



ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

INTERNATIONAL HELLENIC UNIVERSITY
DEPARTMENT OF INFORMATION AND ELECTRONIC ENGINEERING

THESIS

Implementation of a Bibliometric Database for Greek Computer Science Departments



Student: Gavriil Ilikidis
Student ID: 175127

Supervisor:
Antonis Sidiropoulos

23 May 2023

Title of Dissertation
Implementation of a Bibliometric Database for Greek Computer Science Departments
Code of Dissertation 24262
Student's full name Gavriil Ilikidis
Supervisor's full name Antonis Sidiropoulos
Date of undertaking 18-10-2024
Date of completion 2025

We hereby affirm the authorship of this paper as well as the acknowledgement and credit of whichever assistance We received in its composition. We have, furthermore, noted the various sources from which We extracted data, ideas, visual or written material, in paraphrase or exact quotation. Moreover, we affirm the exclusive composition of this paper by myself only, for the purpose of it being a dissertation, in the Department of Information and Electronic Engineering of the I.H.U.

This paper constitutes the intellectual property of Gavriil Ilikidis the student that composed it. According to the open-access policy, the author/composer offers the International Hellenic University authorisation to use the right to reproduce, borrow, publicly present and digitally distribute the paper globally, in electronic form and media of all kinds, for teaching or research purposes, voluntarily. Open access to the full text, by no means grants the right to trespass the intellectual property of the author/composer, nor does it authorise the reproduction, republication, duplication, selling, commercial use, distribution, publication, downloading, uploading, translation, modification of any kind, in part or summary of the paper, without the explicit written consent of the authors.

The approval of this dissertation by the Department of Information and Electronic Engineering of the International Hellenic University, does not necessarily entail the adoption of the author's views, on behalf of the Department.

Dedication

I would like to dedicate this thesis to the people who supported and encouraged me throughout this journey.

To my family, for their unwavering love, patience, and belief in me, your support has been the foundation of all my achievements.

To my partner, for standing by me with understanding and motivation during long hours of work and moments of stress.

To my friends, whose presence and encouragement reminded me to keep a healthy balance and stay optimistic.

And finally, to my thesis supervisor, for his valuable guidance, technical insight, and continuous support throughout the development of this work.

Thank you all for being part of this chapter in my life.

Prolog

This thesis was carried out as part of the requirements for the completion of my undergraduate studies in Computer Science at the International Hellenic University. It represents the culmination of several months of research, design, coding, and data analysis, during which I had the opportunity to apply knowledge gained throughout my academic journey and explore new technological challenges in depth, both as a student and an aspiring software engineer.

The core motivation behind this work was to develop a robust and scalable system capable of collecting and processing academic metrics from Google Scholar. From designing the database schema and writing efficient scraping scripts, to constructing stored procedures and building a RESTful API, the project allowed me to engage with both theoretical and practical aspects of backend development.

Beyond the technical goals, this experience was also helpful in strengthening important skills such as problem-solving, debugging, performance optimization, and documentation. I am particularly grateful for the support of my supervising professor and the feedback I received during the development process, which helped guide my decisions and refine the final result.

May this work serve as a foundation for future improvements, as well as a useful tool for academic institutions interested in bibliometric data analysis and researcher evaluation.

Abstract

This thesis presents the design and implementation of a complete backend system for the collection, storage, and analysis of bibliometric data sourced from Google Scholar. At the core of the system lies a custom-built web scraper, developed in Python, which is capable of dynamically navigating and extracting structured academic data from publicly available profiles.

The scraper handles various edge cases and encoding inconsistencies (such as non-ASCII characters), performs validation checks on dates and titles, and ensures that data integrity is preserved before insertion into the database. The collected data is stored in a normalized relational schema and analyzed through optimized SQL stored procedures that compute a wide range of metrics per researcher, department, or institution.

A RESTful API built in PHP offers secure and efficient access to these metrics, enhanced with token-based authentication and GZIP compression for performance. Visualized results from real academic data demonstrate the system's ability to produce meaningful insights for institutional evaluation and planning. The architecture is modular and extensible, making it suitable for integration with future frontend platforms or additional data sources.

Περίληψη

Η παρούσα εργασία εξετάζει την ανάπτυξη ενός αυτοματοποιημένου συστήματος για τη συλλογή και διαχείριση βιβλιομετρικών δεδομένων στα Τμήματα Πληροφορικής των ελληνικών πανεπιστημίων, με στόχο τη στήριξη της τεκμηριωμένης αξιολόγησης της ερευνητικής τους δραστηριότητας. Αξιοποιώντας τις δημόσια διαθέσιμες πληροφορίες του Google Scholar, η προτεινόμενη προσέγγιση εφαρμόζει τεχνικές web scraping για την άντληση δεδομένων από τα προφίλ των μελών Διδακτικού και Ερευνητικού Προσωπικού (ΔΕΠ), όπως είναι ο αριθμός των δημοσιεύσεων, οι ετεροαναφορές, η χρονολογική κατανομή των εργασιών και δείκτες επίδρασης όπως το h-index και το i10-index.

Τα δεδομένα αποθηκεύονται σε μια σχεσιακή βάση δεδομένων, σχεδιασμένη έτσι ώστε να είναι επεκτάσιμη και να υποστηρίζει σύνθετα στατιστικά ερωτήματα. Η υλοποίηση συνοδεύεται από ένα RESTful API γραμμένο σε PHP, μέσω του οποίου είναι δυνατή η πρόσβαση στα δεδομένα και η αξιοποίησή τους από μελλοντικές εφαρμογές ή εργαλεία ανάλυσης. Σκοπός της εργασίας είναι να καλύψει το υφιστάμενο κενό στην οργανωμένη και συγκριτική αποτίμηση της ερευνητικής παραγωγής σε εθνικό επίπεδο, προσφέροντας ένα λειτουργικό και επαναχρησιμοποιήσιμο σύστημα, το οποίο μπορεί να αξιοποιηθεί τόσο από μεμονωμένα Τμήματα όσο και από φορείς εκπαιδευτικής πολιτικής.

Contents

1	Introduction	2
2	Literature Review	4
2.1	Bibliometric indicators and research review	4
2.2	Existing tools and platforms for capturing bibliometric data	5
2.2.1	Google Scholar	5
2.2.2	Scopus and Web of Science	6
2.2.3	ORCID	6
2.3	Web Scraping approaches for academic information	6
2.4	Data and infrastructure for Greek Universities	7
2.5	Existing Application: The OMEA Citations Platform	8
2.5.1	Structure of the Existing Database	8
2.5.2	User Interface of the OMEA Platform	9
3	Technologies and Software Tools	13
3.1	Python Programming Language	13
3.1.1	Selenium	14
3.2	Relational Databases and SQL	15
3.2.1	Relational Databases	15
3.2.2	SQL	16
3.2.3	Third Normal Form (3NF)	18
3.3	PHP	18
3.3.1	REST API	19
3.4	Supporting Tools and Environments	20
3.4.1	HeidiSQL	20
3.4.2	MySQL Workbench	20
3.4.3	IntelliJ IDEA	21
3.4.4	Visual Studio Code	21
3.4.5	Linux Server	21
3.4.6	Termius	21
3.4.7	SSH (Secure Shell)	21
3.4.8	SCP (Secure Copy Protocol)	21
3.4.9	Postman	21
3.4.10	XeLaTeX	21
3.4.11	Overleaf	22
4	System Architecture	23
4.1	Google Scholar Scraper	23
4.1.1	Synchronisation with the database	25

4.1.2	Calculation of indicators	25
4.1.3	Troubleshooting and CAPTCHA	26
4.1.4	Scraper Structure and Execution Flow	27
4.1.5	Scalability and Future Usage	30
4.2	Design and Implementation of Database	31
4.2.1	Design Principles and Methodology	31
4.2.2	Database Schema Overview	32
4.2.3	Description of Entities and Tables	33
4.2.4	Entity–Relationship Diagram with Triggers and History Tables	43
4.2.5	Stored Procedures and Data Access Logic	45
4.2.6	Limitations, Extensibility and Future Enhancements	51
4.3	Design and Operation of the API	52
4.3.1	Architecture and Structure of the API	53
4.3.2	Types of Requests (Endpoints)	54
4.3.3	Token-based Access and GZIP Encoding	56
4.4	Extensibility	57
5	Results	58
5.1	Collected Data and Indicative Results	58
5.1.1	Tables	58
5.1.2	Stored Procedures	67
5.2	API Responses	75
6	Discussion	80
6.1	Comparison between the Design and Implementation and the Initial Objectives	80
6.2	Technical and Design Decisions	80
6.3	Problems Encountered and Solutions Implemented	81
6.4	Reproducibility and System Extensibility	82
6.5	Overall Assessment and Development Experience	82
7	Conclusion	84
7.1	Overall Conclusions	84
7.2	Outlook and Future Plans	85
7.3	Concluding Evaluation	85
	Bibliography	86

List of Figures

2.5.1 Home Page of OMEA Platform	10
2.5.2 Faculty Members Page of OMEA Platform	11
2.5.3 Departments Statistics Page of OMEA Platform	12
3.1.1 Python and Its Creator	13
3.2.1 Example of many-to-many relationship between staff and publications	16
3.2.2 Donald D. Chamberlin and Raymond F. Boyce	16
4.1.1 Scraper basic Flow Diagram	23
4.1.2 Example citation graphs from Google Scholar. The exact citation counts for each year are displayed interactively when hovering the mouse over the bars.	26
4.1.3 Main execution flow of the Google Scholar Scraper	28
4.2.1 ER Diagram with Triggers and History Tables	44
4.3.1 Request Flow for the Publications of a Staff Member	54
5.1.1 Vertical Bar Chart of the Number of Members for 10 Departments	70
5.1.2 Vertical Bar Chart of Total Publications for 10 Departments	70
5.1.3 Vertical Bar Chart of Total Citations for 10 Departments	71
5.1.4 Vertical Bar Chart of Average H-index for 10 Departments	71
5.1.5 Vertical Bar Chart of Average i10-index for 10 Departments	72
5.1.6 Vertical Bar Chart of Average Academic Age of Members for 10 Departments	72
5.2.1 Example JSON response from GET /departments/all	76
5.2.2 Example JSON response from GET /roles	76
5.2.3 Example JSON response from GET /staff/stats/token?departments=<token> &roles=<token>	77
5.2.4 Example JSON response from GET /staff/publications/per-year?token=<token>	77
5.2.5 Example JSON response from GET /staff/citations/per-year?token=<token>	78
5.2.6 Example JSON response from GET /staff/overall?token=<token>	78
5.2.7 Example JSON response from GET /departments/statistics/token?departments= <token>&roles=<token>	79

List of Tables

4.1.1	Used Libraries in Python Scraper	24
4.2.1	Structure of staff table	33
4.2.2	Structure of departments table	34
4.2.3	Structure of role table	35
4.2.4	Structure of staff_dept_role table	36
4.2.5	Structure of publications table	37
4.2.6	Structure of publications_staff table	38
4.2.7	Structure of staff_statistics table	39
4.2.8	Structure of staff_citations_per_year table	40
4.2.9	Structure of publication_citations_per_year table	41
4.2.10	Structure of staff_records table	42
4.2.11	Example data row for the stored procedure get_all_staff_summary()	46
4.2.12	Example data for the stored procedure get_overall_stats_by_staff_ids()	47
4.2.13	Example data row for the stored procedure get_department_stats_by_depts_- and_roles()	48
4.2.14	Non-basic stored procedures available for extended or future use.	50
4.3.1	Mapping of API Endpoints to MySQL Stored Procedures	56
5.1.1	Example Department Records	58
5.1.2	Departments and Universities	59
5.1.3	Roles	61
5.1.4	Example of Faculty Members Entries	61
5.1.5	Example Mapping of Staff Members to Departments and Roles	62
5.1.6	Publication Data	63
5.1.7	Example Mapping of Staff to Publications and Author Order	63
5.1.8	Staff Statistics	64
5.1.9	Citations per Year for Each Staff Member	65
5.1.10	Citations per Year for Each Publication	66
5.1.11	Overall Aggregated Staff Statistics	67
5.1.12	Example Result of Stored Procedure get_department_stats_by_depts_and_- roles() for two Departments and all the Roles	68
5.1.13	Result returned by the stored procedure get_all_staff_summary()	73
5.1.14	Publication Details for a Specific Member	74
5.1.15	Result of stored procedure get_publications_per_year_by_staff for staff ID 76	75
5.1.16	Result of stored procedure get_citations_per_year_by_staff for staff ID 76	75

Chapter 1

Introduction

The evaluation and systematic documentation of scientific work have become an essential aspect of contemporary academic life. In today's academic landscape, particularly within the field of Computer Science, where research is both highly competitive and rapidly evolving the need to accurately track and assess the scholarly output of faculty and researchers is more critical than ever.

Although significant steps have been made in the digital transformation of higher education institutions in Greece, many Computer Science Departments still lack a cohesive and automated infrastructure for collecting and analyzing bibliometric data [1]. In many cases, departments continue to depend on manual methods or a patchwork of disparate tools, making the process time-consuming and unsuitable for the demands of long-term academic planning and strategic development [2].

Even though Google Scholar is among the most widely used platforms for tracking and documenting scientific output, it does not provide an official, open and freely accessible API for data extraction. This limitation turns the retrieval of key metrics such as the number of publications, citation counts and impact indicators like the h-index or i10-index into a largely manual and potentially error-prone task. The absence of automated access poses considerable challenges, particularly when dealing with large volumes of data, as is often necessary at the departmental level.

This thesis project aims to address this gap by presenting the development of an automated internal system for the collection and management of bibliometric data in Greek Computer Science Departments, using Google Scholar as the main data source. The system is based on web scraping techniques, which enable the extraction of relevant information directly from the public profiles of faculty members, removing the need for manual data entry. The retrieved data are stored in a relational database and made accessible through a dedicated REST API developed in PHP. This setup allows for potential integration with other systems or user interfaces, while also supporting comparative analysis, statistical processing and long-term monitoring of research activity at the departmental level.

The structure of this thesis is organized into seven chapters. Chapter 1 introduces the objectives, motivation, and scope of the project. Chapter 2 presents a review of related literature regarding bibliometric indicators, academic evaluation systems, and existing approaches to extracting data from platforms such as Google Scholar. Chapter 3 outlines the technologies and tools used for the development of the system, including the programming languages, frameworks, and supporting environments. In Chapter 4, the system architecture is examined in detail, covering the scraper design, database schema, ER diagram, stored procedures, triggers, and the RESTful API used to access the data. Chapter 5 presents selected results extracted from the implemented system,

including indicative data, aggregated statistics, and visualizations. Chapter 6 provides a critical discussion of the technical decisions and implementation challenges. Finally, Chapter 7 concludes the thesis by summarizing the key findings and suggesting directions for future work.

Chapter 2

Literature Review

2.1. Bibliometric indicators and research review

Bibliometrics is a field that focuses on the quantitative analysis of scientific publications and reports, aiming to shed light on how knowledge is produced and disseminated. It is widely applied as a tool for assessing research activity, offering metrics that seek to reflect the scientific performance of individuals, research teams, institutions, or even entire academic disciplines[3].

The role of bibliometric indicators in modern academic evaluation has become increasingly important. These metrics are now integral to a wide range of processes, including:

- the internal assessment of university departments,
- the evaluation and promotion of faculty members,
- the development of institutional and international rankings,
- the formulation of educational policies,
- and the preparation of research funding proposals.

While bibliometric indicators are not a substitute for qualitative evaluation methods such as peer review, they are increasingly being used as a complementary tool in various countries, academic systems, and institutional practices. Notably, countries like the Netherlands, the United Kingdom, and Finland have integrated bibliometric data into their processes for allocating research funding and other academic resources [4].

Main bibliometric indices

The main indices collected and used in this thesis project are presented below:

- **h-index (Hirsch Index):** This is the most commonly used metric that combines both the quantity and the impact of a researcher's work. A researcher has an h-index of h if they have published h papers, each of which has received at least h citations. For example, an h-index of 10 means that the researcher has 10 publications with at least 10 citations each.
 - *Advantages:* Simple to calculate, and reflects both productivity and influence.
 - *Disadvantages:* Does not account for citations beyond the h threshold, does not reflect the level of individual contribution to co-authored works, and may disadvantage early-career researchers[5].

- **i10-index:** Introduced by Google Scholar, this metric counts how many of a researcher's publications have received at least 10 citations. While it serves as a useful complementary indicator, its use is almost entirely limited to the Google Scholar platform.
- **Total citations:** This is the sum of all citations received across a researcher's body of work. Although it offers a general sense of impact, it can be misleading in cases where self-citations or excessive citations inflate the numbers.
- **5-year h-index / citations:** These metrics focus on the researcher's output and impact within the most recent five-year period. They offer a more current view of research activity, which is particularly relevant in the context of ongoing evaluations and evolving academic priorities.

The use of bibliometric indicators, although increasingly common, is not without important challenges and limitations. One of the main issues is that most indicators do not reflect the specific contribution of each author to a publication, leading to uncertainty about the actual role of the researcher being assessed [5]. In addition, comparisons across academic disciplines can be problematic, as each field has distinct norms regarding publication frequency and citation practices [6]. For instance, authorship and citation patterns in Computer Science differ significantly from those in fields such as Biomedicine or the Humanities.

Another concern involves the potential manipulation of bibliometric metrics, either through excessive self-citation or through citation circles within small, tightly connected scientific communities [7]. Furthermore, these indicators often fail to account for key qualitative aspects of research, such as its originality, interdisciplinary nature or broader societal and technological impact [8]. For these reasons, while bibliometric measures are undoubtedly useful tools for evaluation, their use should be approached with care and always complemented by qualitative, content-based assessment methods.

2.2. Existing tools and platforms for capturing bibliometric data

A variety of digital platforms and tools have been developed in response to the growing need for organised and systematic documentation of scientific output. These tools and platforms make it easier to track publications, generate reports, and calculate performance indicators. The ORCID platform, Web of Science, Scopus, and Google Scholar are some of the most widely used systems. Each of these offers a unique combination of features, advantages, and disadvantages that should be carefully considered in light of the particular requirements and goals of the intended application.

2.2.1. Google Scholar

Google Scholar is a freely available and widely used search engine for scientific literature. It enables researchers to create personal profiles that display their publications, citation counts, and bibliometric indicators such as the h-index and i10-index. One of its key strengths is its extensive coverage, as it includes not only journal articles but also conference proceedings, doctoral dissertations, preprints and book chapters. This broad scope makes it especially valuable in technical disciplines like computer science, where a significant portion of scholarly work is published outside traditional journals.

However, Google Scholar also presents several important limitations that affect its reliability as a bibliometric resource. A key concern is the lack of a public API, which restricts automated access to data and forces researchers to rely on web scraping techniques or time-intensive manual methods for large-scale data collection [9]. In addition, the platform may display inaccuracies in author profiles and publication records, as it does not implement a robust system for document verification or author disambiguation [10]. Another issue is the susceptibility of its metrics to manipulation. Citation counts and indexes such as the h-index can be artificially inflated through self-citation or the use of automated uploads, raising questions about the overall reliability of the platform's data [11]. These limitations underline the importance of interpreting Google Scholar metrics with care, particularly when used for formal research evaluation.

Nevertheless, it remains the most accessible and comprehensive option, particularly in academic settings where there is no access to commercial subscription-based systems.

2.2.2. Scopus and Web of Science

Scopus (Elsevier) and Web of Science (Clarivate Analytics) are among the most established commercial platforms for the collection and analysis of bibliometric data. These systems offer structured metadata, unique author identifiers, and a range of performance metrics, including journal indicators such as SNIP and SJR [12]. In addition, they support data export through commercial APIs, providing enhanced functionality for institutions with access to these services [13], [14].

These platforms offer several important advantages, including a high level of accuracy in recording bibliographic information and a lower risk of errors in author identification [12]. They also provide access to well-established bibliometric indicators and comprehensive journal statistics. Supported by specialised documentation teams, these systems benefit from continuous updates and improvements, ensuring the reliability and relevance of the data they offer.

However, access to these platforms requires a paid subscription, which can pose a significant barrier for independent academic initiatives at the undergraduate or postgraduate level, as well as for research projects that lack dedicated funding [15]. Additionally, their coverage is not comprehensive across all disciplines. In fields such as computer science, where conference proceedings often hold equal or greater importance than journal publications, these systems may not fully capture the breadth of scholarly output [16].

2.2.3. ORCID

The ORCID (Open Researcher and Contributor ID) platform provides a unique identifier for each researcher and allows the consolidation of their publications across multiple systems. While it is not a bibliometric database in the strict sense, it functions as a central hub for author identification and can be integrated with platforms such as Google Scholar, Scopus, Web of Science, and various institutional repositories [17].

Its primary value lies in reducing errors in author identification, particularly for researchers with common names [18]. However, it does not generate bibliometric metrics or analytical figures on its own.

2.3. Web Scraping approaches for academic information

The lack of formal, free, and fully accessible APIs in widely used bibliometric platforms, such as Google Scholar, has contributed to the growing reliance on web scraping techniques for the extraction of scientific data. Web scraping refers to the process of automatically retrieving content

from web pages to analyze and utilize the information they contain, and it has become the primary method in situations where structured access to data through official channels is unavailable [19].

In the case of Google Scholar, which does not provide a free public API, the use of scraping tools becomes the only practical option for efficiently gathering data such as publication titles, citation counts, publication dates, and metrics like the h-index and i10-index. A key technical challenge, however, is that the platform is designed to limit automated access. It employs measures such as CAPTCHA verification, rate limiting, and, in some cases, temporary IP bans when access patterns do not comply with its usage policies.

Despite these limitations, the use of modern tools such as the Selenium library for Python makes it possible to simulate human browsing behavior. Selenium supports the rendering of JavaScript-based pages, navigation through multiple tabs, and extraction of content using XPath or CSS selectors. When combined with carefully managed delays, controlled iteration, and thorough error logging, the process becomes robust enough to support the systematic collection of bibliometric data, even at the scale of entire academic departments [20].

In this study, web scraping is applied to Google Scholar in a focused and systematic manner. The process begins by accessing the public profile of each faculty member, from which both overall metrics and detailed information for each publication are retrieved. The collected data are then organized into structured tables, sorted by year and author, and stored in a relational database for further analysis. Although this approach involves certain technical challenges, it offers a fully automated and scalable solution, enabling the collection of bibliometric information without requiring manual input.

2.4. Data and infrastructure for Greek Universities

The collection, analysis and evaluation of scientific activity is an increasingly important priority within the Greek academic landscape, particularly in the context of institutional accountability, strategic planning and international university rankings [21]. Although there has been progress in the digital transformation of higher education, most Greek universities still lack unified and fully automated bibliometric systems for their faculty [22]. Information is often scattered, some appear on departmental websites, some on external platforms such as Google Scholar or Scopus, and in many cases, publications are entered manually. This fragmentation makes it difficult to generate reliable statistics or conduct meaningful comparisons across departments and institutions [23].

