |
| --- | --- |
| Customer Booking features | |
| CREATE SERVICE SAFE | |
| TC-01: Create an empty booking | An attempt was made to create an empty booking, which correctly triggered error messages from the application, indicating that certain fields "must not be null". |
| TC-02: INVALID LOCATOR CODE | An invalid locator code was submitted when creating a booking. The code did not match the expected pattern and violated both lower and upper boundary constraints. |
| TC-03: REPEATED LOCATOR CODE | A locator code that already existed in the database was used in a booking creation attempt. The system properly rejected the request, as duplicate locator codes are not allowed. |
| TC-04: VALID LOCATOR CODE | A locator code that complied with the required pattern was tested and was correctly accepted by the system. |
| TC-05: INVALID TRAVEL CLASS | An attempt was made to create a booking with a null travel class, which the system correctly rejected as invalid input. |
| TC-06: VALID TRAVEL CLASS | The same test was conducted using the two available travel class values, business and economy. Both were accepted successfully by the system. |
| TC-07: INVALID LAST NIBBLE | Attempts were made to store the lastNibble field with values breaking the pattern by exceeding upper and lower limits, as well as including letters and non-numeric characters. These attempts were rejected by the database. |
| TC-08: VALID LAST NIBBLE | A lastNibble value conforming exactly to the required 4-digit pattern was tested and correctly accepted by the application. |
| TC-09: INVALID FLIGHT | An attempt was made to create a booking without assigning any flight, resulting in an error indicating that the flight field must not be null. |
| TC-10: VALID FLIGHT | An attempt was made to assign a valid flight not in draft mode within the application, which functioned correctly as expected. |
| BUGS | No bugs were found while recording these tests. |
| CREATE SERVICE HACK | |
| TC-11: INVALID ID FOR CREATE | An attempt was made to modify the id field of the new booking being created, aiming to overwrite an existing booking in the database instead of creating a new one. This attempt resulted in an "Access Unauthorised" response from the application. |
| TC-12: INVALID TRAVEL CLASS | Attempts were made to modify the travelClass field with values not accepted by the enumeration, including numbers, empty strings, and other invalid entries. All these attempts resulted in an "Access Unauthorised" response from the application. |
| TC-13: INVALID FLIGHT | An attempt was made to assign an unpublished flight to the booking, resulting in an "Access Unauthorised" response from the application. The same protocol and handling were applied when attempting to assign a flight that does not exist in the database. |
| BUGS | Bugs were encountered when attempting to fully cover TC-12, as some values received from the view could not be entirely tested. Eventually, complete coverage was achieved, and unauthorized access is now properly handled. |
| DELETE SERVICE SAFE | |
| TC-14: DELETE BOOKING | An attempt was made to delete a booking. Since the only restriction is that it must be the user’s own booking, the deletion was performed successfully after verifying this condition. |
| BUGS | No bugs were found while recording these tests. |
| DELETE SERVICE HACK | |
| TC-15: DELETE NON-EXISTENT BOOKING | An attempt was made to delete a non-existent booking by accessing the /delete endpoint with an ID that does not exist in the database. The application correctly handled the request and responded with an "Access Unauthorised" error. |
| TC-16: DELETE BOOKING NOT MINE | An attempt was made to delete a booking that did not belong to the current user by accessing the /delete endpoint with a valid ID. The system correctly identified the unauthorized action and responded with an "Access Unauthorised" error. |
| TC-17: DELETE PUBLISHED BOOKING | An attempt was made to delete a booking that belonged to the current user but was already published. The application correctly rejected the operation, responding with an "Access Unauthorised" error, as published bookings cannot be deleted. |
| BUGS | No bugs were found while recording these tests. |
| LIST SERVICE SAFE | |
| TC-18: LIST MY BOOKINGS | An attempt was made to list the bookings of a customer. This operation was always successful, as the database query is executed based on the ID of the currently logged-in user. |
| BUGS | No bugs were found while recording these tests. |
| LIST SERVICE HACK | |
| THERE IS NO hacking attempt applicable to this specific FEATURE. | |
| PUBLISH SERVICE SAFE | |
| TC-19: PUBLISH an empty booking | An attempt was made to publish an empty booking, which correctly triggered error messages from the application, indicating that certain fields "must not be null". |
| TC-20: INVALID LOCATOR CODE | An invalid locator code was submitted when publishing a booking. The code did not match the expected pattern and violated both lower and upper boundary constraints. |
| TC-21: REPEATED LOCATOR CODE | A locator code that already existed in the database was used in a booking publishing attempt. The system properly rejected the request, as duplicate locator codes are not allowed. |
| TC-22: VALID LOCATOR CODE | A locator code that complied with the required pattern was tested and was correctly accepted by the system. |
| TC-23: INVALID TRAVEL CLASS | An attempt was made to publish a booking with a null travel class, which the system correctly rejected as invalid input. |
| TC-24: VALID TRAVEL CLASS | The same test was conducted using the two available travel class values, business and economy. Both were accepted successfully by the system. |