At the same time, the inconsistency in the quality and structure of available data makes it challenging to form a clear and comprehensive overview of each institution's research activity [23]. In many cases, departments do not maintain an official list of faculty members with linked profiles on platforms such as Google Scholar or ORCID. Furthermore, the recording of publications is often left to individual discretion, without any formal validation process. As a result, it becomes difficult to assess a department's research output or scientific impact based on a stable, long-term and verifiable dataset.

Establishing an internal, institution-wide system for the collection of bibliometric data can provide a range of important benefits [21]. These include greater transparency and accountability, more informed strategic planning, and stronger representation of departments in international evaluations and rankings. Such a system also allows for the generation of structured, comparable reports based on department, year or individual researcher, thereby contributing meaningfully to academic documentation and institutional development.

In this context, Google Scholar was selected as the primary data source for the present study due to its broad coverage of published research in the field of Computer Science. This discipline

typically includes both journal articles and conference papers, and Google Scholar reflects this publication pattern to a reasonable extent. Its free access and availability to all users also make it a practical choice for public universities in Greece, even though it has certain limitations in terms of data quality and author identification.

2.5. Existing Application: The OMEA Citations Platform

2.5.1. Structure of the Existing Database

The existing platform [OMEA Citations (<https://omea.iee.ihu.gr/citations>)] is based on a simple relational database, consisting of a few tables with limited relationships between them and no historical data or dynamic change tracking.

Below is an overview of the main tables:

Departments

- Lists the departments of Greek universities.
- Fields:
 - id: unique identifier (e.g. cs@ihu)
 - deptname, University: department/university names
 - URLs to the department site or other sources (urldep, urledip, url)
- It is the "reference" table for the association of faculty members with the respective department.

dep

- The main table that stores the data of each faculty member.
- Fields:
 - gsid: unique Google Scholar ID - serves as primary key.
 - name, position: full name and position (e.g. Professor, Lecturer)
 - inst: reference to the ID of the department
 - Bibliometrics:
 - * hindex, citations, publications: overall
 - * hindex5, citations5, publications5: last 5 years
- Contains aggregate metrics by person, but no reference to individual publications.
- There is no possibility of historical analysis or tracking value changes (e.g., when the h-index increased).
- It is overburdened: it contains identifiers, personal data, and metrics, resulting in a lack of separation of concerns.

citations and publications

- They contain the number of citations and publications per year (cyear) for each gsid.
- It is not possible to track which paper had how many citations, nor to correlate with metadata (title, journal, authors).

notingsdep

- Contains former members or entries that are not actively joining the system.
- Fields: name, position, inst.
- There is no gsid or metrics.
- It cannot be functionally connected to the other tables.

Despite the functionality of the system in terms of capturing key bibliometric indicators, the database design presented significant limitations on multiple levels.

First, there was no provision for repeatability or historical tracking. Key indicators such as the h-index, number of citations and number of publications were stored as static values, with no mechanism to record changes over time. There were also no triggers or control tables to record updates, making longitudinal analysis impossible.

In addition, the data were stored at an overly aggregated level, with no detail at the level of individual publications or citations. Citations and publications were simply recorded as annual counters, with no possibility of analysing the data by paper, author, journal or other relevant metadata. This lack of detail severely limited the possibility of meaningful evaluation of research output.

These design deficiencies were reinforced by the weak relational structure. There was no normalized entity representing publications as discrete objects (including fields such as title, authors, etc.), nor a mapping table linking faculty members to publications (e.g., to account for co-authorship or author order). As a result, the system was difficult to extend and carried a high risk of data redundancy and inconsistency in future development.

Overall, the architecture of the old system served a basic, static display of indicators, but did not meet the requirements of a modern, dynamic and scalable bibliometric analysis environment.

2.5.2. User Interface of the OMEA Platform

The OMEA Citations platform website presented a static and simple-to-use interface, focused on the display of aggregated bibliometric data for faculty members of Computer Science Departments of Greek universities. Its main function was to display members by department and to show key indicators (such as h-index, citations, publications) overall and by five-year period.

Home Page

The home page contained a basic list of the available Departments/Universities and of the roles. Upon selecting departments and roles, a table of all related researchers was displayed.

← → ↻ <https://omea.iee.ihu.gr/citations/> 🔍 ☆ ⚙️ | Sign in ...

Google Scholar citations [See departments' stats](#)

☒ Professor ☒ Associate Professor ☒ Assistant Professor
☒ Lecturer ☒ Lab Lecturer ☐ Former member [\[select all\]](#)

Click column head to sort

☐ ceid@upatras ☐ cs@aueb ☐ cs@ihu ☐ cs@ionio ☐ cs@unipi ☐ cs@uoc
☐ cs@uoi ☐ cs@uowm ☐ cs@uth ☐ csd@auth ☐ dai@uom ☐ di@uoa
☐ dib@uth ☐ dit@hua ☐ dit@uoi ☐ dit@uop ☐ ds@unipi ☐ ds@uop
☐ ds@uth ☐ ece@hmu ☐ ece@ntua ☐ ece@tuc ☐ ece@uop
☐ ece@uowm ☐ ece@upatras ☐ ece@uth ☐ ee@auth ☐ ee@duth
☐ ee@hmu ☐ eee@uniwa ☐ ice@uniwa ☐ icsd@aegean ☐ ict@ihu
☒ iee@ihu [\[select all\]](#) [\[clear all\]](#)

Figure 2.5.1: Home Page of OMEA Platform

Table of faculty members

The table contained columns such as name, position (e.g., Professor, Lecturer), total publications, total citations, h-index, corresponding indicators for the last 5 years, and annual publications/citations. The page also provides combined statistics for the selected members, such as:

- Total Members
- Total Publications and in the last 5 years
- Total Citations and in the last 5 years
- Average h-index and in the last 5 years
- Average Publications per Member
- Average Citations per Member
- Average Academic Age

- Average Publications per Member per Year
- Average Citations per Member per Year

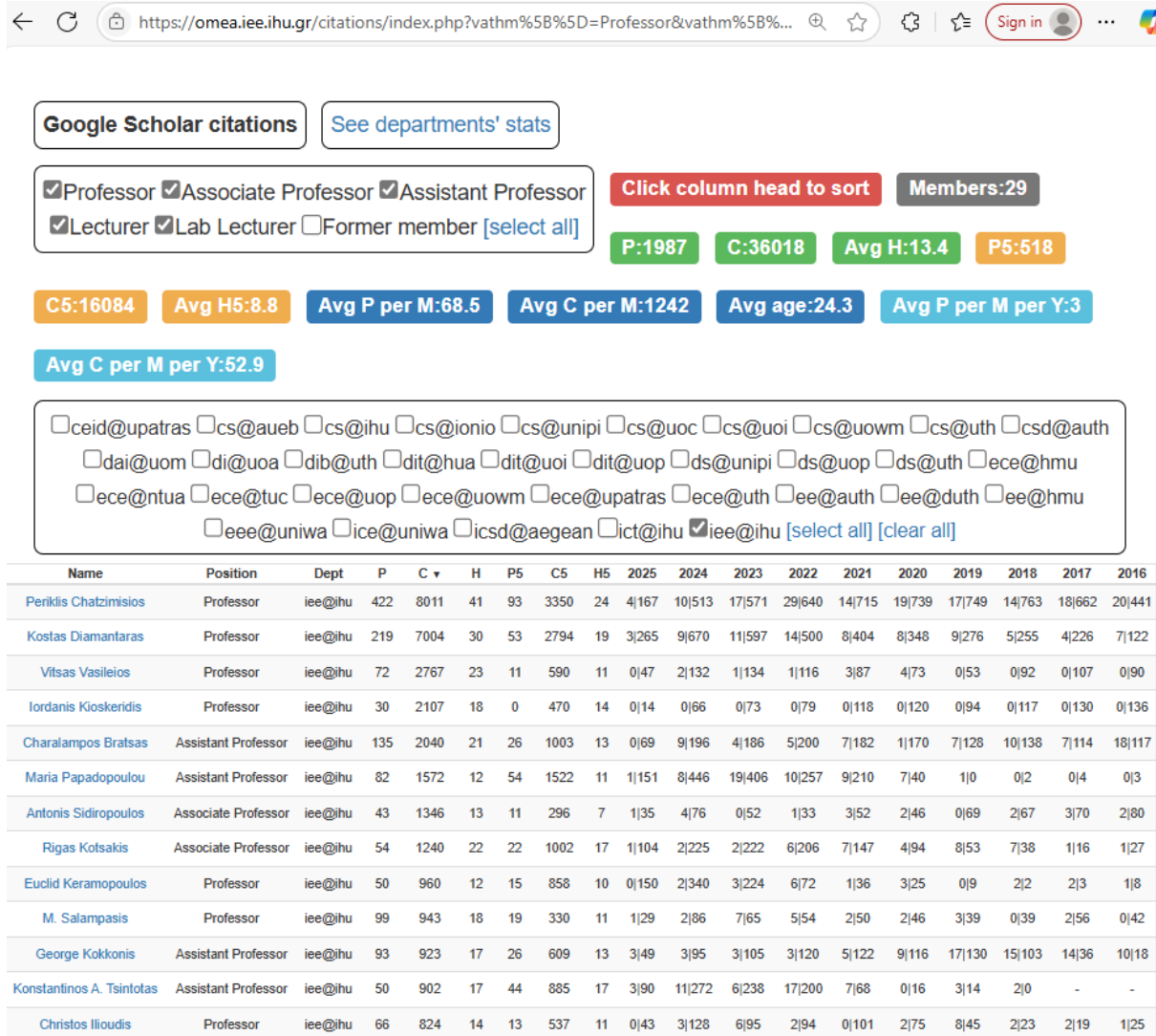
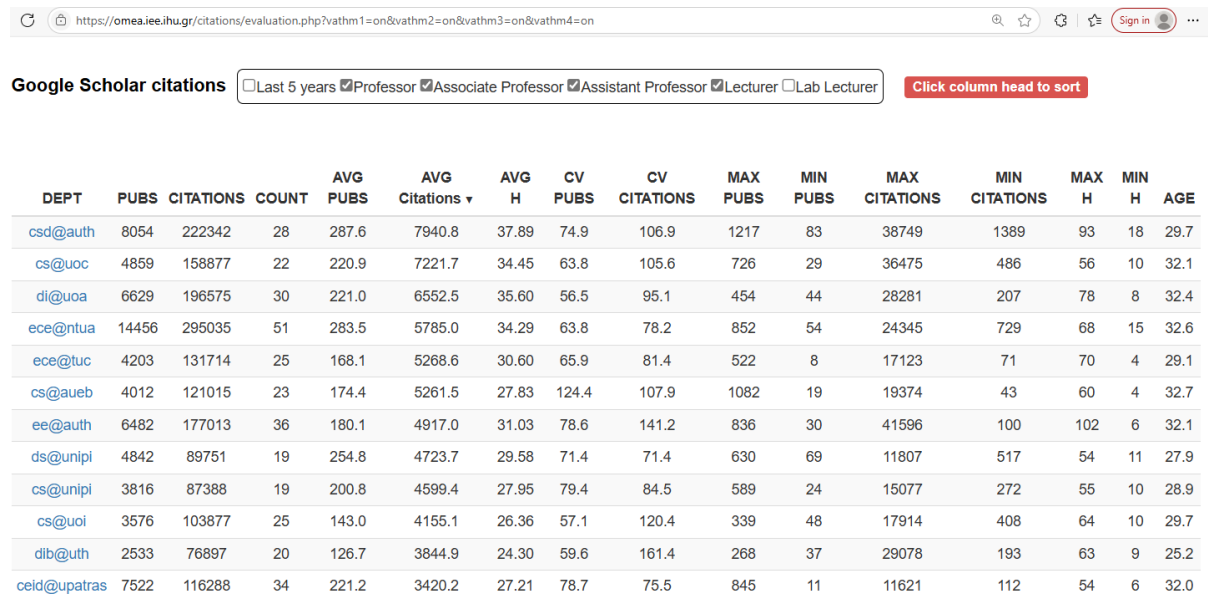


Figure 2.5.2: Faculty Members Page of OMEA Platform

Departments Statistics Page

This page presents a comparative table of bibliometric indicators for all the university departments. The table includes aggregated statistics for each department, such as the total number of publications (PUBS), total citations (CITATIONS), and the number of evaluated faculty members (COUNT). It also displays averages per member, including average publications (AVG PUBS), average citations (AVG Citations), and average h-index (AVG H). Measures of variability are included by the coefficient of variation for publications and citations (CV PUBS, CV CITATIONS), offering insights into distribution consistency within each department. Additional columns report the maximum and minimum values for publications, citations, and h-index among individual

members. The average academic age (AGE) provides context on the career stage of the faculty. The interface allows filtering by academic rank (e.g., Professor, Associate Professor) and offers sorting functionality to facilitate comparisons across departments.



Google Scholar citations ☐ Last 5 years ☒ Professor ☒ Associate Professor ☒ Assistant Professor ☒ Lecturer ☐ Lab Lecturer [Click column head to sort](#)

DEPT	PUBS	CITATIONS	COUNT	AVG PUBS	AVG Citations ▼	AVG H	CV PUBS	CV CITATIONS	MAX PUBS	MIN PUBS	MAX CITATIONS	MIN CITATIONS	MAX H	MIN H	AGE
csd@auth	8054	222342	28	287.6	7940.8	37.89	74.9	106.9	1217	83	38749	1389	93	18	29.7
cs@uoc	4859	158877	22	220.9	7221.7	34.45	63.8	105.6	726	29	36475	486	56	10	32.1
dl@uoa	6629	196575	30	221.0	6552.5	35.60	56.5	95.1	454	44	28281	207	78	8	32.4
ece@ntua	14456	295035	51	283.5	5785.0	34.29	63.8	78.2	852	54	24345	729	68	15	32.6
ece@tuc	4203	131714	25	168.1	5268.6	30.60	65.9	81.4	522	8	17123	71	70	4	29.1
cs@aueb	4012	121015	23	174.4	5261.5	27.83	124.4	107.9	1082	19	19374	43	60	4	32.7
ee@auth	6482	177013	36	180.1	4917.0	31.03	78.6	141.2	836	30	41596	100	102	6	32.1
ds@unipi	4842	89751	19	254.8	4723.7	29.58	71.4	71.4	630	69	11807	517	54	11	27.9
cs@unipi	3816	87388	19	200.8	4599.4	27.95	79.4	84.5	589	24	15077	272	55	10	28.9
cs@uoi	3576	103877	25	143.0	4155.1	26.36	57.1	120.4	339	48	17914	408	64	10	29.7
dlb@uth	2533	76897	20	126.7	3844.9	24.30	59.6	161.4	268	37	29078	193	63	9	25.2
ceid@upatras	7522	116288	34	221.2	3420.2	27.21	78.7	75.5	845	11	11621	112	54	6	32.0

Figure 2.5.3: Departments Statistics Page of OMEA Platform

The analysis of the existing OMEA Citations platform shows that the design of the database is very limited and does not meet the requirements of a modern, scalable and analytical bibliometrics system. Data is stored at the aggregate level, without normalised entities for individual publications, authors or citations. There is no possibility to capture the evolution of indicators over time, as there are no historical data, triggers or change control mechanisms. In addition, no multi-author relationships are maintained, nor metadata per paper, resulting in information that is fragmented and unsuitable for complex analysis or interoperability. The database essentially functions as a static repository of basic numerical values per faculty member, limiting the possibilities for in-depth exploration and evaluation. Similarly, the application interface simply displays these values, without support for interactivity or automated update mechanisms. These weaknesses necessitate a complete redesign of the database, both in terms of structure and functionality, as implemented in the new approach of this paper.

Chapter 3

Technologies and Software Tools

3.1. Python Programming Language

Python is an object-oriented, high-level programming language, created to simplify and clarify programming. Its first release was in 1991 by Guido Van Rossum with version Python 0.9.0.[24]



(a) Python's Logo 1990–2006,
en.m.wikipedia.org



(b) Guido Van Rossum, 2015,
www.facesofopensource.com

Figure 3.1.1: Python and Its Creator

Currently, Python is one of the top programming languages due to its ease of use, learnability, and efficiency. Its user base includes even inexperienced programmers. With its simple syntax and logical structure, it helps both beginners and more experienced users write clean and functional code without wasting time on unnecessary details.

Python does not work like other common programming languages (e.g. C or Java) by compiling the executable file, but reads the code line by line. Each line is executed immediately after being parsed by its interpreter, offering faster executions when the programmer wishes to test the code. In addition, Python does not require you to specify the type of variables beforehand - it adapts according to their use. This speeds up software development and makes the code more flexible.

Below, we see the simplicity that Python provides in printing "Hello, World!" compared to the Java language:

Python

```
print("Hello, World!")
```

Java

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

One of Python's biggest advantages is its extensive library, which provides a wide variety of tools and functions for developing applications. From the simplest tools for text editing, to the most complex libraries for machine learning, image processing, and data analysis, Python has a huge community that is constantly creating new libraries and extensions. The way that Python allows for easy integration of external libraries is also a reason why it is so widely used in the scientific community and technology industry.

In conclusion, Python is a combination of usability and power, making it one of the most popular programming languages on the market today. Its broad user community, its libraries and its capabilities make it ideal for every type of project, from web development to data analysis and data processing.

The official Python documentation provides a full reference for the language's features and libraries [25].

3.1.1. Selenium

Selenium is one of the most widespread and powerful automation libraries offered by Python for interacting with web pages, simulating human behavior inside a browser. It is mainly used for automated web testing, but also for web scraping, as it allows full control of a browser in a programmatic way.

This library is a web browsing automation tool. It allows the developer to write scripts that run in real browsers such as Chrome, Firefox, Edge and others. These scripts can perform any action a user would do, such as:

- Browsing websites and scrolling pages.
- Reading and processing HTML data.
- The text in the entries may be of any length.
- Waiting for dynamic content to load (e.g., from JavaScript).
- Click on buttons or links.
- Fill and submit forms.

This makes Selenium ideal not only for test automation, but also for developing workflows based on web pages.

Selenium does not interact directly with the HTML code of a page, but with the browser itself through a driver. For each browser there is a matching WebDriver (such as ChromeDriver for Chrome), which is activated by Python and controls the browser in real time.

For example, the following simple code launches a web page:

```
from selenium import webdriver

driver = webdriver.Chrome()
driver.get("https://www.google.com")
```

This is used in several areas, most famously in web control. When a website is built, developers want to make sure that everything works properly. Selenium can help with this by running "tests" that mimic a real user. It can also be very useful for extracting data from pages (web scraping), especially when content is dynamically displayed via JavaScript and can't be caught in other, simpler ways.

A key advantage of Selenium is that it works with normal browsers - that is, we see exactly what a user would see. This makes testing much more realistic. In addition, it supports many programming languages, not just Python, and can work with tools for automation or more complex applications. Modern implementations even combine Selenium with Machine Learning methods for more efficient and intelligent scraping, especially in large-scale scenarios [26].

On the other hand, it does have some difficulties. For example, it is relatively slower than other scraping methods (e.g., BeautifulSoup) because it loads entire web pages. Also, it requires a bit more installation, since we also need to download the appropriate WebDriver for our browser.

In summary, Selenium is a pretty powerful tool when we want to automate actions on web pages. Whether for testing, data extraction, or just to make some tasks faster and more relaxing, it can be very useful. Although it is not always the fastest solution, it gives us a lot of flexibility and realism in interacting with a browser's interface.

More details about available drivers and usage examples can be found in the official documentation[27].

3.2. Relational Databases and SQL

3.2.1. Relational Databases

Relational databases are a cornerstone of modern data management, offering a structured and logically sound approach to storing and retrieving information. The relational model, introduced by Edgar F. Codd, organizes data into tables (relations) consisting of rows and columns, facilitating data integrity and reducing redundancy. This model supports powerful querying capabilities through Structured Query Language (SQL), enabling users to perform complex data operations efficiently. For a comprehensive exploration of relational database concepts and their practical applications, refer to the textbook by Elmasri and Navathe [28].

What distinguishes the relational model is its reliance on formal mathematical principles, particularly set theory and first-order logic. This foundation provides not only theoretical rigor but also practical advantages in terms of consistency, scalability, and ease of use. Each relation is defined with a unique primary key to identify individual records, while relationships across tables are expressed using foreign keys. This design enables data to be queried and manipulated through declarative languages like SQL, allowing users to specify what data they want rather than how to retrieve it.

Relational databases enforce data integrity through constraints such as uniqueness, not-null values, and referential rules. These mechanisms help ensure the correctness of data even in complex transactional environments. Furthermore, the tabular format aligns well with human logic, making relational systems widely accessible to developers, analysts, and non-technical users alike.

Despite the emergence of non-relational (NoSQL) models that focus on flexibility and unstructured data, relational databases remain dominant in domains where data relationships, consistency, and reliability are critical, such as finance, healthcare, academic administration, and logistics. Their combination of theoretical soundness and practical efficiency has allowed them to remain relevant for over five decades, continuing to serve as a cornerstone of data-driven decision making.

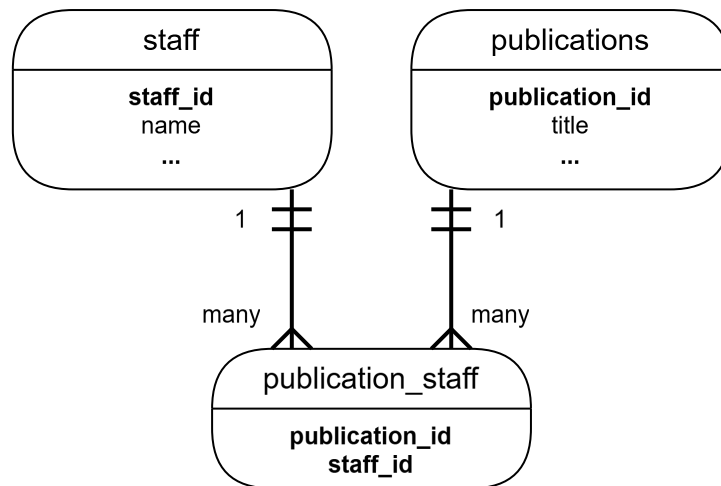


Figure 3.2.1: Example of many-to-many relationship between staff and publications

The above diagram illustrates a classic example of a many-to-many relationship in a relational database. It shows how academic staff and publications are connected through an intermediate table called **publication_staff**. In this schema, each staff member (from the **staff** table) can be linked to multiple publications, and each publication (from the **publications** table) can have multiple contributors. To manage this relationship efficiently and without redundancy, the **publication_staff** table acts as a bridge, storing only pairs of IDs (**staff_id** and **publication_id**). This approach is a hallmark of relational databases, where data is normalized into separate, well-related tables to ensure clarity, consistency, and scalability.

3.2.2. SQL

SQL (Structured Query Language) is a specialized programming language designed to interact with relational databases. It was created by Donald Chamberlin and Raymond Boyce of IBM in 1970, with the name SEQUEL[29].



(a) Donald D. Chamberlin,
computerhistory.org



(b) Raymond F. Boyce,
alchetron.com

Figure 3.2.2: Donald D. Chamberlin and Raymond F. Boyce

It started as a research project and gradually evolved into one of the most important and widely used languages in computer science. Currently, SQL is at the heart of almost every modern

application that handles structured data.