| TC-25: INVALID LAST NIBBLE | Attempts were made to store the lastNibble field with values breaking the pattern by exceeding upper and lower limits, as well as including letters and non-numeric characters. These attempts were rejected by the database. |
| TC-26: VALID LAST NIBBLE | A lastNibble value conforming exactly to the required 4-digit pattern was tested and correctly accepted by the application. |
| TC-27: EMPTY LAST NIBBLE | An attempt was made to publish a booking without the lastNibble being stored. This resulted in an error, as the requirements state that only bookings with a properly stored lastNibble can be published. |
| TC-28: INVALID FLIGHT | An attempt was made to create a booking without assigning any flight, resulting in an error indicating that the flight field must not be null. |
| TC-29: VALID FLIGHT | An attempt was made to assign a valid flight not in draft mode within the application, which functioned correctly as expected. |
| TC-30: EMPTY FLIGHT | An attempt was made to publish a booking with no flight assigned. This resulted in a controlled error triggered by the service’s validator. |
| TC-31: NO PASSENGERS | An attempt was made to publish a booking without any passengers assigned. This resulted in a controlled error triggered by the service’s validator. |
| TC-32: VALID PUBLISH | Finally, a booking was successfully published after ensuring that all required fields were valid and properly completed, and all restrictions were checked. |
| BUGS | Bugs were encountered when attempting TC-30. A validation error occurred during the price calculation process, as the flight field was null. |
| PUBLISH SERVICE hack | |
| TC-33: PUBLISH NON-EXISTENT ID | An attempt was made to access the /publish URI with a booking ID that does not exist. The system correctly handles it with an "access unauthorised" response. |
| TC-34: PUBLISH ALREADY PUBLISHED | An attempt was made to access the /publish URI with the ID of an already published booking, resulting in an "unauthorised" response. |
| tc-35: publish not mine | An attempt was made to publish a booking that does not belong to the client in session, resulting in an authorisation error. |
| tc-36: publish invalid travel class | An attempt was made to publish a booking with a tampered travel class field, resulting in Access Unauthorised. |
| TC-37: PUBLISH INVALID FLIGHT | An attempt was made to publish a booking with an invalid flight, resulting in Access Unauthorised. |
| BUGS | No bugs were found while recording these tests. |
| SHOW SERVICE SAFE | |
| TC-38: show valid booking | An attempt was made to display a valid booking, as it belongs to the current client. |
| BUGS | No bugs were found while recording these tests. |
| show service hack | |
| tc-39: show non-existent booking | An attempt was made to display a booking that does not exist in the database, resulting in Access Unauthorised. |
| tc-40: show other customer’s booking | An attempt was made to display another client's booking, resulting in Access Unauthorised, regardless of whether it is published or not, since data from other clients must not be visible. |
| UPDATE SERVICE SAFE | |
| TC-41: UPDATE an empty booking | An attempt was made to update an empty booking, which correctly triggered error messages from the application, indicating that certain fields "must not be null". |
| TC-42: INVALID LOCATOR CODE | An invalid locator code was submitted when updating a booking. The code did not match the expected pattern and violated both lower and upper boundary constraints. |
| TC-43: REPEATED LOCATOR CODE | A locator code that already existed in the database was used in a booking update attempt. The system properly rejected the request, as duplicate locator codes are not allowed. |
| TC-44: VALID LOCATOR CODE | A locator code that complied with the required pattern was tested and was correctly accepted by the system. |
| TC-45: INVALID TRAVEL CLASS | An attempt was made to update a booking with a null travel class, which the system correctly rejected as invalid input. |
| TC-46: VALID TRAVEL CLASS | The same test was conducted using the two available travel class values, business and economy. Both were accepted successfully by the system. |
| TC-47: INVALID LAST NIBBLE | Attempts were made to store the lastNibble field with values breaking the pattern by exceeding upper and lower limits, as well as including letters and non-numeric characters. These attempts were rejected by the database. |
| TC-48: VALID LAST NIBBLE | A lastNibble value conforming exactly to the required 4-digit pattern was tested and correctly accepted by the application. |
| TC-49: INVALID FLIGHT | An attempt was made to update a booking without assigning any flight, resulting in an error indicating that the flight field must not be null. |
| TC-50: VALID FLIGHT | An attempt was made to assign a valid flight not in draft mode within the application, which functioned correctly as expected. |
| BUGS | No bugs were found while recording these tests. |
| UPDATE SERVICE hack | |
| TC-51: UPDATE NON-EXISTENT ID | An attempt was made to access the /update URI with a booking ID that does not exist. The system correctly handles it with an "access unauthorised" response. |
| TC-52: UPDATE ALREADY PUBLISHED | An attempt was made to access the /update URI with the ID of an already published booking, resulting in an "unauthorised" response. |
| tc-53: UPDATE not mine | An attempt was made to update a booking that does not belong to the client in session, resulting in an authorisation error. |
| tc-54: UPDATE invalid travel class | An attempt was made to publish a booking with a tampered travel class field, resulting in Access Unauthorised. |
| TC-55: UPDATE INVALID FLIGHT | An attempt was made to publish a booking with an invalid flight, resulting in Access Unauthorised. |
| BUGS | No bugs were found while recording these tests. |