SQL is not a general-purpose programming language like Python or Java; rather, it is a domain-specific language built to communicate with relational database management systems (RDBMS). Its primary role is to allow users to define, access, manipulate and control data stored in structured formats, but tables consist of rows and columns.

It is a multi-faceted language. Its functionality can be broken down into several core areas:

- **Data Definition:** Creating and modifying the structure of tables and databases using commands like CREATE, ALTER, and DROP.
- **Data Manipulation:** Inserting, updating, and deleting data within tables using INSERT, UPDATE, and DELETE.
- **Data Retrieval:** Extracting data using the SELECT statement, often combined with conditions (WHERE), sorting (ORDER BY), grouping (GROUP BY), and joining multiple tables (JOIN).
- **Data Control:** Managing user access and security with commands like GRANT and REVOKE.
- **Transaction Control:** Ensuring data consistency during multi-step operations through commands such as COMMIT and ROLLBACK[30, Chapter 1, p. 10].

The following are some examples of the above categories:

1. Data Definition (DDL) – Creating a table

```
CREATE TABLE students (
  id INT PRIMARY KEY,
  name VARCHAR(100)
);
```

2. Data Manipulation (DML) – Inserting data

```
INSERT INTO students (id, name)
VALUES (1, 'Nikos');
```

3. Data Retrieval (SELECT) – Getting data

```
SELECT name FROM students;
```

4. Data Control (DCL) – Granting access

```
GRANT SELECT ON students TO user1;
```

5. Transaction Control (TCL) – Saving a change

```
COMMIT;
```

SQL is everywhere: in web applications, business intelligence tools, mobile apps, and even on the backend of cloud platforms. It integrates easily with programming languages like Python, Java, and JavaScript, and thanks to interfaces like ODBC, JDBC, and APIs, it acts as a bridge between the database and the application layer.

In recent years, the language has also adapted to newer paradigms. Object-relational databases and data warehouses still rely heavily on it, while cloud services like Amazon RDS, Azure SQL, and Google BigQuery show how it continues to evolve with technology[30, Chapter 23, p. 578].

Technologies like MySQL, PostgreSQL, SQLite, and Microsoft SQL Server are all relational database management systems (RDBMS) that use it as their primary means of interacting with data. While each one has its own features and performance characteristics, they all handle tasks such as creating tables, inserting records, and running queries in a similar way. Among them, MySQL stands out for being open-source, lightweight, and especially popular in web development. It's often the first choice for beginners and startups because it's easy to set up and integrates well with many programming languages and platforms. More details on MySQL's capabilities and usage examples can be found in the official MySQL Reference Manual[31].

Lastly, SQL continues to stand as a core component of modern data management. Its combination of clarity, structure, and efficiency has allowed it to remain relevant across decades of technological evolution. From simple applications to complex enterprise systems, it consistently provides a reliable framework for storing, accessing, and organizing structured information in a consistent and effective manner. For further exploration of SQL syntax and features, you may consult the official SQL documentation provided by ISO standards and supported by major RDBMS platforms[32].

3.2.3. Third Normal Form (3NF)

Third Normal Form (3NF) is a key concept in the design of relational databases, aiming to improve data integrity and eliminate unnecessary redundancy. Originally introduced by Edgar F. Codd and later refined, 3NF builds on the foundational principles of First Normal Form (1NF) and Second Normal Form (2NF). Its purpose is to ensure that the structure of a database supports consistent and efficient data storage without introducing anomalies during insertion, update, or deletion operations.

To understand 3NF, it is essential to first grasp the idea of functional dependencies. A table satisfies the conditions of 3NF if it is already in 2NF and, crucially, if none of its non-prime attributes (attributes that are not part of any candidate key) are transitively dependent on a candidate key. In simpler terms, each non-key attribute must depend directly on the primary key and not through another non-key attribute. This prevents the duplication of data and ensures that each fact is stored in exactly one place.

For example, consider a table that stores employee information along with the department name and the department's manager. If the department name determines the manager, and the department name is not a key in the table, then the manager's name is transitively dependent on the key through the department. This would violate 3NF. To resolve this, the table should be split, placing department-related data in a separate table to avoid redundancy and ensure integrity.

The value of 3NF lies in its ability to produce cleaner, more modular databases that are easier to maintain and less prone to logical inconsistencies. While there are situations where a denormalized structure may be intentionally used for performance optimization, especially in analytical systems, 3NF remains a foundational guideline in transactional database design. A detailed and approachable explanation of this concept is provided in Betty Salzberg's article *Third Normal Form Made Easy*, published in the SIGMOD Record [33].

3.3. PHP

PHP (Hypertext Preprocessor) is a programming language used mainly in the field of web development and web applications. It is a server-side language, PHP code is executed on the server and not on the user's computer. This allows the creation of dynamic web pages that respond to user requests, manage data, and communicate with databases such as MySQL.

It began as a simple set of CGI binaries written in C by Rasmus Lerdorf in 1994 to track visits to his online résumé. What started as “Personal Home Page Tools” evolved rapidly through community collaboration and expansion. By the early 2000s, it had matured into a full-fledged scripting language with robust features, including object-oriented programming and database connectivity. The release of PHP 5 brought a significant leap forward, featuring a new object model and better error handling[34, Introduction, p. 2–3].

PHP’s popularity is due to its simplicity, performance and flexibility. Beginners appreciate its easy syntax and fast learning, while experienced developers appreciate its scalability and extensive built-in functionality. PHP integrates easily with HTML, supports numerous databases (especially MySQL) and is platform independent, meaning it can run seamlessly on Unix, Linux or Windows systems[34, Introduction, p. 4–5]. Some of its notable strengths include:

- Fast performance even on modest servers
- Native support for MySQL and other databases.
- A huge ecosystem of frameworks (like Laravel and Symfony).
- Free and open-source licensing.

Here are some basic examples to demonstrate the syntax and functionality of PHP:

```
<?php
echo "Hello, world!";
?>
```

This simple snippet outputs the phrase “Hello, world!” to the browser. Here’s a slightly more dynamic example using variables and basic arithmetic:

```
<?php
$students = 400;
$staff = 30;
$schoolPopulation = $students + $staff;
echo "The school has a total population of $schoolPopulation people.";
?>
```

This code calculates and displays the total number of people in a school by adding students and staff. Such simple but dynamic behavior shows how PHP can be used to process data and create customized results.

3.3.1. REST API

REST API (Representational State Transfer Application Programming Interface) is a modern architectural style for communicating systems over the Internet using HTTP methods such as GET, POST, PUT and DELETE. What makes REST stand out is its simplicity and commitment to standard web protocols, making it easy to implement and consume across platforms. At its core, REST promotes a static, uniform interface where each resource is accessed by a unique URI, often returning data in lightweight formats such as JSON or XML[35]. These design principles not only enhance scalability and maintainability, but also ensure that REST APIs remain intuitive for developers to understand and extend. For an accessible yet thorough overview of RESTful architecture and best practices, Mariano Rodríguez’s online guide is a valuable resource[36].

PHP is a server-side scripting language designed for the web, and naturally fits into the RESTful paradigm. With just a few lines of code, PHP can handle HTTP requests, parse incoming data,

and generate structured responses, making it a reliable choice for building web services. Despite the growing popularity of newer technologies like NodeJS, PHP still plays a vital role in backend development, especially in small to mid-scale REST API systems[35, Sections 3.2 & 5, p. 3–5]. In fact, real-world implementations show PHP being actively used to develop REST APIs, such as those supporting academic information systems. While performance benchmarks reveal that NodeJS may handle high concurrency more efficiently, PHP remains a practical and accessible tool for API development when simplicity and ease of deployment are priorities.

Here is a minimal example of a PHP script that handles a GET request:

```
<?php
$name = $_GET['name'];
echo "Hello, $name!";
?>
```

To use this script, save it as `greet.php` and access it via a URL like:

`http://yourserver.com/greet.php?name=Anna`

```
Hello, Anna!
```

This example demonstrates how easily PHP can process query parameters and generate dynamic output.

PHP is a powerful and accessible language for building dynamic web applications. It allows seamless integration with HTML, efficient data handling, and strong support for databases like MySQL. Its clear syntax and server-side execution make it ideal for web development tasks. Moreover, PHP works naturally with RESTful APIs, enabling developers to create web services that are simple, lightweight, and effective. Through basic HTTP request handling and structured responses, PHP continues to offer a practical solution for modern, connected web systems. A well-structured and comprehensive manual for PHP developers is maintained by the language's original contributors and community[37].

3.4. Supporting Tools and Environments

In this section, we present the tools and environments that were used or evaluated to support development, data management, remote access, and API control. These tools provide functionality across different layers of development and infrastructure, covering both the programming aspects and the service management components of the project.

3.4.1. HeidiSQL

HeidiSQL is a graphical database management tool that supports MySQL, MariaDB, PostgreSQL and Microsoft SQL Server. It provides capabilities for creating, modifying and viewing tables, executing SQL queries, exporting data, synchronizing databases, and connecting via SSH tunneling [38]. Used to edit tables and other database elements, such as the stored procedures description.

3.4.2. MySQL Workbench

MySQL Workbench is an official tool from Oracle for visually designing and managing MySQL databases. It offers capabilities for drawing ER diagrams, creating SQL scripts, debugging, managing indexes and stored procedures, and performance monitoring [39]. Used to create the database in the early stages.

3.4.3. IntelliJ IDEA

IntelliJ IDEA is an IDE (Integrated Development Environment) from JetBrains that supports many programming languages, including Java, PHP, Python, Kotlin and SQL. It has features such as intelligent code completion, tool refactoring, integrated terminal and debugging [40]. Used for creating and managing the php api

3.4.4. Visual Studio Code

Visual Studio Code is a Microsoft code editor, lightweight but powerful, with support for dozens of languages through extensions. It features a built-in terminal, syntax highlighting, debugging, git integration, live share, and extensive community support [41]. Used for creating and managing the Python scraper.

3.4.5. Linux Server

Linux Server is a server based on a Linux distribution (such as Ubuntu Server), offering stability, security and full command-line control. It supports cron jobs, package installation, web services (Apache/Nginx), and scripting for automation [42]. The whole project is stored and executed on the OMEA Linux server. The supervisor granted permission.

3.4.6. Termius

Termius is an advanced SSH client for Windows, macOS, Linux and mobile devices. It provides multiple connection management, credentials storage, snippets of reusable commands, and the ability to create SSH tunnels through a GUI [43]. Used to interact with the Linux server.

3.4.7. SSH (Secure Shell)

SSH is a secure remote management protocol that uses encryption to allow connection to remote systems. In addition, it supports port forwarding, file transfer (e.g. via scp), and public/private key authentication [44].

3.4.8. SCP (Secure Copy Protocol)

SCP is a command-line tool for securely transferring files between local and remote systems over the SSH protocol. It supports recursive folder transfer, changing SSH ports, and maintaining file permissions [45].

3.4.9. Postman

Postman is a tool for sending and parsing HTTP requests, which is widely used for testing RESTful APIs. It supports testing with different request methods (GET, POST, PUT, DELETE), configuration management, authentication, and response metrics collection [46].

3.4.10. XeLaTeX

XeLaTeX is a modern LaTeX compiler (engine), which natively supports Unicode characters and allows the use of any TrueType or OpenType system font without additional [47] packages. It

significantly improves over traditional PDFLaTeX, particularly for documents requiring multilingual support or complex typography. In addition, full integration with the fontspec package enables the user to select fonts accurately and consistently. XeLaTeX is widely used in academic papers and scientific publications due to the high-quality typographic output it offers. Used for the writing of the thesis paper.

3.4.11. Overleaf

Overleaf is a web-based LaTeX document editing platform that offers real-time collaborative authoring capabilities, cloud storage, and an integrated compilation system with support for XeLaTeX, pdfLaTeX, and LuaLaTeX [48]. It provides a complete development environment (IDE) with syntax hints, PDF output view, file management, and connection to Git or GitHub repositories. It is particularly popular with students and researchers, as it eliminates the need for local LaTeX installation and allows direct access to documents from any device with an internet connection. Used for the writing of the thesis paper.

Chapter 4

System Architecture

This chapter presents the overall architecture of the system, focusing on how the components interact to enable the extraction, storage, and analysis of bibliometric data. It describes the structure and logic of the web scraper, the design of the relational database, and the implementation of supporting mechanisms such as triggers and stored procedures. Additionally, the chapter discusses the RESTful API that exposes the processed data, enabling integration with external systems or frontend applications. The goal is to illustrate the flow of data across all layers and highlight the decisions made to ensure modularity, efficiency, and scalability.

4.1. Google Scholar Scraper

After investigation, it was found that the official and available Google Scholar API was not free and this is a significant barrier to automated bibliometric data collection, especially when dealing with large, massive-scale datasets, such as those of many departments. In situations where systematic documentation of research activity of large numbers of faculty and research staff (faculty) is required, as is the case at the university department level, manual data collection becomes particularly time-consuming and inefficient. To bridge this gap, an automated scraper in Python is developed that uses web scraping techniques via Selenium to navigate and examine public user profiles in Google Scholar.

This scraper is the heart of the system in terms of data collection, as it is responsible for the initial collection of data such as number of publications, citations, publication years, and impact measures such as h-index and i10 index. This information is collected by faculty and then processed and stored in the relational database for further processing and utilization via the RESTful API interface.

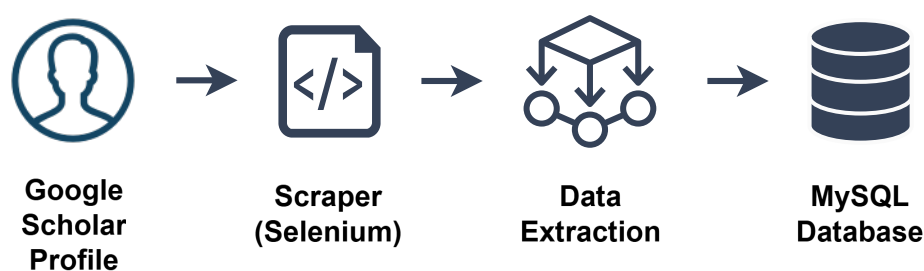


Figure 4.1.1: Scraper basic Flow Diagram

The Python language was chosen for the implementation of the scraper, mainly because of the libraries it offers such as Selenium and Pandas, which proved to be particularly suitable for this type of work. For the interaction with Google Scholar, the Selenium library was chosen, which provides the possibility to programmatically control a browser in real time. Specifically, Firefox WebDriver was utilized in headless mode to achieve full performance and reading of the dynamic elements of the website, without the need for a graphical environment.

The code leverages DOM localization techniques with XPath to accurately identify individual page elements, such as publication titles, years, reference numbers and graphs. In addition, the pandas library is used to store the results, which facilitates the organized representation of data via DataFrames before it is written to the MySQL database. For a secure and encrypted connection to the database, the mysql.connector library is utilized in conjunction with environment variables (dotenv), protecting sensitive information such as login credentials.

Table 4.1.1: Used Libraries in Python Scraper

Library	Short Description
selenium	Web scraping through browser
pandas	Data table management
mysql.connector	Connection with MySQL Database
dotenv	Hidden credentials for Database connection
logging	Error/activity logging

When the scraper logs into a faculty member's public profile in Google Scholar, the data collection process is activated, aiming to maximize bibliometric coverage. In the first stage, the user's general statistics are extracted, such as the total number of citations (total citations), the citations of the last five years, as well as the key impact indices, h-index and i10-index. This data is located in the table on the right of the profile(gsc_rsb_st) and retrieved through the XPath.

Next, the scraper loads all of the user's posts by clicking on the "Show more" button, repeating the process until all the records are displayed. For each post, the following information is extracted:

- Title of the publication (or the value "NULL" if the title cannot be retrieved)
- Year of publication
- Number of citations
- Unique URL of the publication
- Scholar ID of the entry (used for internal tracking)

At the second step, the scraper enters the page of each publication individually and collects further metadata, such as:

- Authors (in plain text format)

- Position of the faculty member in the author list (estimated using fuzzy matching)
- Journal, publisher, and publication date
- Year-by-year citation history (graph data), which is displayed as a bar chart and requires specialized parsing

4.1.1. Synchronisation with the database

Once the data is successfully collected from Google Scholar, the system moves on to syncing that information with the existing MySQL database. This step is important to ensure the bibliometric data remains accurate and consistent, while also avoiding duplicate entries or outdated records. For each publication retrieved from a faculty member's profile, the system checks if it already exists in the publications table by using the paper's unique URL as a reference. If the publication is already in the database, the system compares key fields such as the title, authors, citation count, and year of publication with the existing data to spot any updates or changes.

If any differences found, such as a change in the number of citations or an updated title, the system automatically performs an SQL UPDATE to refresh the relevant records. The relationship between staff members and their publications, which involves multiple connections on both sides, is managed through an intermediate table called `publications_staff`. This table also keeps track of the author order, which is determined using fuzzy matching algorithms. When a new publication is not already in the database, the system adds it by running INSERT operations across all the necessary tables. These include `publications`, `publications_staff`, and `publication_citations_per_year`.

Data comparison and updates are handled through well-structured processes, using pandas DataFrames to quickly and reliably match information between Google Scholar and the local database. A diff-check approach is used to detect any changes efficiently, which helps avoid unnecessary updates or rewriting entire datasets. All actions and modifications are carefully logged in a dedicated log file with a `.log` extension. This ensures that every step is traceable and can be reviewed later if any issues come up.

4.1.2. Calculation of indicators

Beyond just recording the values shown on public Google Scholar profiles, the system also calculates bibliometric indicators locally, using the data it has already gathered and stored. This approach helps ensure better accuracy and consistency when measuring faculty impact, especially in cases where Scholar profiles might be outdated or contain inconsistencies.

Specifically, the system calculates a comprehensive set of indicators:

- h-index and i10-index based on current total citation counts.
- 5-year h-index and 5-year i10-index, calculated from the last five years of publication data.
- `h_index_local / i10_index_local`, derived from the verified citation totals stored in the local database (via `publications` and `publications_staff` tables).
- `last_5_years_h_index_local / last_5_years_i10_index_local`, restricted to publications from the past five years.
- `h_index_from_graph / i10_index_from_graph`, calculated from annual citation data scraped from Google Scholar's citation graphs and stored in `publication_citations_per_year`.

- `last_5_years_h_index_from_graph / last_5_years_i10_index_from_graph`, using only the graphical data from the most recent five-year window.

These calculations are carried out using the pandas library, which makes it easy to work with structured data. Citation records are retrieved, grouped, and organized into DataFrames. From there, Python functions are used to compute key metrics based on the Hirsch algorithm, along with threshold-based logic for determining the i10 index. In addition to the raw numbers, graphical citation data is also used as a secondary check. This helps spot any inconsistencies by allowing comparisons between different views of citation history.

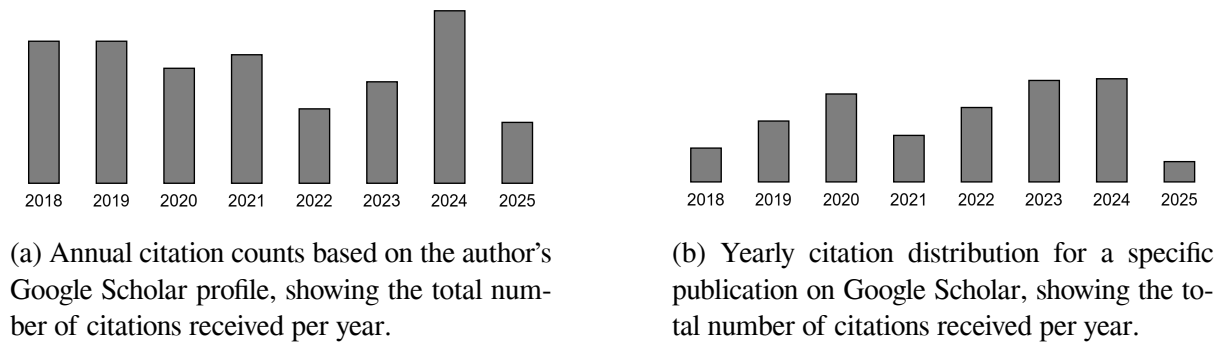


Figure 4.1.2: Example citation graphs from Google Scholar. The exact citation counts for each year are displayed interactively when hovering the mouse over the bars.

What makes this approach valuable is its ability to generate reliable and verifiable metrics that do not depend on Google's platform. Instead, the indicators are based entirely on trusted, internally managed data. This helps avoid issues like inflated metrics from duplicates, incorrect citations, or misattributed publications. By combining both raw citation records and visual citation trends, the system ensures that results are not only accurate but also consistent and suitable for broader use. This makes them especially useful for comparing departments, guiding strategic decisions, and supporting long-term academic evaluations.

4.1.3. Troubleshooting and CAPTCHA

When running the scraper on a large scale, it's expected that occasional issues will come up, such as errors, connection timeouts, or limitations from the Google Scholar platform itself. To handle these situations, the code includes built-in checks, delay functions, and automatic restart mechanisms. These features help keep the process running smoothly and allow it to recover on its own without needing manual oversight.

One of the main challenges when collecting data from Google Scholar is the appearance of CAPTCHA or "unusual traffic" warning pages, which are triggered when the platform detects automated activity. During testing, there were several instances where Google Scholar triggered CAPTCHA because of increased activity. To address these cases, the scraper was integrated with checks that detect URLs like `'sorry/` and messages like `'please show you're not a robot` to pause and restart the process. When one of these triggers is detected, the system automatically pauses for five minutes (`sleep(300)`) and logs a message. This helps prevent the process from continuing while the connection is temporarily restricted.

Listing 4.1: CAPTCHA check on Google Scholar

```
def is_captcha_page(driver):
    try:
        current_url = driver.current_url
        if "/sorry/" in current_url or "/sorry" in current_url:
            return True

        page_source = driver.page_source.lower()
        if "unusual traffic" in page_source or "please show you're not a
robot" in page_source:
            return True

    except Exception as e:
        LOGGER.error(f"Error checking captcha page: {e}")

    return False
```

In addition, the scraper maintains a `failures_in_a_row` counter, which is incremented whenever an error occurs when loading a profile or post. If the failures exceed a predefined threshold (e.g. 5 consecutive failures), the program performs a final restart of WebDriver. This restart helps to restore the stability of the tool, especially in environments where there are memory leaks, timeouts or browser resource blockages.

At the same time, when processing the publication data, it was found that in some cases the publication dates were either missing or contained incomplete or incorrect data (e.g. year < 1901 or inappropriate date formats). To avoid introducing invalid dates into the database, which could cause errors during storage or processing, a validity check was applied, which sets a minimum allowable year (1901) and completely ignores dates that cannot be reliably analysed. This decision was found necessary to ensure the integrity of the chronological data and to avoid false statistics when calculating bibliometric indicators based on time series. Also, during the data collection process, a problem was identified with publication titles containing non-ASCII characters, specifically a character that resembled the letter "m" in mathematical style (e.g. Mathematical Bold Small M in Unicode). The presence of such characters caused errors during storage or further processing. To avoid application crashes, a control mechanism was implemented: if a title cannot be parsed as ASCII, it is replaced with the string "Non-ASCII Title". The case is recorded in the logs for future checking, while the process flow continues normally.

Finally, any operation that might fail, such as sending data to the database, retrieving elements from the page, or loading URLs, is handled using a structured error logging system through Python's logging module. This approach ensures that every step of the process is recorded, making it easier to trace issues, evaluate data quality, and identify opportunities for further improvement.

4.1.4. Scraper Structure and Execution Flow

The `Google_Scholar_Scrape.py` file is built using a modular and layered structure, with separate sections for scraping, data processing, and database operations. This organization makes the code easier to read, maintain, and debug. The entire process runs in a clear sequence, beginning with the database connection and finishing with inserting and syncing the collected data.

The `main()` function serves as the system's entry point. It initiates the connection to the database, launches WebDriver, and starts processing each faculty member included in the staff table. During its execution, it handles retrieving bibliometric data from Google Scholar, calculates and verifies key research indicators, and updates or imports the corresponding statistics and publications into the database. In addition, it includes mechanisms for detecting and recovering from WebDriver

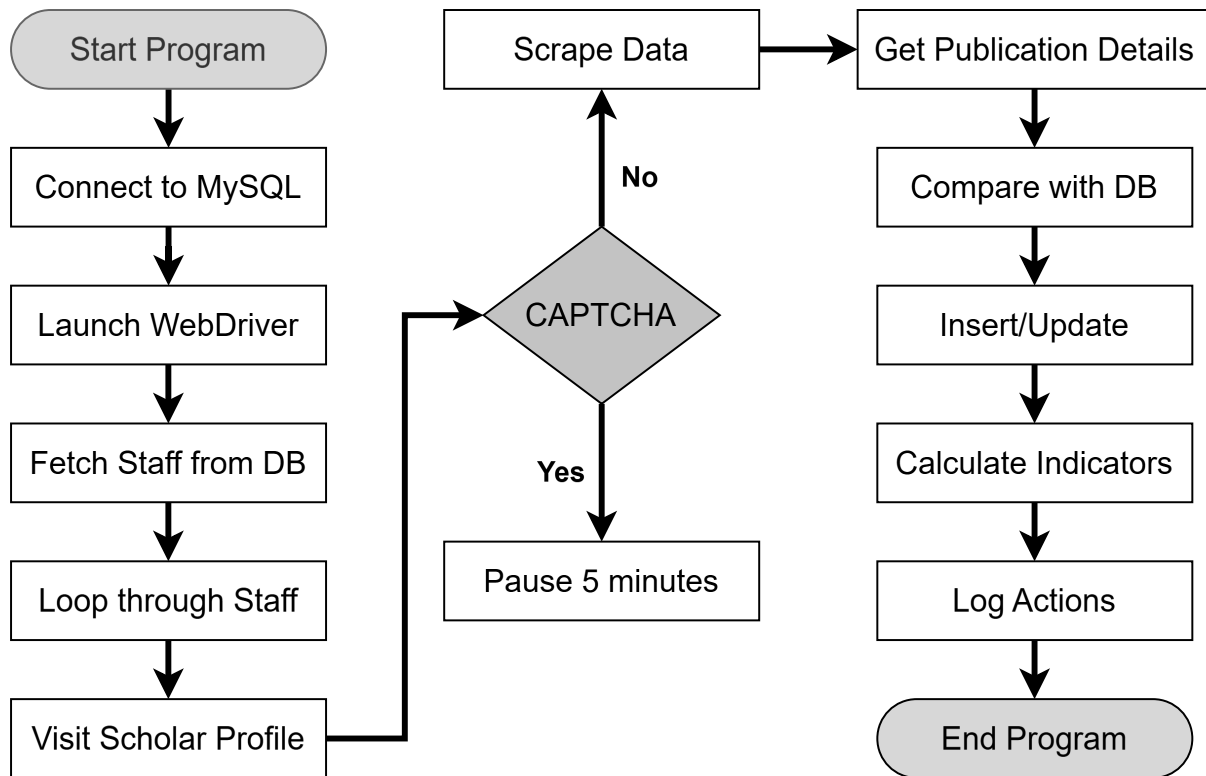


Figure 4.1.3: Main execution flow of the Google Scholar Scraper

failures, ensuring that the scraping process remains stable and reliable over time.

Listing 4.2: Main execution function main()

```

def main():
    LOGGER.info("-- START PROGRAM --")

    global failures_in_a_row, failed_times_to_restart

    connection = create_connection()
    create_driver()

    all_staff = get_all_staff(connection)

    for staff_id, scholar_id, full_name in all_staff:
        LOGGER.info(f"Processing {staff_id=} {full_name=}")

        # Retrieval of data from Scholar
        publications_df, _, staff_graph, staff_stats =
            get_publications_scrape(scholar_id)

        if not publications_df or not staff_stats:
            LOGGER.warning("Skipping due to empty response or error.")
            continue

        # Retrieval of existing data from a database
        stats_db, graph_db, pubs_db = select_all(connection, staff_id)

        # Calculation of indicators
        local_stats = calculate_stats(publications_df)
        citation_df = construct_graph_df(pubs_db, connection)

```

```

graph_stats = calculate_indices_from_graph(citation_df)

# Import or update statistics
sync_statistics(connection, staff_id, staff_stats, local_stats,
graph_stats, stats_db)

# Update years from graphs
sync_staff_citations_per_year(connection, staff_id, staff_graph,
graph_db)

# Import or update publications
sync_publications(connection, staff_id, full_name, publications_df,
pubs_db)

LOGGER.info("END PROGRAM")

```

The `calculate_indices_from_graph()` function is responsible for calculating bibliometric indices based on the annual citations of each publication, as extracted from the citation graphs in Google Scholar. Unlike indices calculated directly from total citation counts, this function utilizes the time-distributed data recorded in the `publication_citations_per_year` table.

First, citations are grouped by `publication_id` and the total number of citations per paper (overall and for the last five years) is calculated. The citation year is dynamically determined based on the current date (`datetime.now().year`), allowing flexible and time-relevant calculation.

The indicators extracted are:

- `h_index_from_graph`: the Hirsch index calculated from the total references of the graphs.
- `i10_index_from_graph`: number of publications with at least 10 citations from the graphs.
- `last_5_years_h_index_from_graph / last_5_years_i10_index_from_graph`: respective indices, limited to the last five years.

The h-index is calculated by first sorting the list of citation counts in descending order, then applying the Hirsch method. In simple terms, this means that the h-th paper must have at least h citations. The i10-index is calculated by counting how many publications have received 10 or more citations.

Using this function brings real value, especially in cases where the total citation count might be inaccurate or outdated. Because the system pulls citation data directly from the Google Scholar interface, it provides an independent source that helps confirm the accuracy of the information. This makes it possible to calculate indices based on each publication's citation history.

Once calculated, these indices are saved in the relevant fields of the `staff_statistics` table. This setup allows for easy comparison between different sources, such as Google Scholar metrics, local calculations, and data pulled from graphs. As a result, the system becomes more reliable and consistent, with built-in cross-checks that reinforces the accuracy of the reported statistics.

Listing 4.3: Calculating indicators from graphs of citations

```

def calculate_indices_from_graph(publication_citations_per_year_df):
    current_year = datetime.now().year

    # Grouping citations by publication
    total_citations = publication_citations_per_year_df.groupby("
publication_id")["citations"].sum()

```

```

# Citations of the last 5 years
recent_df = publication_citations_per_year_df[
    publication_citations_per_year_df["year"] >= current_year - 5
]
citations_last_5 = recent_df.groupby("publication_id")["citations"].sum
()

def h_index(citations):
    citations = sorted(citations, reverse=True)
    return sum(c >= i + 1 for i, c in enumerate(citations))

def i10_index(citations):
    return sum(c >= 10 for c in citations)

return {
    "h_index_from_graph": h_index(total_citations.tolist()),
    "i10_index_from_graph": i10_index(total_citations.tolist()),
    "last_5_years_h_index_from_graph": h_index(citations_last_5.tolist()),
    ,
    "last_5_years_i10_index_from_graph": i10_index(citations_last_5.
        tolist()),
}

```

4.1.5. Scalability and Future Usage

The architecture of the scraper and the entire bibliometric data collection system was designed with scalability in mind, both from a technical perspective and in terms of practical use. The source code is organized into clear, self-contained functions, each with a specific role. This makes it easy to update or expand the system with new features without having to overhaul the entire structure.

In practical terms, the data collection process can be repeated regularly to keep the bibliometric information up to date. The scraper can be set to run automatically using tools like cron, so updates happen regularly without needing manual input. Since the system uses Selenium with headless browsers, it can run smoothly on servers that do not have a graphical interface.

Scalability also applies to the data itself. When new faculty members are added to the database, the system automatically starts collecting information from their Google Scholar profiles. This makes the setup ideal for use on a larger scale, such as across an entire faculty or institution. On top of that, by using annual citation graphs, the system can track research activity over time and produce indicators that reflect trends and changes in scholarly impact.

From a technical standpoint, the scraper is built to handle errors and connection failures by automatically restarting when needed. Each run is fully tracked through a log file, which makes it easy to monitor activity and troubleshoot if something goes wrong. This level of reliability makes the system well-suited for production use, even when working with large volumes of data.

In addition, the system includes several advanced features such as local index calculations, year-based tracking, fuzzy matching for author names, and support for 4-byte UTF characters. These elements show that the scraper is built with flexibility in mind and is capable of handling more complex tasks. It also has the potential to grow further, whether that means importing historical data, adding support for multiple languages, or expanding to other data sources beyond Google Scholar.

4.2. Design and Implementation of Database

The storage and management of bibliometric data is the foundation for the functionality of the system developed in this thesis. The database was designed to organize, categorize and historically capture information related to the faculty members of the Greek Computer Science Departments, their scientific publications and the relevant performance indicators (such as the h-index and the i10- index).

The primary objective of the database was to establish a well- structured, scalable, and historically traceable storage system capable of supporting both statistical analysis queries and the presentation of results through an application programming interface. The implementation was carried out using the MySQL relational database management system, which provides the necessary functionality while also offering a reliable and developer-friendly environment.

The design follows the Entity-Relationship model and includes a number of tables that model the main objects of interest (such as staff, departments, publications), but also auxiliary structures such as many-to-many relationships, stored procedures and triggers.

This section provides a detailed overview of the database design, covering both the logical modeling and implementation aspects. It presents the structure of the tables, the relationships among them, the stored procedures, and the key technical choices made during development.

4.2.1. Design Principles and Methodology

The design of the relational database was guided by core principles of data modeling and normalization to preserve logical integrity and enable future scalability [28]. A central objective was to develop a schema capable of representing the intricate relationships among departments, faculty members, academic roles, and bibliometric indicators, while ensuring that the data remains consistent, coherent, and easy to manage.

For this reason, an Entity-Relationship approach was followed in the design phase. The key entities, such as staff, departments, publications and roles, and the relationships between them were defined. Particular emphasis was placed on modelling many-to-many relationships, such as linking publications to authors, through the use of intermediate tables (publications_staff).

The methodology followed included the following steps:

- Entity Identification: all actual entities in the scope (e.g. faculty members, publications, departments) were mapped to database tables.
- Relationship modelling: the relationships between entities (e.g. member to department, author to publication) were clearly described and normalised.
- Normalization: The base shape was formulated based on the Third Normal Form (3NF)[49] to avoid redundancy and enhance referential integrity.
- Background Recording: For each base table a corresponding action log table (*_records) was created, which in combination with triggers maintains a complete history of insertions, modifications and deletions.
- Stored Procedures: complex questions or frequent aggregations of data (e.g. h-indexes per department) were implemented in the form of stored procedures for reusability and security reasons.

- Forward Engineering: the final schema of the database was created using the MySQL Workbench tool, which allowed for the visual design of the ER diagram and automatic generation of SQL code for the database implementation.

This approach ensured that the system is capable of efficiently handling bibliometric data, supporting complex statistical queries, and maintaining synchronization with external information sources such as Google Scholar.

4.2.2. Database Schema Overview

The structure of the database was carefully designed to meet the specific requirements for storing and organizing bibliometric data related to the Computer Science Departments of Greek universities. Its architecture is centered around core entities, relational mappings, statistical indicators, and mechanisms for tracking changes over time.

The primary functions of the database can be grouped into five main categories:

1. Core Entities

- staff: includes faculty members and their basic data, such as scholar_id and full name, etc.
- departments: information about academic departments, their title, university and website addresses.
- roles: specifies the role of each member (e.g. Professor, Researcher, Research Associate).
- publications: records all publications by title, authors, journal, publisher and number of citations.

2. Relationships

- staff_dept_role: Associates staff with departments and their roles, allowing time-limited assignment of roles.
- publications_staff: intermediate relationship table for defining authors (many-to-many between staff and publications).

3. Statistics and Time Series from the graphs

- staff_statistics: recording the total and five-year values of the bibliometric indicators for each member.
- staff_citations_per_year and publication_citations_per_year: time series of citations for each member or publication, useful for longitudinal analysis and index calculation.

4. Audit Trail Tables

- For each basic table, there is a corresponding table *_records, in which all actions (INSERT, UPDATE, DELETE) are recorded using triggers, allowing full traceability of changes.

5. Stored Procedures

- The database supports a significant number of stored procedures for retrieving statistics, classifying staff, isolating specific indicators, and composing complex queries without the need for multiple JOINS from the application.

The design of the database was based on the needs of this implementation, emphasizing flexibility and scalability. Although in some cases compromises to the structure were required for performance reasons, the final schema fully meets the basic functional requirements.

4.2.3. Description of Entities and Tables

Table staff

Below, the staff table is one of the main entities of the database and contains the main information for each member of the teaching and research staff (faculty). Each entry in the table refers to a person for whose bibliometric data are collected and recorded.

Table 4.2.1: Structure of staff table

Field Name	Data Type	Description
staff_id	INT (PK, AUTO_INCREMENT)	Unique identification number of staff member
scholar_id	VARCHAR(100)	Unique identification number of staff member from Google Scholar profile
first_name	VARCHAR(100)	First name of the staff member
last_name	VARCHAR(100)	Last name of the staff member
created_at	TIMESTAMP (current)	Date of creation of record

The staff table serves as a central element within the database and is defined by a set of key characteristics. The staff_id field functions as the primary key, ensuring that each faculty member is uniquely identified, while the scholar_id provides a reference to their publicly available Google Scholar profile. Any insertion, update, or deletion within the staff table is automatically logged in the staff_records table through the use of triggers. The use of triggers and logging tables enhanced the traceability of changes, but it was found that in environments with increased data load, caution is required to avoid performance problems.

With regard to its connections, the staff table is associated with several other tables, enabling rich interactions across the dataset. Specifically, the staff_dept_role table captures the affiliation of each member with departments and assigned roles. The publications_staff table facilitates the linkage between staff members and their respective publications. Additionally, the staff_statistics table stores key bibliometric indicators, while the staff_citations_per_year table compiles annual citation data. Together, these relationships contribute to a detailed and dynamic representation of each member's scientific output.

Example of staff insertion

```
INSERT INTO staff (scholar_id, first_name, last_name)
VALUES ('abc123def', 'Maria', 'Papadopoulou');
```

Table departments

The departments table contains information about the Computer Science Departments of Greek universities that are monitored by the system. It is a key reference point, as most relationships concerning faculty members and publications are mediated through the departments to which they belong.

Table 4.2.2: Structure of departments table

Field Name	Data Type	Description
department_id	INT (PK, AUTO_INCREMENT)	Unique identification number of department
short_code	VARCHAR(45)	Departmental abbreviation (e.g. iee@ihu)
department_title	VARCHAR(255)	Full department title
university	VARCHAR(255)	Name of the university
city	VARCHAR(45)	City of the University
department_url	VARCHAR(255)	Official URL of the department's website
department_ staff_url	VARCHAR(255)	URL of the website with faculty members
department_ supporting_ staff_url	VARCHAR(255)	URL of the website of the support staff
created_at	TIMESTAMP (current)	Date of creation of record

The departments table provides a central role in the interconnection of the database structure, as it is directly linked to the staff_dept_role table, through which the assignment of members to specific departments and their respective roles is determined. At the same time, it uses the departments_records table to record and monitor changes (audit trail), using triggers that automatically trigger the creation of relevant records, ensuring traceability and transparency of changes.

At the functionality level, the use of multiple URLs is supported, allowing the integration of future automation solutions for data collection (scraping). The existence of a short_code field facilitates the reference of the departments both in user interfaces and in internal processes. Finally, the architectural design provides for the possibility of managing departments belonging to different universities, without restrictions on their number or geographical distribution, giving flexibility and scalability to the model.

Example of department insertion

```
INSERT INTO departments (short_code, department_title, university, city)
VALUES ('iee@ihu', 'Information and Electronic Engineering', 'International
Hellenic University', 'Thessaloniki');
```

Table roles

The roles table lists the roles that faculty members can have within a university department. This table is an auxiliary table, but critical for categorizing staff (e.g., Professor, Associate Professor, Lecturer).

Table 4.2.3: Structure of role table

Field Name	Data Type	Description
role_id	INT (PK, AUTO_INCREMENT)	Unique identification number of role
role_title	VARCHAR(45)	The title of the role (e.g. Professor, Lecturer)
created_at	TIMESTAMP (current)	Date of creation of record

The roles table maintains a direct relationship with the staff_dept_role table, which defines the role held by each member in each individual department. At the same time, any change in roles is recorded in the roles_records table using triggers, ensuring that changes are fully recorded for traceability and audit purposes.

The architecture of the system allows the roles to be independent of the user or the system, which allows them to be enriched without affecting the overall structure. This approach allows a member to maintain different roles in different departments or over different periods of time, offering flexibility in modelling the organisational structure and the evolution of responsibilities.

Example of role insertion

```
INSERT INTO roles (role_title)
VALUES ('Assistant Professor');
```

Table staff_dept_role

The staff_dept_role table acts as an intermediate link that associates each staff member with a specific department and role. The existence of this table allows the support of multiple role assignments per member and per time period.

Table 4.2.4: Structure of staff_dept_role table

Field Name	Data Type	Description
staff_role_id	INT (PK, AUTO_INCREMENT)	Unique identification number of the association
staff_id	INT (FK)	Reference to a member of the staff table
department_id	INT (FK)	Reference to a department of the departments table
role_id	INT (FK)	Reference to a role of the roles table
date_from	TIMESTAMP (current)	Date of start of the association
date_until	TIMESTAMP (nullable)	Association end date (null means active association)

The table staff_dept_role is linked to:

- staff from the staff_id
- departments from the department_id
- roles from the role_id
- staff_dept_role_records and corresponding triggers for full change tracking

The functionality of the system is reflected in the structure of the records, each of which represents a clearly defined assignment of a role to a specific faculty member, within a particular department and for a designated time period. This design enables precise monitoring of academic and administrative roles within evolving institutional settings.

Moreover, the ability to maintain multiple entries for the same individual enhances the system's capacity to capture the professional or academic development of each member over time. This supports the construction of a well-documented and comprehensive career timeline. The model's flexibility also accommodates the assignment of multiple roles concurrently, whether within the same department or across different ones, offering a structured yet adaptable solution for representing complex organizational relationships.

Example of staff_dept_role insertion

```
INSERT INTO staff_dept_role (staff_id, department_id, role_id, date_from)
VALUES (1, 3, 2, CURRENT_TIMESTAMP);
```

Table publications

The publications table is a central element of the bibliometric system, responsible for storing the scientific publications of faculty members. Each entry corresponds to a single publication and includes information such as the title of the work, the list of authors, the name of the journal

or conference where it was published, the number of citations it has received, and the year of publication.

The structure of the table is designed to accommodate different types of publications, including journal articles, conference papers, books, and book chapters. It also supports chronological organization and statistical processing. Key fields include `publication_scholar_id`, which links the entry to its record in Google Scholar, and `publication_url`, which directs to the corresponding online source.

Although the authors field allows for the inclusion of all listed contributors, it is not used to establish detailed relational links between authors and publications. This mapping is handled in a normalized way through the `publications_staff` table, ensuring data consistency and clarity.

Each entry is timestamped using the `created_at` field, and any changes are automatically recorded in the `publications_records` table through the use of triggers. The presence of the `publication_year` field supports time-based queries, while the citations field enables bibliometric analysis of scholarly impact.

Table 4.2.5: Structure of publications table

Field Name	Data Type	Description
<code>publication_id</code>	INT (PK, AUTO_INCREMENT)	Unique identification number of publication
<code>publication_scholar_id</code>	VARCHAR(100)	Unique identification number of publication from Google Scholar
<code>publication_title</code>	VARCHAR(1000)	Title of publication
<code>authors</code>	VARCHAR(5000)	List of authors of publication as a text
<code>publication_date</code>	DATE	Publication date
<code>journal</code>	VARCHAR(1000)	Journal or conference title
<code>publisher</code>	VARCHAR(1000)	Publisher title
<code>citations</code>	INT	Total citations of publication
<code>publication_url</code>	VARCHAR(255)	URL of publication on Google Scholar
<code>publication_year</code>	YEAR	Year of publication
<code>created_at</code>	TIMESTAMP (current)	Date of creation of record

Table `publications_staff`

The `publications_staff` table is an intermediate entity that implements the correlation between faculty members and their scientific publications. Since each publication may have multiple authors and each faculty member may be involved in multiple publications, the relationship is of the many-to-many type[49]. This table normalises this relationship and captures it accurately.

Each entry in the `publications_staff` table represents an author's participation in a specific publication and may optionally include the author's position in the list through the `author_order` field. The inclusion of this detail enables more refined analyses of author contributions, as the order of authorship holds significant meaning in many academic disciplines, including Computer Science.

The creation of this table serves as an essential normalization of the publications table, which only contains a static list of authors in text form. By establishing explicit links between individuals and their publications, the system allows for efficient querying of all works associated with a given faculty member and supports the calculation of individual bibliometric indicators.

Any modification to the `publications_staff` table is automatically recorded in the `publications_staff_records` table, with triggers ensuring that the historical record is consistently maintained. This approach guarantees full traceability of changes and reinforces the reliability and accuracy of the database model.

Table 4.2.6: Structure of `publications_staff` table

Field Name	Data Type	Description
<code>publication_staff_id</code>	INT (PK, AUTO_INCREMENT)	Unique identification number of publication and staff association
<code>staff_id</code>	INT (FK)	Reference to a member of the staff table
<code>publication_id</code>	INT (FK)	Reference to a publication of the publications table
<code>author_order</code>	INT	Author's position in the list of authors (e.g. 1 = main author)
<code>created_at</code>	TIMESTAMP (current)	Date of creation of record

The existence of this table makes it possible to create an accurate and dynamic representation of each member's research activity.