|  |  |
| --- | --- |
| Customer passenger features | |
| CREATE SERVICE SAFE | |
| TC-56: Create an empty PASSENGER | An attempt was made to create a passenger without filling in the fields, resulting in several "may not be null" warnings. |
| TC-57: INVALID FULL NAME | An attempt was made to create a passenger with a full name that does not comply with the upper limit constraints. |
| TC-58: VALID full name | A passenger was successfully created with a full name within the specified range, including non-Latin characters, without any issues. |
| TC-59: INVALID email | An attempt was made to enter an email that is not considered valid. |
| TC-60: VALID email | A valid email was entered for the creation of a passenger. |
| TC-61: INVALID passport | An attempt was made to enter a passport that does not match the specified pattern, violating both upper and lower limits. |
| TC-62: repeated passport | An attempt was made to enter a passport that already exists, resulting in a proper error. |
| TC-63: invalid BIRTH DATE | An attempt was made to enter a future date of birth when creating a passenger, resulting in a failure, as a passenger cannot be born in the future. |
| TC-64: VALID special needs | An attempt was made to create a passenger with empty special needs, as it is an optional field. Upper limits were also tested and correctly violated. |
| BUGS | No bugs were found while recording these tests. |
| CREATE SERVICE HACK | |
| TC-65: INVALID ID FOR CREATE | An attempt was made to modify the id field of the new passenger being created, aiming to overwrite an existing passenger in the database instead of creating a new one. This attempt resulted in an "Access Unauthorised" response from the application. |
| BUGS | No bugs were found while recording these tests. |
| DELETE SERVICE SAFE | |
| TC-66: DELETE passenger | An attempt was made to delete a passenger. Since the only restriction is that it must be the user’s own passenger, the deletion was performed successfully after verifying this condition. |
| BUGS | No bugs were found while recording these tests. |
| DELETE SERVICE HACK | |
| TC-67: DELETE NON-EXISTENT passenger | An attempt was made to delete a non-existent passenger by accessing the /delete endpoint with an ID that does not exist in the database. The application correctly handled the request and responded with an "Access Unauthorised" error. |
| TC-68: DELETE passenger NOT MINE | An attempt was made to delete a passenger that did not belong to the current user by accessing the /delete endpoint with a valid ID. The system correctly identified the unauthorized action and responded with an "Access Unauthorised" error. |
| TC-69: DELETE PUBLISHED passenger | An attempt was made to delete a passenger that belonged to the current user but was already published. The application correctly rejected the operation, responding with an "Access Unauthorised" error, as published passengers cannot be deleted. |
| BUGS | No bugs were found while recording these tests. |
| LIST SERVICE SAFE | |
| TC-70: LIST MY PASSENGERS | An attempt was made to list the passengers of a customer. This operation was always successful, as the database query is executed based on the ID of the currently logged-in user. |
| TC-71: LIST PASSENGERS FROM BOOKING | The passengers of a booking were listed by accessing through the booking entity, correctly reaching the associated passengers. |
| BUGS | No bugs were found while recording these tests. |
| LIST SERVICE HACK | |
| TC-72: LIST PASSENGERS FROM non-existent booking | An attempt was made to list the passengers of a non-existent booking, resulting in Access Unauthorised from the application. |
| TC-73: LIST PASSENGERS FROM BOOKING WHICH IS NOT MINE | An attempt was made to list the passengers of a booking that does not belong to me, resulting in Access Unauthorised. |
| BUGS | Some bugs were found while testing TC-73, since the view was not being able to retrieve the bookingId from the service. |
| PUBLISH SERVICE SAFE | |
| TC-74: PUBLISH an empty PASSENGER | An attempt was made to publish a passenger without filling in the fields, resulting in several "may not be null" warnings. |
| TC-75: INVALID FULL NAME | An attempt was made to publish a passenger with a full name that does not comply with the upper limit constraints. |
| TC-76: VALID full name | A passenger was successfully published with a full name within the specified range, including non-Latin characters, without any issues. |
| TC-77: INVALID email | An attempt was made to enter an email that is not considered valid. |