Table `staff_statistics`

The `staff_statistics` table serves as the primary repository for key bibliometric indicators associated with each faculty member, capturing essential data related to their research performance. These indicators are obtained either through direct extraction from Google Scholar using scraping techniques or through internal computations based on the publication data stored within the database. The system is designed to accommodate both overall and five-year values for each metric, supporting a more nuanced evaluation of academic output.

For every faculty member, the table stores values such as total citations, h-index, and i10-index, along with alternative versions of these metrics derived using different methodologies. The distinction between fields ending in `*_local` and `*_from_graph` enables the comparison of results obtained through direct citation graph analysis and those generated by custom algorithms implemented within the system.

The `staff_statistics` table precomputes key metrics such as h-index, citation count, and publication growth, which enables fast, read-only queries via the API without repeated aggregation across

raw data tables. Each record includes a timestamp, which supports future enhancements such as version control and longitudinal performance analysis[50].

To ensure full transparency and historical accuracy, any updates to the data are automatically recorded in the `staff_statistics_records` table through the use of triggers.

Table 4.2.7: Structure of `staff_statistics` table

Field Name	Data Type	Description
<code>staff_statistic_id</code>	INT (PK, AUTO_INCREMENT)	Unique identification number of staff statistic record
<code>staff_id</code>	INT (FK)	Reference to a member of the staff table
<code>total_citations</code>	INT	Total number of citations of staff member
<code>last_5_years_citations</code>	INT	Total number of citations of staff member of the last 5 years
<code>h_index</code>	INT	h-index based on Google Scholar data
<code>last_5_years_h_index</code>	INT	h-index based on Google Scholar data of the last 5 years
<code>i10_index</code>	INT	i10-index based on Google Scholar data (number of papers with >10 references)
<code>last_5_years_i10_index</code>	INT	i10-index based on Google Scholar data of the last 5 years
<code>h_index_local</code>	INT	h-index calculated on the scraper python file using the collected data
<code>last_5_years_h_index_local</code>	INT	h-index calculated on the scraper python file using the collected data of the last 5 years
<code>i10_index_local</code>	INT	i10-index calculated on the scraper python file using the collected data
<code>last_5_years_i10_index_local</code>	INT	i10-index calculated on the scraper python file using the collected data of the last 5 years

Continued on next page

Field Name	Data Type	Description
h_index_from_graph	INT	h-index calculated on the scraper python file using the collected data from Google Scholar's graph
last_5_years_h_index_from_graph	INT	h-index calculated on the scraper python file using the collected data from Google Scholar's graph of the last 5 years
i10_index_from_graph	INT	i10-index calculated on the scraper python file using the collected data from Google Scholar's graph
last_5_years_i10_index_from_graph	INT	i10-index calculated on the scraper python file using the collected data from Google Scholar's graph of the last 5 years
created_at	TIMESTAMP (current)	Date of creation of record

Table staff_citations_per_year

The staff_citations_per_year table stores the annual citation counts for each faculty member, providing a time-based view of their scholarly impact. Organizing this data in a time series format is essential for tracking how the academic influence of each individual evolves over the years.

Each record in the table represents a unique combination of a faculty member and a specific year, documenting the total number of citations received by all of that member's publications during that calendar year. This information serves as a foundation for calculating derived metrics, such as five-year citation totals, and for identifying patterns of growth or decline in research impact.

To ensure transparency and data integrity, each entry includes a timestamp, and any modifications like insertions, updates, or deletions, are automatically logged in the staff_citations_per_year_records table. This design allows for comprehensive traceability and supports consistent long-term analysis.

Table 4.2.8: Structure of staff_citations_per_year table

Field Name	Data Type	Description
staff_citations_per_year_id	INT (PK, AUTO_INCREMENT)	Unique identification number of record
staff_id	INT (FK)	Reference to a member of the staff table

Continued on next page

Field Name	Data Type	Description
year	YEAR	The year for which the number of citations is recorded
citations	INT	The number of citations received by the member's publications for the year
created_at	TIMESTAMP (current)	Date of creation of record

Table publication_citations_per_year

The publication_citations_per_year table captures the number of citations received by each scientific publication on a yearly basis. Structured as a time series, this table focuses on the individual level of each publication, enabling precise monitoring of its citation trajectory over time.

Each entry corresponds to a specific combination of publication and year, indicating how many citations that work received during the respective calendar year. This information is valuable for computing impact metrics, detecting sudden increases in citation activity, and illustrating the broader citation life cycle of a research output.

Maintaining this data allows for detailed annual analysis of each publication's performance and supports more advanced queries, such as identifying the years in which a particular publication had its highest impact.

The existence of the publication_citations_per_year table enhances the ability of the system to track not only the overall impact of a researcher, but also the temporal dynamics of each individual publication. In this way, the model supports more accurate calculations of changing indicators, and provides the basis for deriving detailed impact charts by paper.

As with other dynamic components of the database, any changes to this table, like insertions, updates, or deletions, are automatically logged in the publication_citations_per_year_records table through the use of triggers. This ensures reliable historical tracking and thorough documentation of all modifications.

Table 4.2.9: Structure of publication_citations_per_year table

Field Name	Data Type	Description
publication_citations_per_year_id	INT (PK, AUTO_INCREMENT)	Unique identification number of record
publication_id	INT (FK)	Reference to a publication of the publications table
year	YEAR	The year for which the number of citations is recorded

Continued on next page

Field Name	Data Type	Description
citations	INT	The number of citations received publication for the year
created_at	TIMESTAMP (current)	Date of creation of record

History Tables (*_records)

In the context of the database design, particular emphasis was placed on the traceability of the data, i.e. the recording of all changes to the content of the database over time. For this purpose, a set of parallel history tables, named *_records, was implemented, which record the complete information of each insertion, update or deletion in key tables of the database.

For each master table (such as staff, departments, roles, publications, staff_dept_role, staff_statistics, publications_staff, etc.) there is a corresponding *_records table with a similar structure. These tables additionally include an operation field, which reflects the type of operation (INSERT, UPDATE, DELETE), and an automatic time stamp (created_at). Records in the history tables are automatically created via triggers during any change in the master tables.

The integration of triggers and help log tables allows for maintaining a history of changes, making it easier to review actions and enhance transparency. Although this implementation covers the basic traceability needs, it may require further improvement in large-scale cases.

Example table: staff_records

The staff_records table is the historical record of all actions (insertions, updates, deletions) performed in the staff table. Each record captures the full content of the change, along with the type of action (operation) and the time at which it occurred.

Table 4.2.10: Structure of staff_records table

Field Name	Data Type	Description
record_id	INT (PK, AUTO_INCREMENT)	Unique identification number of record
staff_id	INT	Reference to a member of the staff table
scholar_id	VARCHAR(100)	Unique identification number of staff member from Google Scholar profile
first_name	VARCHAR(100)	First name of the staff member
last_name	VARCHAR(100)	Last name of the staff member
operation	VARCHAR(55)	Action type (INSERT, UPDATE, DELETE)

Continued on next page

Field Name	Data Type	Description
created_at	TIMESTAMP (current)	Date of creation of record

Example record

record_id	staff_id	scholar_id	first_name	last_name	operation	created_at
76	76	42WdccQAAAAJ	Antonis	Sidiropoulos	INSERT	2024-12-19 23:28:04

Similarly, the departments_records, publications_records, roles_records, etc. tables follow the same logic, with the appropriate structure depending on the main table to which they correspond.

4.2.4. Entity–Relationship Diagram with Triggers and History Tables

The below diagram shows the logical structure of the database, which has been extended to include both the history tables (named with the _records suffix) and the triggers responsible for automatically updating them. This representation provides a comprehensive view not only of the static schema but also of the supporting mechanisms for tracking and maintaining historical data changes.

Each primary table in the system, including staff, departments, roles, publications, staff_dept_role, publications_staff, staff_statistics, staff_citations_per_year, and publication_citations_per_year, is directly linked to a corresponding audit table. These audit tables are automatically populated through the use of dedicated triggers, which are executed after every insert, update, or delete operation. As a result, any modification to the original data is consistently recorded, providing a transparent and traceable history of changes over time.

This architectural approach ensures a high level of accountability within the database. It allows the recreation of previous states in case of errors, whether due to user error or software errors, and supports deeper analysis of how data evolves over time. This functionality is particularly valuable for applications that may require version control, historical performance evaluation or retrospective data validation.

Within the diagram, the flow of data from each master entity to the associated control panel is clearly illustrated through trigger links. In addition, the relationships between the main entities are visualised, such as associations between faculty members and their publications, statistical metrics and citation time series. These connections reflect a well-structured and normalized schema capable of supporting complex academic use cases.

Overall, this design reflects a deliberate and methodical effort to create a system that prioritizes security, transparency, and long-term scalability. It goes beyond the basic requirements of a simple bibliometric application and is more closely aligned with the characteristics of an advanced real-time academic research tracking infrastructure.

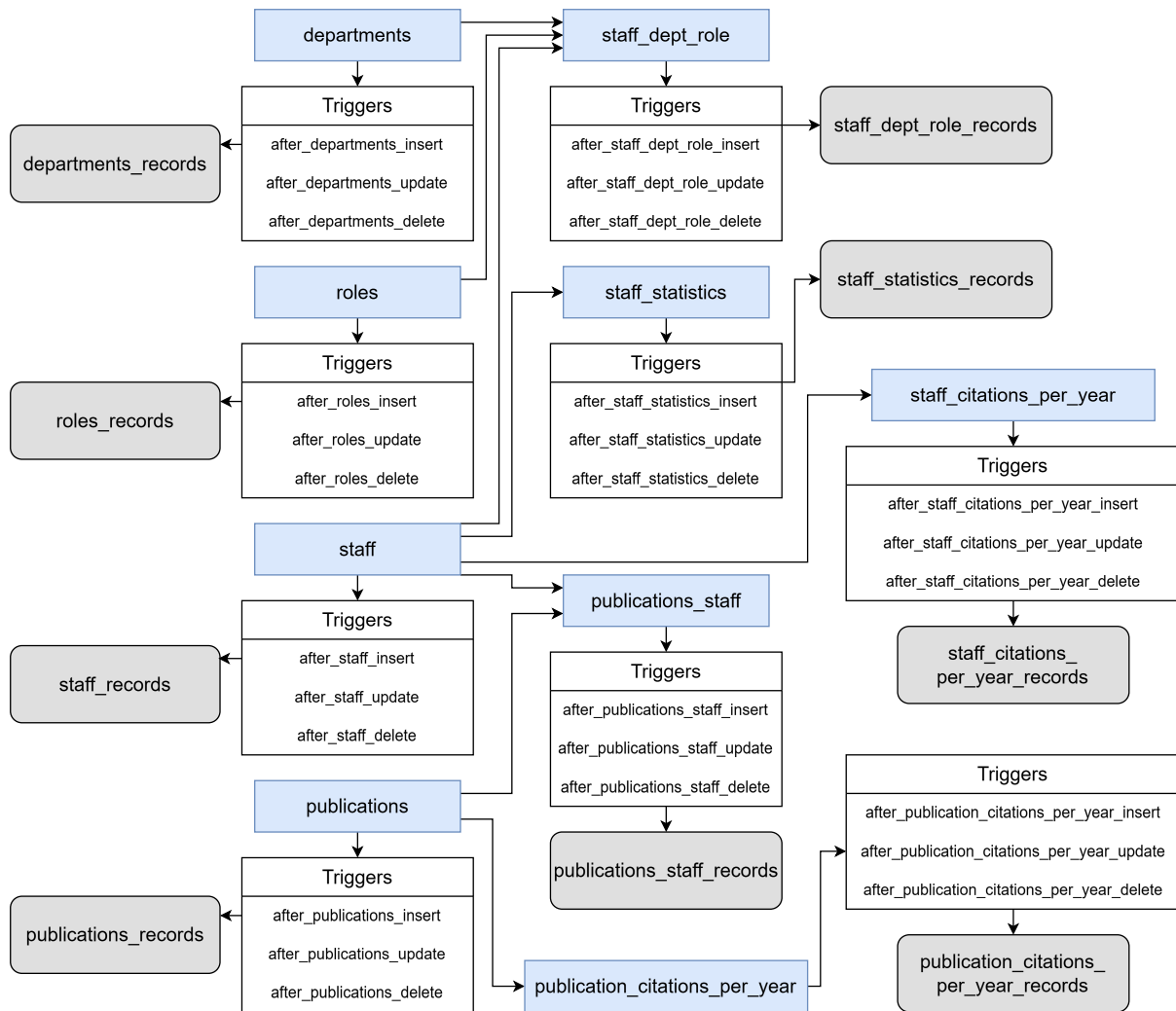


Figure 4.2.1: ER Diagram with Triggers and History Tables

Examples of creating triggers

Example of publication_staff trigger on insert

```
CREATE TRIGGER after_publications_staff_insert
AFTER INSERT ON publications_staff
FOR EACH ROW
BEGIN
    INSERT INTO publications_staff_records (publication_staff_id, staff_id,
    publication_id, author_order, operation)
    VALUES (NEW.publication_staff_id, NEW.staff_id, NEW.publication_id, NEW.
    author_order, 'INSERT');
END$$
```

Example of publication_staff trigger on update

```
CREATE TRIGGER after_publications_staff_update
AFTER UPDATE ON publications_staff
FOR EACH ROW
BEGIN
```

```

INSERT INTO publications_staff_records (publication_staff_id, staff_id,
publication_id, author_order, operation)
VALUES (NEW.publication_staff_id, NEW.staff_id, NEW.publication_id, NEW.
author_order, 'UPDATE');
END$$

```

Example of publication_staff trigger on delete

```

CREATE TRIGGER after_publications_staff_delete
AFTER DELETE ON publications_staff
FOR EACH ROW
BEGIN
    INSERT INTO publications_staff_records (publication_staff_id, staff_id,
publication_id, author_order, operation)
VALUES (OLD.publication_staff_id, OLD.staff_id, OLD.publication_id, OLD.
author_order, 'DELETE');
END$$

```

The remaining triggers are implemented in the same way.

4.2.5. Stored Procedures and Data Access Logic

The design of the system was followed by the implementation of a complete set of stored procedures, aiming to optimize the accessibility, efficiency and reusability of the data retrieval functions from the database. The use of procedures allows to decouple the logic from the application layer [51], while ensuring better security and performance on recurring queries.

Basic Stored Procedures used in the API

The following stored procedures are used directly by the PHP API to display data in the future application:

get_all_roles() and **get_all_departments()**: these procedures return the contents of the roles and departments tables respectively. They are useful for preloading dropdown filters into the application, allowing the user to choose from valid and available values.

get_all_staff_summary(dept_ids, role_ids): the most critical process for the implementation. Returns for each staff member that matches the department and role filters:

- standard identification data (name, scholar ID, department, role)
- bibliometric data from the staff_statistics table
- number of publications (total and for the last five years)
- number of citations (total and for the last five years)
- ‘age’ of the researcher, calculated as the difference between the first and last publication date
- all indices in different versions (scholar, local, from_graph), allowing comparison and flexibility on the front-end. This procedure is fully optimized with subqueries and LEFT JOIN to avoid query fragmentation.

Table 4.2.11: Example data row for the stored procedure `get_all_staff_summary()`

Column Name	Value
staff_id	76
first_name	Antonis
last_name	Sidiropoulos
scholar_id	42WdccQAAAAJ
department_id	15
role_id	2
total_citations	1344
last_5_years_citations	294
h_index	13
last_5_years_h_index	7
h_index_local	13
last_5_years_h_index_local	4
h_index_from_graph	13
last_5_years_h_index_from_graph	7
i10_index	14
last_5_years_i10_index	4
i10_index_local	14
last_5_years_i10_index_local	1
i10_index_from_graph	14
last_5_years_i10_index_from_graph	4
age	22
total_publications	45
total_publications_5y	13

get_all_publications_by_staff(p_staff_id): returns the details of all publications for a staff member (identification number, title, authors, author order, date, journal, publisher, citations, url).

get_publications_per_year_by_staff(p_staff_id): returns a time series with the number of publications for each year for the selected staff member. Uses GROUP BY publication_year with

COUNT and applies only to records that have a valid chronology.

get_citations_per_year_by_staff(p_staff_id): Similar to the above, but instead of number of publications it returns the sum of citations per year for each staff member's work.

get_overall_stats_by_staff_ids(staff_ids): Computes aggregate statistics for a list of staff_ids. Includes:

- total number of staff members
- total and five-yearly publications and citations
- averages for those staff members combined
- reduction of these per year (e.g. avg citations per member per year)
- and average statistics of h-index and i10-index with all variations. It is the basis for high-level reports or group comparisons.

Table 4.2.12: Example data for the stored procedure get_overall_stats_by_staff_ids()

Column Name	Value
count_staff	20
total_pubs	3030
total_citations	61232
total_pubs_5y	835
total_citations_5y	9095
avg_pubs_per_m	151.5
avg_cits_per_m	3061.6
avg_age	22.45
avg_pubs_per_m_per_y	6.0
avg_cits_per_m_per_y	115.0
avg_h_index	21.4
avg_h_index_local	21.5
avg_h_index_from_graph	21.5
avg_h_index_5y	14.2
avg_h_index_local_5y	8.1

Continued on next page

Column Name	Value
avg_h_index_from_graph_5y	14.4
avg_i10_index	51.1
avg_i10_index_local	51.2
avg_i10_index_from_graph	53.3
avg_i10_index_5y	28.2
avg_i10_index_local_5y	9.5
avg_i10_index_from_graph_5y	29.6

get_department_stats_by_depts_and_roles(dept_ids, role_ids): Generates aggregate statistics by department for members who match the selected role filters. Includes:

- total staff members
- number of publications and citations
- index averages (h-index, i10-index) with variations
- average and maximum number of publications/citations
- coefficient of variation (CV) on key variables (e.g. citations), useful for internal homogeneity analysis. The procedure uses aggregation with GROUP BY department and subtables to compute member-level statistics and summarize them at the department level.

Table 4.2.13: Example data row for the stored procedure get_department_stats_by_depts_and_roles()

Column Name	Value
department_id	15
short_code	iee@ihu
staff_count	25
total_pubs	1784
total_citations	35250
avg_h_index	21.6
avg_h_index_5y	13.5

Continued on next page

Column Name	Value
avg_h_index_local	21.6
avg_h_index_local_5y	17.0
avg_h_index_from_graph	21.5
avg_h_index_from_graph_5y	13.5
avg_i10_index	45.5
avg_i10_index_5y	23.7
avg_i10_index_local	45.5
avg_i10_index_local_5y	6.5
avg_i10_index_from_graph	46.0
avg_i10_index_from_graph_5y	24.0
avg_pubs	143.0
avg_pubs_5y	39.3
avg_citations	3194.7
avg_citations_5y	429.5
avg_age	25.09
cv_pubs	82.8
cv_cits	98.9
max_pubs	353
min_pubs	10
max_cits	8022
min_cits	57
max_h_index	41
min_h_index	5
max_i10_index	107
min_i10_index	1

The above example is calculated for the department with id 15 and roles: Professor (id: 1), Associate Professor (id: 2), Assistant Professor (id: 3).

Design Evolution and Optimization through Aggregation Procedures

During the process of designing and implementing the backend, various design models were tested. Initially, a set of small stored procedures was implemented, each responsible for a specific index or statistic (such as `get_staff_h_index_all`, `get_count_publications`, `get_age`, etc.). Although functionally efficient in isolated queries, in practice, they proved to be unviable for mass use. Simultaneously calling many such procedures for tens or hundreds of staff members caused delays and overhead on the database server, significantly reducing the application's response speed.

To solve this problem, the logic of the backend was redesigned with the purpose of batch and mass production of results. Instead of individual calls per index, more complex stored procedures were created, such as `get_all_staff_summary` and `get_overall_stats_by_staff_ids`, which collectively compute all necessary indexes for multiple faculty members in a single query. The backend and frontend can then filter or exclude data at will, without requiring additional SQL execution. The result is increased performance, reduced latency, and easier management of data size.

Table 4.2.14: Non-basic stored procedures available for extended or future use.

Procedure Name	Short Description
<code>get_staff_h_index_scholar(IN staff_id)</code>	Returns the scholar-based h-index and 5-year h-index for a given staff member.
<code>get_staff_h_index_local(IN staff_id)</code>	Returns the locally computed h-index and 5-year h-index.
<code>get_staff_h_index_from_graph(IN staff_id)</code>	Returns h-index metrics based on citation graph analysis.
<code>get_staff_i10_index_scholar(IN staff_id)</code>	Returns the i10-index and 5-year value from Google Scholar data.
<code>get_staff_i10_index_local(IN staff_id)</code>	Returns i10-index metrics computed from local publication data.
<code>get_staff_i10_index_from_graph (IN staff_id)</code>	Returns i10-index based on citation graph extraction.
<code>get_count_publications(IN staff_id)</code>	Counts the total number of publications for a staff member.
<code>get_count_citations(IN staff_id)</code>	Returns the sum of all citations for a staff member.
<code>get_count_publications_5y(IN staff_id)</code>	Counts publications over the last 5 years.
<code>get_count_citations_5y(IN staff_id)</code>	Computes total citations from the last 5 years.

Continued on next page

Procedure Name	Short Description
get_age(IN staff_id)	Calculates academic age based on earliest and latest publication year.
get_staff_citations_per_year(IN staff_id)	Returns yearly citation data for the selected staff member.
get_publication_citations_per_year (IN publication_id)	Returns citations per year for a specific publication.
get_publications(IN staff_id)	Returns the list of publications for a given staff member.
get_publications_staff_by_dept(IN dept_id)	Returns publications filtered by department.
get_department_stats_by_roles (IN role_ids)	Computes department-level statistics filtered by academic roles.
get_all_basic_stats_for_a_staff (IN staff_id)	Consolidates all key metrics into a single view for one staff member.

These procedures can be used in the future for extensions of the application (e.g., dynamic filters, export, comparison between members). If not functionally required, they can be removed from the base to reduce footprint and complexity.

In any case, the use of stored procedures provides the necessary structure to support a robust backend, scalable and adaptable to new queries and analyses.

4.2.6. Limitations, Extensibility and Future Enhancements

Despite the comprehensive functionality of the system and its ability to support real-time bibliometric analysis, there are individual limitations identified during development, as well as areas that need further expansion in the future.

Limitations

One of the main limitations is the reliance on Google Scholar as the exclusive data source. Although it offers great coverage and is freely accessible, the formatting of its profiles is not stable and is not formally supported via API. This results in inconsistencies, accessibility limitations (rate limits), and differences in data structure, which require constant maintenance of the extraction software.

At the performance level, the original model relied on multiple small stored procedures, each responsible for one index. Although functionally sound, they proved to be inefficient at scale, as their massive use increased the application response time. This problem was addressed by the introduction of massive aggregation procedures. However, running heavy queries on large

volumes of data can still be difficult in systems with limited resources.

In addition, there are multiple index records (Scholar, Local, From Graph), which offers flexibility in analysis but increases structural complexity and creates the need for consistent updating and validity checking of results. There is currently no automated reconciliation mechanism between these versions.

Finally, the system is mainly oriented towards data reading and analysis. Functionality for interactive data correction or editing (e.g. correcting wrong titles or manually unlinking duplicate publications) is not yet supported.

Extensibility of the System

The system architecture was designed with scalability in mind. All key subsystems such as entities, statistics and time series are normalised and have a clear functional distinction. This allows:

- Easy integration of new data sources (such as Scopus, Web of Science) by simply linking identifiers in the staff and publications fields.
- Expanding the scheme to support new bibliometric or alternative indicators (e.g. altmetrics, author collaborations, journal quality indicators).
- Configuration for use by departments outside of IT or outside Greece, through customization of department metadata and role mapping.

There is also provision for the integration of authentication mechanisms and user roles to support secure administration, change logging, and access to personalized functions.

Future Enhancements

Various improvements can be made to enhance the functionality and usability of the system. A key area concerns the development of an interactive editing interface that allows authorised users to display, correct, validate, or highlight inconsistencies in the data. This functionality is intended to improve data quality through direct human oversight. Another proposed improvement is the incorporation of advanced visualizations such as referral graphs, collaboration networks, and dynamic trend charts categorized by department and academic role. These visual tools are expected to enrich the interpretability of the data and support deeper analysis. In addition, a reporting mechanism is being considered to enable automated PDF or Excel reports containing key statistics for each department or individual user. Finally, there is the possibility of integrating natural language processing (NLP)[52] technologies to support the classification of publications by research area, further extending the analytical capabilities of the system.

4.3. Design and Operation of the API

The API (Application Programming Interface) function is a core building block of the system, which allows transparent and organized access to the bibliometric data stored in the relational database. In the context of this project, the API enables the search, aggregation and retrieval of data derived from Google Scholar, utilizing stored procedures (stored procedures) and standardized SQL queries.

The primary purpose of the API is to offer secure and efficient access to the system's data through HTTP requests. By centralizing and standardizing the data extraction logic, it enables

consistent reuse across various platforms and consumption methods, such as standalone applications, scripts, or third-party services. Additionally, the API lays the groundwork for the future development of frontend applications, which will be able to leverage this interface for dynamic data presentation, analysis, or user interaction, without needing to interact with the underlying database structure directly.

It is designed according to RESTful principles, with a clear and understandable structure of endpoints (such as /staff, /departments, /roles) and support for configuration via URL parameters. Data is returned in JSON format, allowing easy use in web applications or JavaScript libraries in the future.

It is important to note that at this phase there is no frontend or any direct user interface implemented. The API functionality is intended as a prepared backend on which a visualization or interactive data analysis system can be based in the future.

The implementation is entirely based on PHP files, which perform calls to stored procedures in MySQL and return the results to the client. The use of stored procedures ensures performance, reusability and independence from application software, making the API flexible for future re-design or integration into a new environment.

4.3.1. Architecture and Structure of the API

The API is implemented in a PHP environment, following a simple but scalable architecture, with separation of functionality into thematic modules and clear naming rules for endpoints. Each PHP file on the server maps directly to a RESTful endpoint and handles a specific type of information or function.

The architecture has the following main parts:

- **Endpoints (modules):** each main object (such as staff, departments, roles) corresponds to its own PHP file, which controls the input, communicates with the database and returns a JSON response.
E.g., departments.php, roles.php, staff.php
- **Router file (index.php):** The index.php file works as a basic router for the API. All HTTP requests that are made to the server go through this file first, due to the redirection defined in .htaccess. index.php parses the path (PATH_INFO) and forwards the request to the appropriate endpoint, e.g., staff.php, departments.php, etc. This way, a clean URL structure (REST-style) is achieved without the need for a framework.
- **Support functions (functions.php):** includes common functions such as decoding tokens of URL IDs. Used by all endpoints files to avoid repetitive code.
- **Database connection configuration (db.php):** host file that opens a connection to the MySQL server by reading parameters from the .env configuration file. It is properly protected using environment variables and supports read-only queries.
- **Configuration and security files (.env, .htaccess):**
 - .env contains the sensitive environment variables, such as base credentials and compression settings.
 - .htaccess restricts direct access to internal files and ensures proper recognition of requests by the web server (Apache).

The basic flow of a request is the following:

1. The client sends an HTTP GET request to a specific endpoint, e.g. `/staff/stats?departments=15&roles=2,3,4`
2. The `index.php` file handles the recognition of the path and forwards the request to the corresponding module.
3. The associated file calls functions from `functions.php` to check parameters and initialize queries, if needed.
4. The connection to the database is made through `db.php`.
5. The related stored procedure with the parameters is called.
6. The database results are formatted as JSON and returned to the client.

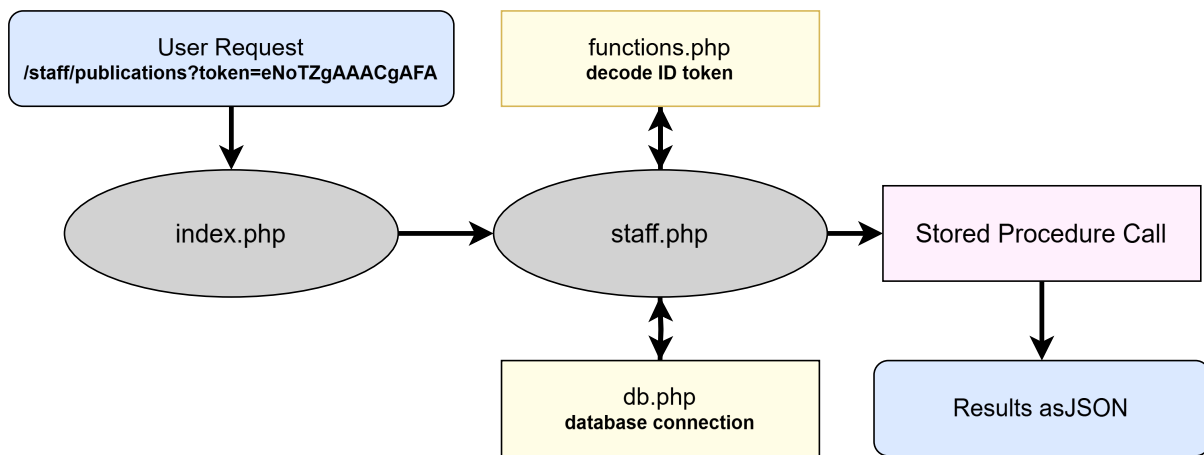


Figure 4.3.1: Request Flow for the Publications of a Staff Member

This architecture is simple, easy to read and offers potential for future expansion. In addition, the use of stored procedures isolates business logic from code, facilitating maintenance and security.

4.3.2. Types of Requests (Endpoints)

The API provides a set of endpoints based on a REST architecture, which allow access to system data. All endpoints accept HTTP GET requests and return results in JSON format.

The main categories of functionality include:

Static Data

- `/roles`: Returns all the IDs and the titles of the roles (e.g., 1, Assistant Professor)
- `/departments/all`: Returns all departments participating in the system, with ID, title and short code.

Staff Data

- `/staff/stats`: Returns bibliometric statistics for members belonging to departments and roles given as departments and roles parameters. E.g., `/staff/stats?departments=14&roles=2,5`
- `/staff/stats/token`: Same operation as above, but parameters are given compressed using GZIP and Base64. E.g., `/staff/stats/token?departments=eNpjPgAAAMgAxA&roles=eNpjaAiQYGdiAAAFEADy`
- `/staff/publications/per-year?id=76`: Returns the member's number of publications per year.
- `/staff/citations/per-year?id=76`: Returns the number of citations per year for that member.
- `/staff/publications?id=76`: Returns a list of all publications and their details for a member.
- `/staff/overall?ids=1,2,3,...`: Provides summary statistics for group members based on the IDs provided. Includes number of publications, citations, h-index, i10-index and indexes per member/year.
- `/staff/overall?token=safewdfdfD`: Same operation as above, but parameters are given compressed using GZIP + Base64

Listing 4.4: Example code for handling overall statistics by staff IDs

```
if ($segments[1] === 'overall') {
    $staff_ids = isset($_GET['ids'])
        ? array_map('intval', explode(',', $_GET['ids']))
        : decode_token($_GET['token'] ?? '');

    if (empty($staff_ids)) {
        http_response_code(400);
        echo json_encode(["error" => "Missing or invalid staff IDs"]);
        exit;
    }

    $staffStr = implode(',', $staff_ids);

    try {
        $stmt = $pdo->prepare("CALL get_overall_stats_by_staff_ids(:staff_ids)");
        $stmt->bindParam(':staff_ids', $staffStr);
        $stmt->execute();

        $result = $stmt->fetch(PDO::FETCH_ASSOC);
        echo json_encode($result, JSON_UNESCAPED_UNICODE);
    } catch (PDOException $e) {
        http_response_code(500);
        echo json_encode(["error" => "Database error"]);
    }
    exit;
}
```


Department Data

- `/departments/statistics/ids?departments=14,15&roles=4,5,6`: Returns aggregate statistics by department for the given roles and IDs.
- `/departments/statistics/token?departments=<fdsgFDf>&roles=kdaKHKSaja`: Alternative format for sending parameters via GZIP token.

Each endpoint can be combined with the others to allow flexible access to data, both for visualization and for extraction and analysis. The use of GZIP tokens reduces the size of URLs and allows secure and efficient configuration management in frontend applications, which can be integrated in the future.

Table 4.3.1: Mapping of API Endpoints to MySQL Stored Procedures

Endpoint	Stored Procedure
<code>/staff/citations/per-year?token=...</code>	<code>get_citations_per_year_by_staff()</code>
<code>/staff/publications/per-year?token=...</code>	<code>get_publications_per_year_by_staff()</code>
<code>/staff/stats/token?departments=...&roles=...</code>	<code>get_all_staff_summary()</code>
<code>/roles</code>	<code>get_all_roles()</code>
<code>/departments/all</code>	<code>get_all_departments()</code>
<code>/departments/statistics/token?departments=...&roles=...</code>	<code>get_department_stats_by_depts_and_roles()</code>
<code>/staff/overall?token=...</code>	<code>get_overall_stats_by_staff_ids()</code>
<code>/staff/publications? token=...</code>	<code>get_all_publications_by_staff()</code>

4.3.3. Token-based Access and GZIP Encoding

To support efficient and secure parameter transfer via URL, the API adopts a parameter encoding mechanism using GZIP compression and Base64 encoding. This practice is particularly useful in cases where parameter lists (e.g., multiple IDs for departments or roles) are large and it is impractical to include them directly in the URL.

Use of Token

In place of the normal configuration, the frontend can send a token to the endpoint. The token contains the same information that would otherwise be sent as e.g. `departments=14,15&roles=2,4,6`.

Apply to the Backend

On the backend, the decoding mechanism is implemented in `functions.php` and follows the following steps:

```
function decompress_token($encoded) {
    $compressed = base64_decode($encoded);
    $json = gzdecode($compressed);
    return json_decode($json, true);
}
```

- Base64 Decoding: Recovers the original binary from the encoded format.
- GZIP Decompression: Decompresses the data in the original JSON string.
- JSON Decoding: Returns the data as a PHP table for use in SQL queries.

Benefits

Using compressed tokens to encode parameters in URL requests offers multiple advantages. First and foremost, a significant reduction in the overall length of URLs is achieved, which makes it easier to manage and send large lists of data, such as user IDs or filters. At the same time, URLs are made cleaner and less error-prone by the end user, as direct parameter processing is reduced. Finally, greater flexibility is provided to the client, allowing frontend applications or external services to store or share entire sets of parameters in the form of a compact and portable token.

The use of token-based parameters is not mandatory; all endpoints accept the "normal" parameter format (e.g. `?departments=...&roles=...`) for convenience and simplicity where needed.

4.4. Extensibility

Although this API operates independently of a graphical frontend, it has been designed from the outset with a view to its future interface with a fully functional interface. All endpoints return normalized data in JSON format, allowing direct utilization by frontend applications of any technology (React, Vue, Angular, etc.) or even external systems.

The implementation of token-based parameters, structured routing architecture and fixed response format provides a strong basis for extensions such as:

- dynamic dashboards and statistics tables,
- interactive filters in real time,
- Data export (e.g. to CSV, Excel),
- creating reports and comparisons between members.

Moreover, the backend logic is completely independent of the presentation layer, which allows the reuse of the same infrastructure by multiple clients (e.g. internal management tools, shared web interface, mobile application).

The connection to a frontend can be implemented at any time without any change to the base or server, the infrastructure is already ready to support it.

Chapter 5

Results

This chapter presents indicative results obtained from the process of data collection and processing through scraping data from Google Scholar. The data relates to faculty members (Teaching Research Staff), their publications, and their respective bibliometric metrics such as citations, h-index and i10-index.

All data were stored in a relational database according to the schema discussed in the previous chapters. They were recorded in order to provide a complete and reliable representation of the research profile of the members, both in temporal and quantitative dimensions.

For a better understanding and evaluation of the collected data, some results are accompanied by graphs, which were created using Python. These graphs are not part of a functional application, but have been created solely to present and explain the results within this thesis.

5.1. Collected Data and Indicative Results

5.1.1. Tables

Departments

The data of the departments were collected in greek language.

Table 5.1.1: Example Department Records

Field	Department 1	Department 2
department_id	1	2
short_code	csd@uoc	dai@uom
department_title	Επιστήμης Υπολογιστών	Εφαρμοσμένης Πληροφορικής
university	Πανεπιστήμιο Κρήτης	Πανεπιστήμιο Μακεδονίας
city	Ηράκλειο	Θεσσαλονίκη

Continued on next page

Field	Department 1	Department 2
department_url	http://www.csd.uoc.gr/	https://www.uom.gr/dai
department_staff_url	https://www.csd.uoc.gr/index.jsp?content=aca...	https://www.uom.gr/en/dai/academic-staff
department_supporting_staff_url	https://www.csd.uoc.gr/index.jsp?content=edi...	https://www.uom.gr/en/dai/special-teaching-te..

The following are two indicative entries from the departments table. Similar information has been recorded for **all 35** departments included in the database. Below is the full list of the departments:

Table 5.1.2: Departments and Universities

Department	University
Επιστήμης Υπολογιστών	Πανεπιστήμιο Κρήτης
Εφαρμοσμένης Πληροφορικής	Πανεπιστήμιο Μακεδονίας
Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών	Πανεπιστήμιο Πελοποννήσου
Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών	Ελληνικό Μεσογειακό Πανεπιστήμιο
Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών	Πανεπιστήμιο Δυτικής Μακεδονίας
Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών	Πολυτεχνείο Κρήτης
Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών	Πανεπιστήμιο Θεσσαλίας
Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών	Εθνικό Μετσόβιο Πολυτεχνείο
Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών	Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών	Δημοκρίτειο Πανεπιστήμιο Θράκης
Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών	Πανεπιστήμιο Πατρών

Continued on next page

Department	University
Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής	Πανεπιστήμιο Ιωαννίνων
Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής	Πανεπιστήμιο Πατρών
Μηχανικών Πληροφορικής και Επικοινωνιακών Συστημάτων	Πανεπιστήμιο Αιγαίου
Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων	Διεθνές Πανεπιστήμιο Ελλάδας
Μηχανικών Πληροφορικής και Υπολογιστών	Πανεπιστήμιο Δυτικής Αττικής
Μηχανικών Πληροφορικής, Υπολογιστών και Τηλεπικοινωνιών	Διεθνές Πανεπιστήμιο Ελλάδας
Πληροφορικής	Πανεπιστήμιο Πειραιά
Πληροφορικής	Ελληνικό Ανοικτό Πανεπιστήμιο
Πληροφορικής	Ιόνιο Πανεπιστήμιο
Πληροφορικής	Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Πληροφορικής	Οικονομικό Πανεπιστήμιο Αθηνών
Πληροφορικής	Πανεπιστήμιο Δυτικής Μακεδονίας
Πληροφορικής	Δημοκρίτειο Πανεπιστήμιο Θράκης
Πληροφορικής	ΕΚΠΑ
Πληροφορικής	Χαροκόπειο Πανεπιστήμιο
Πληροφορικής και Τηλεματικής	Χαροκόπειο Πανεπιστήμιο
Πληροφορικής και Τηλεπικοινωνιών	ΕΚΠΑ
Πληροφορικής και Τηλεπικοινωνιών	Πανεπιστήμιο Ιωαννίνων
Πληροφορικής και Τηλεπικοινωνιών	Πανεπιστήμιο Πελοποννήσου
Πληροφορικής με εφαρμογές στη Βιοϊατρική	Πανεπιστήμιο Στερεάς Ελλάδας
Ψηφιακών Συστημάτων	Πανεπιστήμιο Πειραιά
Ψηφιακών Συστημάτων	Πανεπιστήμιο Θεσσαλίας
Ψηφιακών Μέσων	Ιόνιο Πανεπιστήμιο
Ψηφιακών Μέσων και Επικοινωνίας	Ιόνιο Πανεπιστήμιο
Ψηφιακών Συστημάτων	Πανεπιστήμιο Πελοποννήσου
Πληροφορικής	Πανεπιστήμιο Μακεδονίας

Continued on next page

Department	University
Πληροφορικής	Ελληνικό Μεσογειακό Πανεπιστήμιο

Roles

Below are indicative results from the roles table, which includes the available roles that each faculty member can have in each academic institution.

Table 5.1.3: Roles

role_id	role_title
1	Professor
2	Associate Professor
3	Assistant Professor
4	Lecturer
5	Researcher
6	Visiting Professor
7	Adjunct Professor
8	PhD Student
9	Lab Lecturer
10	Former

Staff

The staff table contains information for each faculty member, such as name and unique Google Scholar ID. Below are some indicative records, while the database contains data for **948** members.

Table 5.1.4: Example of Faculty Members Entries

staff_id	scholar_id	first_name	last_name
1	-0BLhicAAAAJ	Evgenia	Adamopoulou
2	-A1_PpoAAAAJ	Kerstin	Siakas
3	-COFDBMAAAAJ	Antonis	Protopsaltis
4	-E58MA0AAAAJ	Haralampos	Karanikas

Continued on next page

staff_id	scholar_id	first_name	last_name
5	-EzNuW4AAAAJ	Alex	Chroneos
6	-h2va3cAAAAJ	Stefanos	Kollias
7	-hDlKqwAAAAJ	Vasilis	F. Pavlidis
8	-kUKfxsAAAAJ	Kostas	Tsichlas
9	-nTRBSMAAAAJ	Spyridon	Doukakis
10	-sAFa84AAAAJ	Ioannis	Kouretas
11	-UEUsr4AAAAJ	Leonidopoulos	Georgios
12	-UG3Wv0AAAAJ	Nikolaos	Mitianoudis
13	-vblrYcAAAAJ	Michalis	Zervakis

Connection Table of Staff Members with Department and Role

Table 5.1.5: Example Mapping of Staff Members to Departments and Roles

staff_ role_id	staff_id	department_id	role_id
1	1	8	9
2	2	15	10
3	3	5	9
4	4	30	3
5	5	7	1
6	6	8	1
7	7	9	2
8	8	13	3
9	9	20	3
10	10	11	9

Publications

The publications table lists the scientific publications collected from faculty members' profiles. Indicative data are presented below, while in total **more than 140.000** publications have been stored until today(May 2025).

Table 5.1.6: Publication Data

Field	Publication 3	Publication 4
publication_id	85149	85150
publication_scholar_id	u5HHmVD_u08C	u-x6o8ySG0sC
publication_title	Acoustics of children's speech: Developmental changes of temporal and spectral parameters	A comparison of the energy operator and the Hilbert transform approach to signal and speech demodulation
authors	Sungbok Lee, Alexandros Potamianos, Shrikanth Narayanan	Alexandros Potamianos, Petros Maragos
publication_date	1999-03-01	1994-05-01
journal	The Journal of the Acoustical Society of America	Signal Processing
publisher	Acoustical Society of America	Elsevier
citations	1211	363
publication_url	https://scholar.google.com/citations?view_op=view_citation&hl=en&user=pBQViyUAAAAJ&citation_for_view=pBQViyUAAAAJ:u5HHmVD_u08C	https://scholar.google.com/citations?view_op=view_citation&hl=en&user=pBQViyUAAAAJ&citation_for_view=pBQViyUAAAAJ:u-x6o8ySG0sC
publication_year	1999	1994

Connection Table of Publication with Staff Member

This table also contains the order of the staff member in the list of authors.

Table 5.1.7: Example Mapping of Staff to Publications and Author Order

publication_- staff_id	staff_id	publication_id	author_order
1	1	1	4
2	1	2	3

Continued on next page

publication_ staff_id	staff_id	publication_id	author_order
3	1	3	1
4	1	4	1
5	1	5	4
6	1	6	4
7	1	7	3
8	1	8	1
9	1	9	3
10	1	10	2

Staff Statistics

The staff_statistics table collects bibliometric statistics for each faculty member, such as h-index and i10-index, both overall and for the last five years, as well as in different calculation variants. Below are indicative results from the values of these indices.

Table 5.1.8: Staff Statistics

Field	Staff 1	Staff 2	Staff 3
staff_statistic_id	1	2	3
staff_id	1	2	3
total_citations	1597	3122	331
last_5_years_citations	995	2015	320
h_index	18	25	8
last_5_years_h_index	14	18	7
i10_index	35	63	7
last_5_years_i10_index	17	25	6
h_index_local	19	25	8
last_5_years_h_index_local	10	8	7
i10_index_local	36	63	7

Continued on next page

Field	Staff 1	Staff 2	Staff 3
last_5_years_i10_index_local	10	8	6
h_index_from_graph	18	26	8
last_5_years_h_index_from_graph	14	19	7
i10_index_from_graph	40	67	7
last_5_years_i10_index_from_graph	20	29	6

Annual Citations of Members

The table staff_citations_per_year contains time series with the number of citations received by each faculty member per year. These data allow for analysis of research impact over time. Indicative extracts from the relevant records are presented below. The total collected data until now (May 2025) is **more than 20.000**.

Table 5.1.9: Citations per Year for Each Staff Member

ID	Staff ID	Year	Citations
10	1	2015	42
11	1	2014	47
12	1	2013	33
13	1	2012	60
14	1	2011	45
15	1	2010	61
16	1	2009	53
17	1	2008	38
18	1	2007	17
19	1	2006	6
20	2	2024	553
21	2	2023	425
22	2	2022	357

Continued on next page

ID	Staff ID	Year	Citations
23	2	2021	326
24	2	2020	214
25	2	2019	152
26	2	2018	125
27	2	2017	76
28	2	2016	83
29	2	2015	69

Annual Citations of Publications

The table `publication_citations_per_year` records the number of citations received by each publication per year. This information allows the impact of each paper to be tracked over time. Indicative results are given below. The total collected data until now (May 2025) is **more than 600.000**.

Table 5.1.10: Citations per Year for Each Publication

ID	Publication ID	Year	Citations
1	1	2024	124
2	1	2023	127
3	1	2022	87
4	1	2021	64
5	1	2020	11
7	2	2024	3
8	2	2023	4
9	2	2022	4
10	2	2021	3
11	2	2020	6
12	2	2019	5
13	2	2018	1

Continued on next page

ID	Publication ID	Year	Citations
14	2	2017	13

5.1.2. Stored Procedures

In the following, indicative results from selected stored procedures, which are used to produce aggregate statistics from the database, are presented. For a better understanding and analysis of the data, some results are also accompanied by graphs created using Python.