| TC-78: VALID email | A valid email was entered for the publishing of a passenger. |
| TC-79: INVALID passport | An attempt was made to enter a passport that does not match the specified pattern, violating both upper and lower limits. |
| TC-80: repeated passport | An attempt was made to enter a passport that already exists, resulting in a proper error. |
| TC-81: invalid BIRTH DATE | An attempt was made to enter a future date of birth when publishing a passenger, resulting in a failure, as a passenger cannot be born in the future. |
| TC-82: VALID special needs | An attempt was made to publish a passenger with empty special needs, as it is an optional field. Upper limits were also tested and correctly violated. |
| BUGS | No bugs were found while recording these tests. |
| PUBLISH SERVICE hack | |
| TC-83: publish NON-EXISTENT passenger | An attempt was made to publish a non-existent passenger by accessing the /publish endpoint with an ID that does not exist in the database. The application correctly handled the request and responded with an "Access Unauthorised" error. |
| TC-84: publish passenger NOT MINE | An attempt was made to publish a passenger that did not belong to the current user by accessing the /publish endpoint with a valid ID. The system correctly identified the unauthorized action and responded with an "Access Unauthorised" error. |
| TC-85: publish PUBLISHED passenger | An attempt was made to delete a passenger that belonged to the current user but was already published. The application correctly rejected the operation, responding with an "Access Unauthorised" error, as published passengers cannot be published. |
| BUGS | No bugs were found while recording these tests. |
| SHOW SERVICE SAFE | |
| TC-86: show valid passenger | An attempt was made to display a valid passenger, as it belongs to the current client. |
| BUGS | No bugs were found while recording these tests. |
| show service hack | |
| tc-87: show non-existent passenger | An attempt was made to display a passenger that does not exist in the database, resulting in Access Unauthorised. |
| tc-88: show other customer’s pasenger | An attempt was made to display another client's passenger, resulting in Access Unauthorised, regardless of whether it is published or not, since data from other clients must not be visible. |
| UPDATE SERVICE SAFE | |
| TC-89: Create an empty PASSENGER | An attempt was made to update a passenger without filling in the fields, resulting in several "may not be null" warnings. |
| TC-90: INVALID FULL NAME | An attempt was made to update a passenger with a full name that does not comply with the upper limit constraints. |
| TC-91: VALID full name | A passenger was successfully updated with a full name within the specified range, including non-Latin characters, without any issues. |
| TC-92: INVALID email | An attempt was made to enter an email that is not considered valid. |
| TC-93: VALID email | A valid email was entered for the creation of a passenger. |
| TC-94: INVALID passport | An attempt was made to enter a passport that does not match the specified pattern, violating both upper and lower limits. |
| TC-95: repeated passport | An attempt was made to enter a passport that already exists, resulting in a proper error. |
| TC-96: invalid BIRTH DATE | An attempt was made to enter a future date of birth when updating a passenger, resulting in a failure, as a passenger cannot be born in the future. |
| TC-97: VALID special needs | An attempt was made to update a passenger with empty special needs, as it is an optional field. Upper limits were also tested and correctly violated. |
| BUGS | No bugs were found while recording these tests. |
| UPDATE SERVICE hack | |
| TC-98: update NON-EXISTENT passenger | An attempt was made to update a non-existent passenger by accessing the /update endpoint with an ID that does not exist in the database. The application correctly handled the request and responded with an "Access Unauthorised" error. |
| TC-99: update passenger NOT MINE | An attempt was made to update a passenger that did not belong to the current user by accessing the /update endpoint with a valid ID. The system correctly identified the unauthorized action and responded with an "Access Unauthorised" error. |
| TC-100: update PUBLISHED passenger | An attempt was made to update a passenger that belonged to the current user but was already update. The application correctly rejected the operation, responding with an "Access Unauthorised" error, as published passengers cannot be updated. |
| BUGS | No bugs were found while recording these tests. |