Procedure `get_overall_stats_by_staff_ids()`

The stored procedure `get_overall_stats_by_staff_ids()` calculates overall statistics for a set of faculty members, such as number of publications, total citations, h-index and i10-index, as well as averages and variances of these quantities. Below are indicative results:

The example below uses staff IDs ranging from 1 to 190.

Table 5.1.11: Overall Aggregated Staff Statistics

Field	Value
count_staff	190
total_pubs	27818
total_citations	638506
total_pubs_5y	7779
total_citations_5y	94979
avg_pubs_per_m	147.2
avg_cits_per_m	3378.3
avg_age	26.76
avg_pubs_per_m_per_y	5.3
avg_cits_per_m_per_y	119.1
avg_h_index	22.6
avg_h_index_local	22.6
avg_h_index_from_graph	22.9
avg_h_index_5y	14.2

Continued on next page

Field	Value
avg_h_index_local_5y	7.6
avg_h_index_from_graph_5y	14.4
avg_i10_index	52.4
avg_i10_index_local	52.5
avg_i10_index_from_graph	54.4
avg_i10_index_5y	27.7
avg_i10_index_local_5y	9.5
avg_i10_index_from_graph_5y	28.9

Procedure `get_department_stats_by_depts_and_roles()`

The stored procedure `get_department_stats_by_depts_and_roles()` returns aggregate statistics by department, based on selected roles. It includes, among other things, number of members, total and five-year publications and citations, performance indicators and measures of dispersion. Below are illustrative results and graphs to help visualize the differences between departments.

Table 5.1.12: Example Result of Stored Procedure `get_department_stats_by_depts_and_roles()` for two Departments and all the Roles

Field	Department 1	Department 2
department_id	1	2
short_code	csd@uoc	dai@uom
staff_count	23	35
total_pubs	4767	4237
total_citations	161969	82752
avg_h_index	39.9	26.8
avg_h_index_5y	21.6	19.0
avg_h_index_local	39.9	26.8
avg_h_index_local_5y	10.6	10.4
avg_h_index_from_graph	40.2	26.8

Continued on next page

Field	Department 1	Department 2
avg_h_index_from_graph_5y	21.7	19.1
avg_i10_index	119.0	62.9
avg_i10_index_5y	50.0	38.8
avg_i10_index_local	119.3	63.0
avg_i10_index_local_5y	13.5	13.5
avg_i10_index_from_graph	123.6	64.6
avg_i10_index_from_graph_5y	52.0	40.0
avg_pubs	331.8	172.9
avg_pubs_5y	69.9	46.7
avg_citations	8956.6	3548.6
avg_citations_5y	665.1	753.3
avg_age	36.35	26.04
cv_pubs	72.1	48.3
cv_cits	72.5	71.1
max_pubs	829	297
min_pubs	27	23
max_cits	36426	8595
min_cits	309	113
max_h_index	56	46
min_h_index	10	5
max_i10_index	223	131
min_i10_index	10	2

For the example below, the stored procedure `get_department_stats_by_depts_and_roles()` was executed with appropriate filters to include 10 departments and all available roles. The result was used to create tables and graphs showing the relative performance of the departments on key bibliometric parameters. In the graphs that follow, basic statistics obtained from the execution of the procedure are shown, such as: number of faculty members (`staff_count`), total publications (`total_pubs`), total citations (`total_citations`), average h-index, average i10-index, as well as the average age of scientific activity (`avg_age`) for each department.

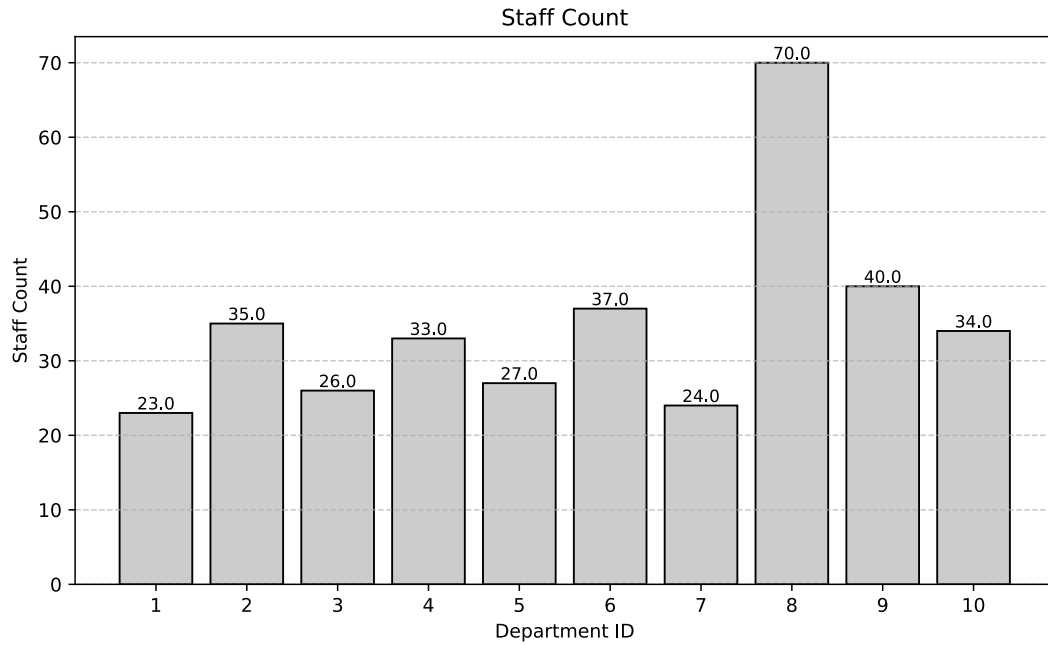


Figure 5.1.1: Vertical Bar Chart of the Number of Members for 10 Departments

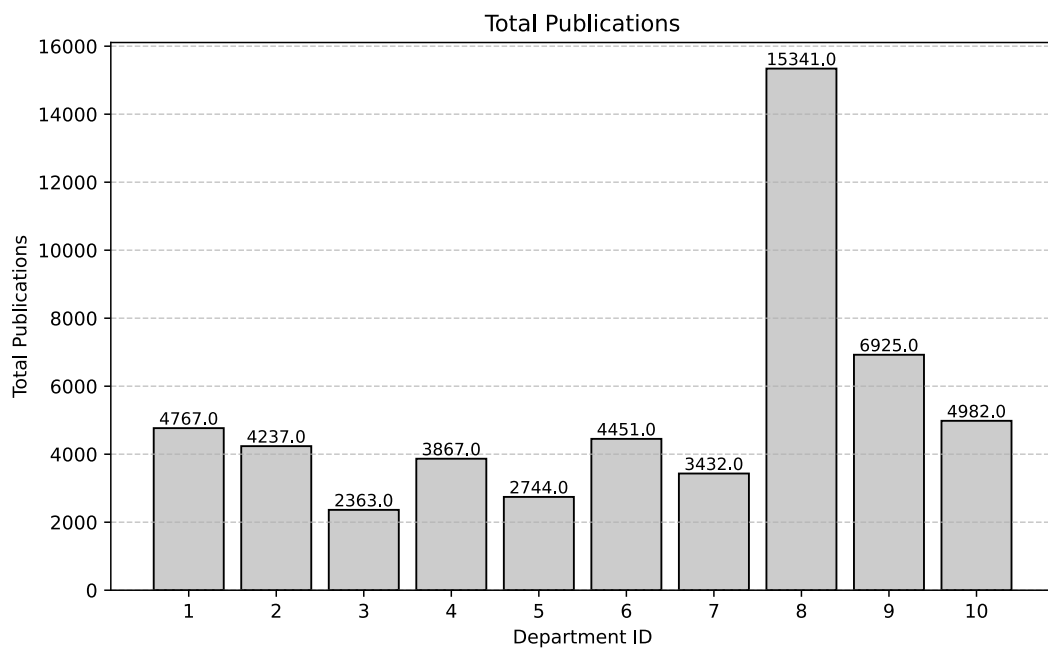


Figure 5.1.2: Vertical Bar Chart of Total Publications for 10 Departments

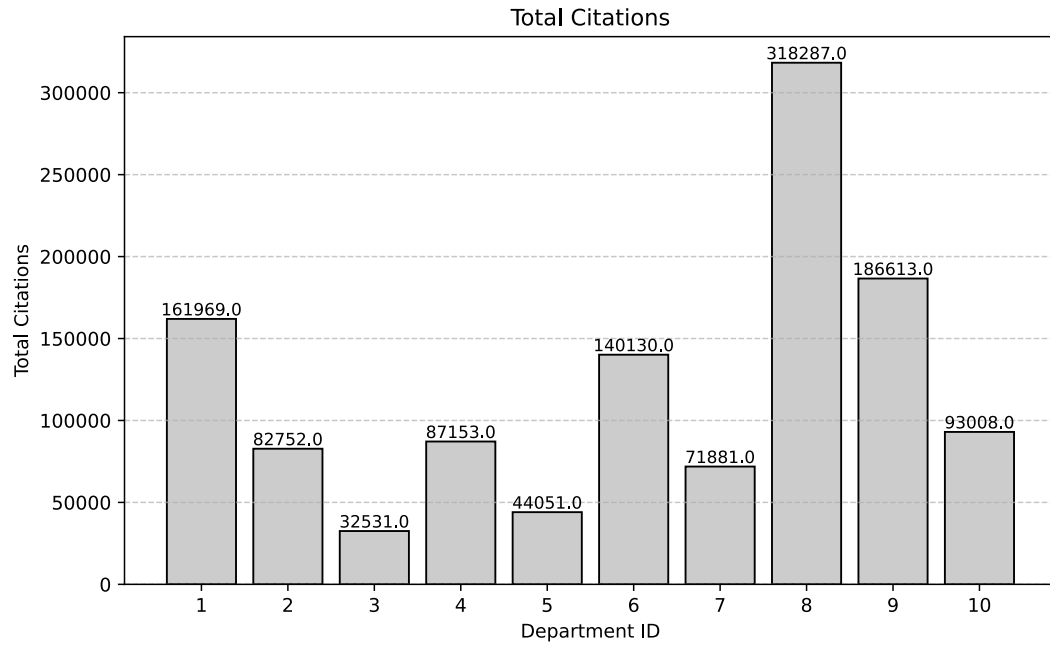


Figure 5.1.3: Vertical Bar Chart of Total Citations for 10 Departments

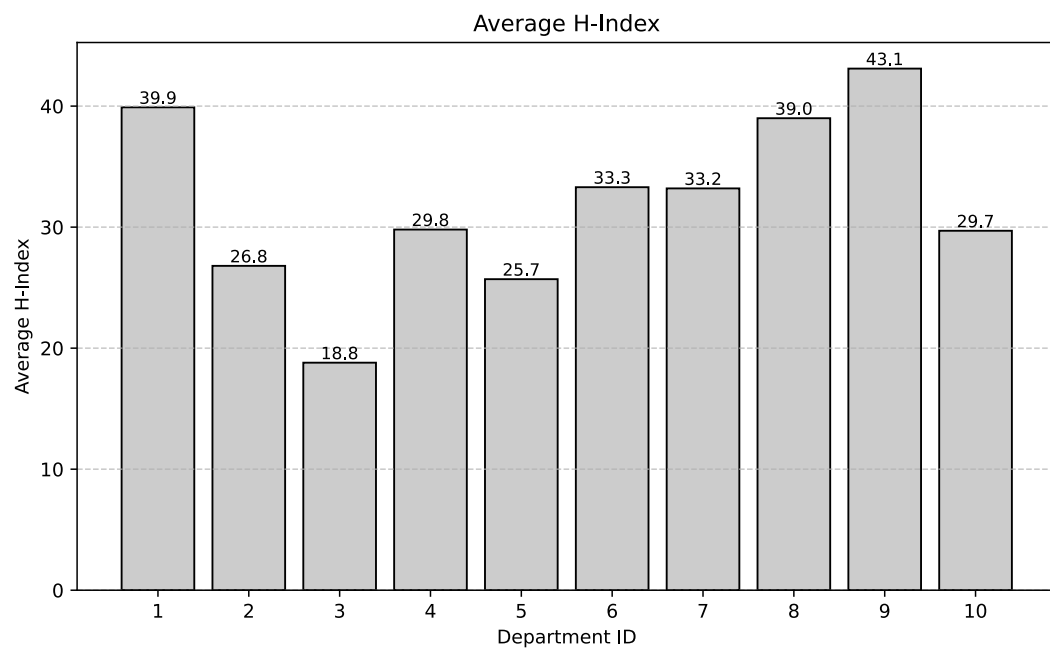


Figure 5.1.4: Vertical Bar Chart of Average H-index for 10 Departments

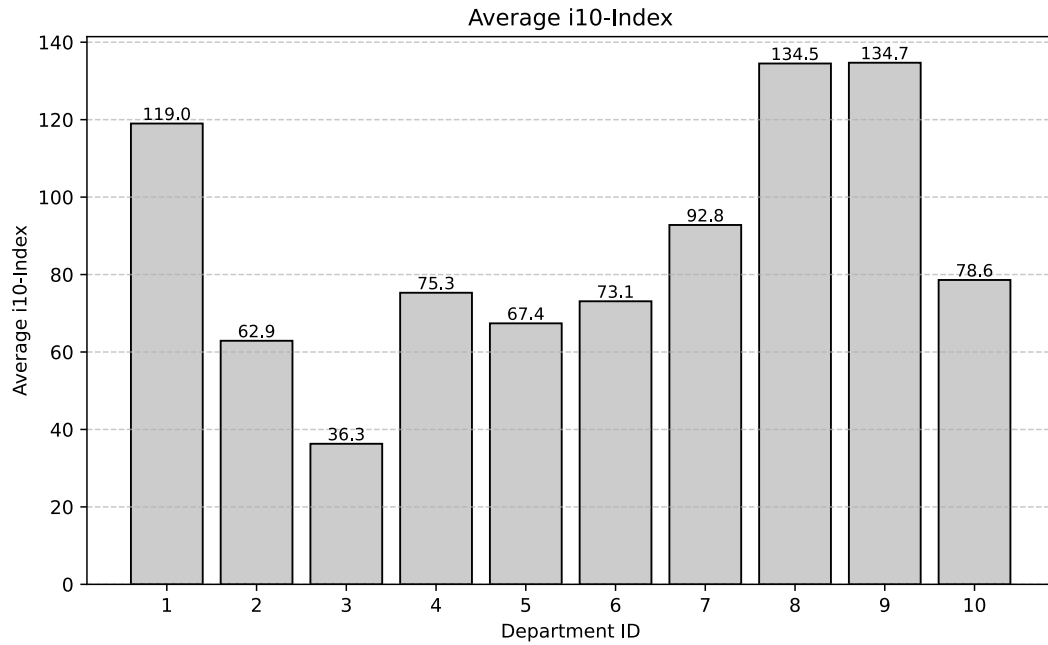


Figure 5.1.5: Vertical Bar Chart of Average i10-index for 10 Departments

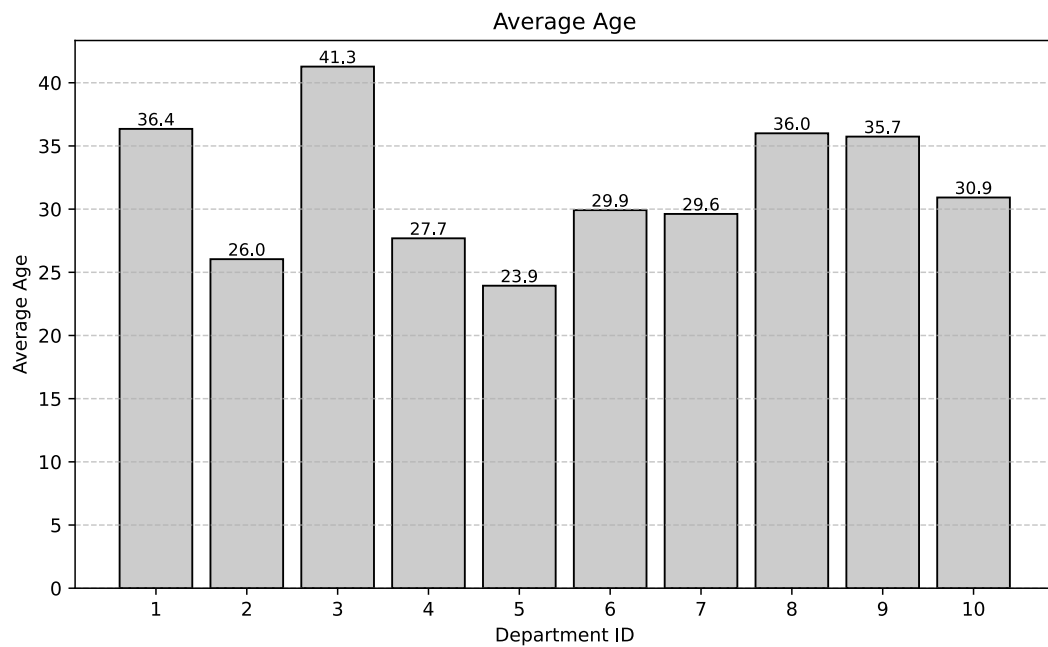


Figure 5.1.6: Vertical Bar Chart of Average Academic Age of Members for 10 Departments

Procedure get_all_staff_summary()

The stored procedure `get_all_staff_summary(dept_ids, role_ids)` returns a set of detailed bibliometric data for each faculty member belonging to the selected departments and roles. The results include the number of publications and citations (total and five-year), the h-index and i10-index in various versions, and the age of scientific output.

The example result below uses department ID 15 and role ID 2.

Table 5.1.13: Result returned by the stored procedure `get_all_staff_summary()`

Field	Member 1	Member 2
staff_id	76	881
first_name	Antonis	Athanasios
last_name	Sidiropoulos	C. Iossifides
scholar_id	42WdccQAAAAJ	xxIK3NwAAAAJ
department_id	15	15
role_id	2	2
total_citations	1344	812
last_5_years_citations	294	616
h_index	13	12
last_5_years_h_index	7	9
h_index_local	13	12
last_5_years_h_index_local	4	6
h_index_from_graph	13	12
last_5_years_h_index_from_graph	7	9
i10_index	14	13
last_5_years_i10_index	4	8
i10_index_local	14	13
last_5_years_i10_index_local	1	3
i10_index_from_graph	14	13
last_5_years_i10_index_from_graph	4	8

Continued on next page

Field	Member 1	Member 2
age	22	31
total_publications	45	59
total_publications_5y	13	13

Procedure `get_all_publications_by_staff()`

The stored procedure `get_publications(staff_id)` returns the total number of publications associated with a particular faculty member, as recorded in the publications table via the publications_staff associative table.

Table 5.1.14: Publication Details for a Specific Member

Field	Publication 1	Publication 2
publication_id	11762	11763
publication_title	Generalized Hirsch h-index for disclosing latent facts in citation networks	Generalized comparison of graph-based ranking algorithms for publications and authors
authors	Antonis Sidiropoulos, Dimitrios Katsaros, Yannis Manolopoulos	Antonis Sidiropoulos, Yannis Manolopoulos
author_order	1	1
publication_date	2007-06-17	2006-12-31
journal	Scientometrics	Journal of Systems and Software
publisher	Akadémiiai Kiadó, co-published with Springer Science+ Business Media BV, Formerly Kluwer Academic Publishers BV	Elsevier
citations	636	98
publication_url	https://scholar.google.com/citations?view_op=view_citation&hl=en&user=42WdccQAAAAJ&citation_for_view=42WdccQAAAAJ:Dip1O2bNi0gC	https://scholar.google.com/citations?view_op=view_citation&hl=en&user=42WdccQAAAAJ&citation_for_view=42WdccQAAAAJ:2osOgNQ5qMEC

Procedures `get_publications_per_year_by_staff()` and `get_citations_per_year_by_staff()`

Table 5.1.15: Result of stored procedure `get_publications_per_year_by_staff` for staff ID 76

Year	Total Publications
2003	1
2005	3
2006	5
2007	4
2008	3
2009	1
2010	1
2012	1
2013	1
2014	3
2015	2
2016	2
2017	3
2018	2
2020	3
2021	3
2022	1
2024	5
2025	1

Table 5.1.16: Result of stored procedure `get_citations_per_year_by_staff` for staff ID 76

Year	Total Citations
2003	0
2005	191
2006	130
2007	664
2008	63
2009	40
2010	82
2012	0
2013	1
2014	8
2015	26
2016	44
2017	10
2018	14
2020	10
2021	13
2022	0
2024	59
2025	0

5.2. API Responses

This section presents illustrative responses of the RESTful API, as obtained by executing requests via the Postman and cURL tools. The results shown are based on stored procedures of the database or simple select queries(e.g., departments, roles), which implement the underlying logic for processing and extracting data. The following images demonstrate the correct operation of the endpoints and the success of the interface between the backend and the database.