|  |  |
| --- | --- |
| MORE FEATURES | |
| ADDING PASSENGERS TO BOOKINGS SAFE | |
| TC-101: ADD AN EMPTY PASSENGER | An attempt was made to create a passenger without filling in the field resulting in "may not be null" warnings. |
| TC-102: ADD A PASSENGER | A valid passenger was correctly added to a booking. |
| BUGS | No bugs were found while recording these tests. |
| ADDING PASSENGERS TO BOOKINGS HACK | |
| TC-103: INVALID ID FOR CREATE | An attempt was made to modify the id field of the new relationship being created, aiming to overwrite an existing relation in the database instead of creating a new one. This attempt resulted in an "Access Unauthorised" response from the application. |
| TC-104: ADDING A NON-EXISTING PASSENGER | An attempt was made to add a passenger that does not exist in the database, resulting in Access Unauthorised. |
| TC-105: ADDING NOT MINE PASSENGER | An attempt was made to add a passenger that does not belong to me, resulting in Access Unauthorised. |
| TC-106: ADDING ALREADY IN BOOKING PASSENGER | An attempt was made to add a passenger who is already in the booking, resulting in Access Unauthorised. |
| TC-107: ADDING NOT PUBLISHED PASSENGER | An attempt was made to add a passenger that belongs to me but is not published, resulting in Access Unauthorised. |
| BUGS | No bugs were found while recording these tests. |
| deleting PASSENGERS from BOOKINGS SAFE | |
| TC-108: DELETING AN EMPTY PASSENGER | An attempt was made to delete a passenger without filling in the field resulting in "may not be null" warnings. |
| TC-109: DELEETING A PASSENGER | A valid passenger was correctly deleted from the booking. |
| BUGS | No bugs were found while recording these tests. |
| DELETING PASSENGERS FROM BOOKINGS HACK | |
| TC-110: deleting A NON-EXISTING PASSENGER | An attempt was made to delete a passenger that does not exist in the database, resulting in Access Unauthorised. |
| TC-111: DELETE NOT MINE PASSENGER | An attempt was made to delete a passenger that does not belong to me, resulting in Access Unauthorised. |
| TC-112: ADDING NOT PUBLISHED PASSENGER | An attempt was made to add a passenger that belongs to me but is not published, resulting in Access Unauthorised. |
| BUGS | No bugs were found while recording these tests. |
|  |  |

## 4.2 Performance testing

Performance testing was conducted on the same computer. For this purpose, all the previously shown traces were executed twice: first without database indexes, and a second time with indexes, aiming to improve performance by reducing the query resolution time. Below, the timing tables of both analyses will be shown:

Gráfico, Gráfico de barras, Gráfico en cascada

Descripción generada automáticamente

Gráfico, Gráfico de barras

Descripción generada automáticamente

**Conclusion on the Impact of Indexes on Performance**

In this timing analysis conducted on two instances — one without indexes and another with indexes — it was observed that the presence of indexes did not improve performance but instead caused a slight delay in the queries. This can be attributed to several reasons:

1. **Small size of the test data:**  
   In scenarios with a small amount of data, indexes may not be effective because the cost of maintaining and querying the index can outweigh the benefit of avoiding a full table scan. In small databases, a full scan is usually fast and straightforward.
2. **Additional cost of index maintenance:**  
   Creating and updating indexes adds overhead to the database, especially during insert, update, or delete operations. In tests where many similar queries are executed or frequent changes occur, this overhead can be reflected in longer response times.
3. **Specific characteristics of the queries and indexes:**  
   Not all indexes benefit all queries. If queries are not optimized to leverage existing indexes, or if the indexes do not cover the appropriate columns, the database engine may not efficiently use them.

After processing and cleaning the collected data, a descriptive statistical analysis was performed on both traces, gathering various data shown below:

Tabla

Descripción generada automáticamente

**Interfaz de usuario gráfica, Aplicación, Tabla, Excel

Descripción generada automáticamente**

For both studies, the confidence intervals are very similar and close in value, indicating consistent results. However, there is a noticeable slight tendency for the traces recorded with indexes to take a bit more time compared to those without indexes. This suggests that, in these tests, the presence of indexes may introduce a small overhead affecting performance.

Finally, the z-value was calculated for both sets of traces to statistically compare their performance. The results indicate that there is practically no significant improvement in query execution times when indexes are added. In fact, this analysis suggests a slight tendency for queries executed with indexes to take a bit longer than those without indexes. This could be due to the overhead involved in maintaining and accessing the indexes, especially in the context of the relatively small dataset used for these tests. Therefore, according to this study, the addition of indexes does not provide a meaningful performance benefit and may even introduce a minor performance penalty in the tested scenarios.

Interfaz de usuario gráfica, Aplicación, Tabla, Excel

Descripción generada automáticamente

It can be clearly observed that the average execution time in the second analysis, which includes indexes, is higher than in the first analysis without indexes. This indicates that adding indexes does not improve performance; rather, it appears to introduce additional overhead that results in longer query times in this specific testing scenario.

# Conclusions

The performance analysis comparing query execution times with and without indexes revealed that, under the tested conditions, adding indexes did not provide a significant improvement. The small dataset and specific query patterns likely contributed to the minimal or negative impact observed. Both the descriptive statistics and the calculated z-value showed that queries with indexes tend to take slightly longer, suggesting an overhead associated with index maintenance and usage. To better understand the true benefits of indexing, further testing with larger, more realistic datasets and varied query types is recommended.

# Bibliography

If there’s no relevant bibliography, write “intentionally blank”.