Results of All Departments Returned

Figure 5.2.1: Example JSON response from GET /departments/all

```
[
  {
    "department_id": 1,
    "short_code": "csd@uoc",
    "department_title": "Επιστήμης" Υπολογιστών",
    "university": "Πανεπιστήμιο" Κρήτης",
    "city": "Ηράκλειο",
    "department_url": "http://www.csd.uoc.gr/",
    "department_staff_url": "https://www.csd.uoc.gr/index.jsp?content=
academic_staff&openmenu=demoAcc2&lang=gr",
    "department_supporting_staff_url": "https://www.csd.uoc.gr/index.jsp?
content=edip_etep&openmenu=demoAcc2&lang=gr",
    "created_at": "2024-12-19 23:28:04"
  },
  {
    "department_id": 2,
    "short_code": "dai@uom",
    "department_title": "Εφαρμοσμένης" Πληροφορικής",
    "university": "Πανεπιστήμιο" Μακεδονίας",
    "city": "Θεσσαλονίκη",
    "department_url": "https://www.uom.gr/dai",
    "department_staff_url": "https://www.uom.gr/en/dai/academic-staff",
    "department_supporting_staff_url": "https://www.uom.gr/en/dai/special-
teaching-technical-personnel",
    "created_at": "2024-12-19 23:28:04"
  },
  ...
]
```

Results of All Roles Returned

Figure 5.2.2: Example JSON response from GET /roles

```
[
  {
    "role_id": 1,
    "role_title": "Professor",
    "created_at": "2024-12-19 23:28:04"
  },
  {
    "role_id": 2,
    "role_title": "Associate Professor",
    "created_at": "2024-12-19 23:28:04"
  },
  {
    "role_id": 3,
    "role_title": "Assistant Professor",
    "created_at": "2024-12-19 23:28:04"
  },
  ...
]
```

Summary Statistics for Selected Staff Members by Departments and Roles

Figure 5.2.3: Example JSON response from GET /staff/stats/token?departments=<token>&roles=<token>

```
[
  {
    "staff_id": 76,
    "first_name": "Antonis",
    "last_name": "Sidiropoulos ",
    "scholar_id": "42WdccQAAAAJ",
    "department_id": 15,
    "role_id": 2,
    "total_citations": 1344,
    "last_5_years_citations": 294,
    "h_index": 13,
    "last_5_years_h_index": 7,
    "h_index_local": 13,
    "last_5_years_h_index_local": 4,
    "h_index_from_graph": 13,
    "last_5_years_h_index_from_graph": 7,
    "i10_index": 14,
    "last_5_years_i10_index": 4,
    "i10_index_local": 14,
    "last_5_years_i10_index_local": 1,
    "i10_index_from_graph": 14,
    "last_5_years_i10_index_from_graph": 4,
    "age": "22",
    "total_publications": 45,
    "total_publications_5y": "13"
  },
  ...
]
```

Publications per Year for a Selected Staff Member

Figure 5.2.4: Example JSON response from GET /staff/publications/per-year?token=<token>

```
{
  "staff_id": 1,
  "publications_per_year": [
    {
      "publication_year": "2005",
      "total_publications": 3
    },
    {
      "publication_year": "2006",
      "total_publications": 7
    },
    {
      "publication_year": "2007",
      "total_publications": 8
    },
    ...
  ]
}
```

Citations per Year for a Selected Staff Member

Figure 5.2.5: Example JSON response from GET /staff/citations/per-year?token=<token>

```
{
  "staff_id": 1,
  "citations_per_year": [
    {
      "publication_year": "2005",
      "total_citations": "80"
    },
    {
      "publication_year": "2006",
      "total_citations": "148"
    },
    {
      "publication_year": "2007",
      "total_citations": "47"
    },
    ...
  ]
}
```

Overall Aggregated Statistics for Selected Staff Members

Figure 5.2.6: Example JSON response from GET /staff/overall?token=<token>

```
{
  "count_staff": 191,
  "total_pubs": "28167",
  "total_citations": "649603",
  "total_pubs_5y": "7832",
  "total_citations_5y": "95563",
  "avg_pubs_per_m": "148.2",
  "avg_cits_per_m": "3419.0",
  "avg_age": "26.84",
  "avg_pubs_per_m_per_y": "5.3",
  "avg_cits_per_m_per_y": "119.8",
  "avg_h_index": "22.8",
  "avg_h_index_local": "22.8",
  "avg_h_index_from_graph": "23.0",
  "avg_h_index_5y": "14.3",
  "avg_h_index_local_5y": "7.6",
  "avg_h_index_from_graph_5y": "14.5",
  "avg_i10_index": "53.0",
  "avg_i10_index_local": "53.1",
  "avg_i10_index_from_graph": "55.0",
  "avg_i10_index_5y": "28.1",
  "avg_i10_index_local_5y": "9.5",
  "avg_i10_index_from_graph_5y": "29.3"
}
```

Department-Level Aggregated Statistics by Roles and Departments

Figure 5.2.7: Example JSON response from GET /departments/statistics/token?departments=<token>&roles=<token>

```
[
  {
    "department_id": 15,
    "short_code": "iee@ihu",
    "staff_count": 13,
    "total_pubs": 580,
    "total_citations": "7818",
    "avg_h_index": "16.0",
    "avg_h_index_5y": "11.1",
    "avg_h_index_local": "16.0",
    "avg_h_index_local_5y": "5.2",
    "avg_h_index_from_graph": "16.2",
    "avg_h_index_from_graph_5y": "11.5",
    "avg_i10_index": "29.2",
    "avg_i10_index_5y": "13.0",
    "avg_i10_index_local": "29.2",
    "avg_i10_index_local_5y": "5.1",
    "avg_i10_index_from_graph": "30.8",
    "avg_i10_index_from_graph_5y": "14.6",
    "avg_pubs": "94.5",
    "avg_pubs_5y": "24.7",
    "avg_citations": "1430.9",
    "avg_citations_5y": "228.2",
    "avg_age": "24.84",
    "cv_pubs": 73.2,
    "cv_cits": 92.3,
    "max_pubs": 189,
    "min_pubs": 2,
    "max_cits": "3278",
    "min_cits": "12",
    "max_h_index": 25,
    "min_h_index": 2,
    "max_i10_index": 63,
    "min_i10_index": 1
  },
  ...
]
```

The analysis of the results obtained from the Google Scholar data, as collected and organized through this application, clearly demonstrated the usefulness of the design and implementation of the database and stored procedures. The ability to extract and visualise these statistics at a mass level proves that the system is suitable for use by both academic administrations and individual researchers wishing to evaluate or compare research activity. At the same time, it offers a basis for further data enhancement, metrics enhancement or linking to external platforms in the future.

This chapter demonstrates the project's practical applicability and the information system's potential to support targeted analysis, feedback and documentation of scientific output in an academic environment.

Chapter 6

Discussion

6.1. Comparison between the Design and Implementation and the Initial Objectives

The main objective of this work was to develop an integrated system for the collection, storage, organization, and presentation of bibliometric data for faculty members, using Google Scholar as the primary source. Additionally, it served as a redesign and extension of the older OMEA system (<https://omea.iee.ihu.gr/citations/>), with the aim of modernizing it and better supporting evaluation needs. At the outset of the design phase, clear functional and technical objectives were established, including:

- Automatic data collection by scraping.
- The correct and efficient storage of data in a relational database.
- The ability to extract statistical and comparative indicators.
- The creation of a backend API for future connection to a presentation application.

In practice, the final implementation largely satisfies all the above objectives. The scraping mechanism works stably, the database is structured in 3NF format, and the stored procedures allow the generation of complex statistics with minimal overhead on the database server.

Certain adjustments were made along the way, mainly for performance and scalability reasons. For example, while the initial intention was to create small stored procedures per statistic size, eventually more clustered and optimized implementations were adopted, reducing the overall backend load and speeding up responses.

The fact that the frontend is not implemented at this stage does not negate the completeness of the backend. On the contrary, it reinforces the separated architecture of the application, leaving the door open for future expansion.

6.2. Technical and Design Decisions

The development of the system was based on technologies and practices that serve both the efficiency and future maintainability of the application. The main technical and design choices adopted are:

1. **Use of Python for Web Scraping:** The data collection tool was developed in Python, using libraries such as requests, BeautifulSoup and re to identify and extract the necessary data from Google Scholar. Special attention was paid to the robustness of the scraper, such as handling non-ASCII characters and HTML parsing errors, in order to prevent failures during execution.
2. **Relational Database:** The database design strictly followed the rules of the third normal form (3NF) to avoid redundant storage and maintain data consistency. The separation of entities (e.g. staff, publications, departments) and the use of correlation tables ensure flexibility and a clean structure.
3. **Stored Procedures for Aggregation and Performance:** Instead of relying on multiple queries from the backend, it was chosen to use stored procedures that implement aggregate operations directly at the database level. These procedures are fully optimized with sub-queries, LEFT JOINS and GROUP BY, offering great speed on complex queries.
4. **Modularity and Separation of Concerns:** All PHP scripts are organized around segregated responsibility. Each API file handles a specific endpoint, while the database and helper functions remain in common reusable modules. This makes the code easily extensible and maintainable.

These options combine simplicity of implementation with functional completeness, and ensure that the project can evolve and incorporate future new features or interfaces with minimum redesign.

6.3. Problems Encountered and Solutions Implemented

During the implementation of the project several technical and logical challenges were encountered, which were addressed with targeted solutions:

Inconsistencies and Unexpected Characters in Data from Google Scholar

During the scraping process, instances were observed where publication titles contained special or unreadable characters, such as the mathematical symbol "m" (in a different encoding from ASCII). This caused a failure in conversion and storage, leading to errors. To avoid crashes, a check was added to detect if the string is ASCII and, otherwise, replace the title with "NON_ASCII_TITLE".

Date validity issues

There were several records of publications with invalid, incomplete or problematic dates (e.g. null, 0, -1). To ensure accuracy in temporal calculations (such as age or five-year statistics), a logical check was applied to accept only those chronologies that fall within a reasonable range (e.g. 1900-current year).

Low Performance with Multiple Calls of Stored Procedures

Initially, an attempt was made to call several individual stored procedures (one for each statistic), but this proved to be inefficient in a multi-user environment or large datasets. The solution came by designing aggregation procedures that return the necessary data in bulk, drastically reducing the cost of queries.

Issues with Transmitting Large URLs in the API

The use of many parameters (e.g. many `staff_ids` values) caused overly large URLs, unsupported by all browsers. The solution was to use a token-based approach, with compression (GZIP) and encoding (base64), which allows complex parameters to be safely transferred in a short URI.

Data Cleaning and Normalization

During data integration into the database, duplicate records or inconsistencies in author and publication matches were identified. To address these, additional checking and filtering logic was applied to enhance the quality of the stored data.

These challenges served as feedback points for the improvement of the system and strengthened the technical design of the application under real-life conditions.

6.4. Reproducibility and System Extensibility

The entire data collection and processing process is fully automated and controlled. The scraper can be re-executed with any input dataset (e.g. list of scholar IDs) to retrieve, filter and store data in a new database or update an existing one. The stored procedures guarantee consistent and predictable results, regardless of data size or content.

The modular structure of the backend, including the scraper, databases and API, allows for easy integration of new features, such as support for additional data sources (e.g. Scopus, ORCID), analysis at new levels (e.g. per project, research field or collaboration), creation of dashboards or dynamic graphs for presentation, and the addition of authentication/authorization mechanisms to the API.

In addition, the use of widely used technologies (PHP, MySQL, Python) and the documentation of stored procedures and routes ensure that the project can be easily used or adapted by other developers, even in different institutions or operating environments.

Replicability and ease of extension make this solution not just a research application, but a solid and scalable basis for future development and exploitation in the field of scientific analysis and assessment.

6.5. Overall Assessment and Development Experience

The development of the system was a particularly demanding but also essentially didactic experience, combining the application of theoretical knowledge with the tackling of practical real-world problems.

During implementation:

- It deepened the understanding of concepts related to database design, bibliometrics, and retrieval and organization of data from semi-structured sources such as Google Scholar.
- Important skills in efficiency optimization were developed, both at the SQL level (through aggregation and combinatorial queries) and at the application level (latency reduction, caching with tokens, use of GZIP).
- They handled data quality issues such as duplicates, invalid dates, character encodings and incomplete values, enhancing experience in handling edge cases and exceptions.

The most important challenge was to ensure scalability and stability, as the system had to remain functional with large amounts of data and different configurations. Solving such issues led to mature design decisions, such as choosing massive stored procedures instead of multiple granular queries.

Finally, the development of the backend API reinforced the understanding of how to design reusable and secure data interfaces for frontend applications or third-party users.

The overall experience enhanced technical skills, problem-solving ability, and an organized approach to complex tasks, providing essential skills for future development or research activity.

Chapter 7

Conclusion

This thesis focused on the design and implementation of an integrated system for the collection, storage, analysis and display of bibliometric data, drawing information from Google Scholar. Through the gradual development of individual parts of the system (scraper, database, stored procedures, REST API), the goal of creating a reliable and scalable infrastructure capable of serving a variety of usage scenarios, of a research or administrative nature, was achieved.

The project implemented:

- A flexible scraping system with data validity checks, edge case management (e.g. non-ASCII characters, incomplete or outlier data), and with appropriate optimizations to avoid performance errors or inconsistencies.
- A strong relational database model, organized according to the principles of the 3rd Normal Form (3NF), ensuring logical consistency, limiting redundancy and ease of expansion. Complete control of records by means of auxiliary tables (records) for the detection of insertions, modifications and deletions of records.
- A set of basic and complex stored procedures that allow efficient production of statistical data, either for a single or not, faculty member or for departments and roles.
- A RESTful API backend, which is the "bridge" between the database and future frontend applications or third-party services, implemented with emphasis on security (token-based access), performance (GZIP encoding), and modular design.

7.1. Overall Conclusions

The paper highlighted the importance of a holistic approach to building such a system: proper design at the base level, efficient data collection, statistical processing, and accessibility through APIs make up a set that can be easily exploited and extended without significant changes to its structure.

Furthermore, the work has demonstrated in practice the importance of using aggregation-based stored procedures, as well as the differentiation between granular and summarized statistics, achieving significant performance improvements in large-scale data retrieval.

7.2. Outlook and Future Plans

Although the system is fully functional at the backend level, there are clear prospects for future development and expansion. One of the main directions concerns the implementation of a frontend interface, so that the end user can interact directly with the data. At the same time, the ability to export data in formats such as CSV, Excel or PDF can be added, facilitating their use for research or administrative purposes.

The addition of an administrative interface (admin panel) will allow data editing and iterative checking or refreshing through scraping. In addition, the system can be extended by incorporating analytical techniques, such as predictive models or clustering, to predict research performance or group faculty members based on their characteristics.

Finally, the integration of data from other scientific databases, such as Scopus or Web of Science, will enhance benchmarking and provide a more comprehensive picture of research activity.

7.3. Concluding Evaluation

The development of this system was a valuable experience combining technological, design and practical skills. This work was not limited to simple technical implementation, but was a substantial exercise in understanding information, optimizing it, and turning it into tools useful to users. The result is a project with real functional value and with clear potential for implementation and development.

Bibliography

- [1] National Documentation Centre (EKT), *Greek scientific publications in international journals - about*, <https://metrics.ekt.gr/en/scientific-publications/about>, Accessed: 2025-05-21.
- [2] I. Z. Koukoutsidis, “Department-level comparison of universities’ scientific output: A bibliometric study of greek universities,” *arXiv preprint arXiv:2212.12032*, 2022.
- [3] S. Papavlasopoulos, *Bibliometrics*. Open Academic Editions Kallipos, 2015, Available in the Kallipos repository. [Online]. Available: <https://repository.kallipos.gr/handle/11419/4755>.
- [4] International Science Council, *The future of research evaluation: A synthesis of current debates and developments*, Accessed: April 2025, 2023. [Online]. Available: <https://council.science/publications/the-future-of-research-evaluation-a-synthesis-of-current-debates-and-developments/>.
- [5] L. Bornmann and H.-D. Daniel, “What do we know about the h index?” *Journal of the American Society for Information Science and Technology*, vol. 58, no. 9, pp. 1381–1385, 2007. doi: 10.1002/asi.20609.
- [6] L. Waltman and N. J. van Eck, “A new methodology for constructing a publication-level classification system of science,” *Journal of the American Society for Information Science and Technology*, vol. 64, no. 12, pp. 2370–2384, 2013. doi: 10.1002/asi.22976.
- [7] E. A. Fong and A. W. Wilhite, “Authorship and citation manipulation in academic research,” *PLOS ONE*, vol. 12, no. 12, e0187394, 2017. doi: 10.1371/journal.pone.0187394.
- [8] D. Hicks, P. Wouters, L. Waltman, S. de Rijcke, and I. Rafols, “The leiden manifesto for research metrics,” *Nature*, vol. 520, no. 7548, pp. 429–431, 2015. doi: 10.1038/520429a.
- [9] E. Orduña-Malea, J. M. Ayllón, A. Martín-Martín, and E. Delgado López-Cózar, “Methods for estimating the size of google scholar,” *Scientometrics*, vol. 104, no. 3, pp. 931–949, 2015. doi: 10.1007/s11192-015-1625-5.
- [10] A.-W. Harzing and S. Alakangas, “Google scholar, scopus and the web of science: A longitudinal and cross-disciplinary comparison,” *Scientometrics*, vol. 106, no. 2, pp. 787–804, 2016. doi: 10.1007/s11192-015-1798-9.
- [11] E. Delgado López-Cózar, N. Robinson-García, and D. Torres-Salinas, “The google scholar experiment: How to index false papers and manipulate bibliometric indicators,” *Journal of the Association for Information Science and Technology*, vol. 65, no. 3, pp. 446–454, 2014. doi: 10.1002/asi.23056.

-
- [12] R. Pranckutė, “Web of science (wos) and scopus: The titans of bibliographic information in today’s academic world,” *Publications*, vol. 9, no. 1, p. 12, 2021. doi: 10.3390/publications9010012.
 - [13] Elsevier, *Scopus api guide*, Accessed: April 2025, 2023. [Online]. Available: https://dev.elsevier.com/guides/Scopus%20API%20Guide_V1_20230907.pdf.
 - [14] Clarivate Analytics, *Web of science apis*, Accessed: April 2025, 2024. [Online]. Available: <https://clarivate.libguides.com/c.php?g=1140539&p=10430704>.
 - [15] R. Lawrence and N. Penfold, “Equity in open research: Challenges and recommendations for inclusion,” *Scholarly Assessment Reports*, vol. 4, no. 1, 2022. doi: 10.29024/sar.36.
 - [16] M. Visser, N. J. van Eck, and L. Waltman, “Large-scale comparison of bibliographic data sources: Scopus, web of science, dimensions, crossref, and microsoft academic,” *Quantitative Science Studies*, vol. 2, no. 1, pp. 20–41, 2021. doi: 10.1162/qss_a_00112.
 - [17] ORCID Support, *Importing works from other systems*, Accessed: April 15, 2025, 2025. [Online]. Available: <https://support.orcid.org/hc/en-us/articles/360006973653-Importing-works-from-other-systems>.
 - [18] J. Kim and J. Owen-Smith, “Orcid-linked labeled data for evaluating author name disambiguation at scale,” *arXiv preprint arXiv:2102.03237*, 2021. [Online]. Available: <https://arxiv.org/abs/2102.03237>.
 - [19] S. Munzert, C. Rubba, P. Meißner, and D. Nyhuis, *Automated Data Collection with R: A Practical Guide to Web Scraping and Text Mining*. Chichester, UK: John Wiley & Sons, 2014, ISBN: 9781118834817.
 - [20] M. Gusenbauer, “Google scholar to overshadow them all? comparing the sizes of 12 academic search engines and bibliographic databases,” *Scientometrics*, vol. 118, no. 1, pp. 177–214, 2019. doi: 10.1007/s11192-018-2958-5.
 - [21] H. F. Moed, W. J. M. Burger, J. G. Frankfort, and A. F. J. van Raan, “The use of bibliometric data for the measurement of university research performance,” *Research Policy*, vol. 14, no. 3, pp. 131–149, 1985. doi: 10.1016/0048-7333(85)90012-5.
 - [22] N. M. Vaxevanidis, “On the evaluation of the quality of research in greek heis using bibliometric indices,” in *Proceedings of the 6th International Conference ICQME*, Tivat, Montenegro, 2011, pp. 63–72.
 - [23] G. Zachos, “Research output evaluation of two university departments in greece with the use of bibliometric indicators,” *Scientometrics*, vol. 21, no. 2, pp. 195–221, 1991. doi: 10.1007/BF02017569.
 - [24] G. V. Rossum, *The history of python: A brief timeline of python*, Last accessed 25 March 2025, 2009. [Online]. Available: <https://python-history.blogspot.com/2009/01/brief-timeline-of-python.html>.
 - [25] Python Software Foundation, *The python standard documentation*, Accessed: March 2025, 2025. [Online]. Available: <https://docs.python.org/3/>.
 - [26] S. Mehta and G. P. (Jain), “An improving approach for fast web scrapping using machine learning and selenium automation,” *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, vol. 8, no. 10, pp. 434–438, 2019.

-
- [27] Selenium Project, *Selenium documentation*, Accessed: March 2025, 2025. [Online]. Available: <https://www.selenium.dev/documentation/>.
 - [28] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, 7th ed. Pearson, 2015, ISBN: 9780133970777.
 - [29] D. D. Chamberlin and R. F. Boyce, “Sequel: A structured english query language,” IBM Research Laboratory, San Jose, California, Research Report RJ1333, 1974.
 - [30] J. R. Groff and P. N. Weinberg, *SQL: The Complete Reference*. Berkeley, California: Osborne/McGraw-Hill, 1999, EnglishOnlineClub.com PDF edition, ISBN: 0-07-211845-8.
 - [31] Oracle Corporation, *Mysql 8.0 reference manual*, <https://dev.mysql.com/doc/refman/8.0/en/>, Accessed: 2025-04-01, 2025.
 - [32] International Organization for Standardization, *Iso/iec 9075: Information technology — database languages — sql*, <https://www.iso.org/standard/63555.html>, Accessed: 2025-04-01, 2016.
 - [33] B. Salzberg, “Third normal form made easy,” *SIGMOD Record*, vol. 15, no. 4, pp. 13–15, 1986. doi: 10.1145/16301.16302. [Online]. Available: <https://dl.acm.org/doi/10.1145/16301.16302>.
 - [34] L. Welling and L. Thomson, *PHP and MySQL Web Development*, 4th ed. Pearson Education, 2009, ISBN: 9780672329166.
 - [35] A. A. Prayogi, M. Niswar, Indrabayu, and M. Rijal, “Design and implementation of rest api for academic information system,” in *IOP Conference Series: Materials Science and Engineering*, vol. 875, IOP Publishing, 2020, p. 012 047. doi: 10.1088/1757-899X/875/1/012047.
 - [36] M. Rodríguez *et al.*, *Rest api tutorial*, <https://restfulapi.net/>, Accessed: 2025-04-01, 2025.
 - [37] T. P. Group, *Php manual*, <https://www.php.net/manual/en/>, Accessed: 2025-04-01, 2025.
 - [38] A. Becker, *Heidisql*, 2024. [Online]. Available: <https://www.heidisql.com/>.
 - [39] O. Corporation, *Mysql workbench*, 2024. [Online]. Available: <https://dev.mysql.com/doc/workbench/en/>.
 - [40] JetBrains, *Intellij idea*, 2024. [Online]. Available: <https://www.jetbrains.com/idea/>.
 - [41] Microsoft, *Visual studio code*, <https://code.visualstudio.com/>, 2024.
 - [42] C. Ltd., *Ubuntu server documentation*, <https://ubuntu.com/server>, 2024.
 - [43] T. Corporation, *Termius ssh client*, <https://termius.com/>, 2024.
 - [44] T. Ylonen and C. Lonvick, *The secure shell (ssh) protocol architecture*, RFC 4251, <https://datatracker.ietf.org/doc/html/rfc4251>, 2006.
 - [45] O. Developers, *Secure copy protocol (scp)*, <https://man.openbsd.org/scp>, 2024.
 - [46] I. Postman, *Postman api platform*, <https://www.postman.com/>, 2024.
 - [47] J. Kew, *Xelatex: A modern tex engine for unicode and opentype*, <https://tug.org/xetex/>, 2024.

- [48] Overleaf, *Overleaf: Online latex editor*, <https://www.overleaf.com/>, 2024.
- [49] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, 7th ed. Boston, MA: Pearson, 2016.
- [50] P. Ziegler and K. R. Dittrich, “Data versioning and provenance for scientific workflows,” in *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*, Springer, 2014. doi: 10.1007/978-3-319-08807-5_4.
- [51] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 8th. Pearson, 2021, Chapter 2: Application Layer, ISBN: 9780136681557.
- [52] S. Bird, E. Loper, and E. Klein, “Natural language processing with python,” *O’Reilly Media*, 2009, <https://www.nltk.org/book/>.