In defense of extreme database-centric architecture

En defensa de una arquitectura extrema centrada en la base de datos

Em defesa de uma arquitetura extrema centrada em banco de dados

Alfonso Vicente[1]

Recibido: 28/05/2023 Aceptado: 28/05/2023

Summary. - A famous aphorism in computer science goes: "All problems in computer science can be solved by another level of indirection", often expanded by the humorous clause "except for the problem of too many levels of indirection". After 30 years of applying the first aphorism, multi-tier architectures (i.e. architectures with many levels of indirection) have become the de facto standard for web applications, leaving little room for alternative architectures. But in the industry, there is a product to develop and run web applications that follows a different architecture, centered on the RDBMS to the extreme of not needing any other component to function. There are not many papers in academia that addresses RDBMS-centric architectures in general, and this extreme architecture in particular has not been considered. In recent works I have analyzed the case of an extreme database-centric architecture, which I have called RDBMS-only architecture. This article defends the relevance and analyzes opportunity cases of this approach.

Keywords: RDBMS, Web Application Architecture, Database-Centric Architectures.

Resumen. - *Un aforismo famoso en ciencias de la computación dice: "Todos los problemas en ciencias de la computación pueden resolverse con otro nivel de indirección", a menudo ampliado con la cláusula humorística "excepto por el problema de tener demasiados niveles de indirección". Después de 30 años de aplicar el primer aforismo, las arquitecturas de varios niveles (es decir, arquitecturas con muchos niveles de indirección) se han convertido en el estándar de facto para las aplicaciones web, dejando poco espacio para arquitecturas alternativas. Pero en la industria existe un producto para desarrollar y ejecutar aplicaciones web que sigue una arquitectura diferente, centrada en el RDBMS al extremo de no necesitar ningún otro componente para funcionar. No hay muchos artículos académicos que aborden las arquitecturas centradas en RDBMS en general, y esta arquitectura extrema en particular no se ha considerado. En trabajos recientes he analizado el caso de una arquitectura extrema centrada en bases de datos, a la que he llamado arquitectura RDBMS-only. Este artículo defiende la pertinencia y analiza casos de oportunidad de este enfoque.*

Palabras clave: *RDBMS, Arquitectura de aplicaciones Web, Arquitecturas centradas en la Base de Datos.*

[1] Magíster en Informática. Facultad de Ingeniería, Universidad de la República (UDELAR), Uruguay, avicente@fing.edu.uy, ORCID iD: https://orcid.org/0000-0003-3575-5326

Resumo. **-** *Um famoso aforismo em ciência da computação diz: "Todos os problemas em ciência da computação podem ser resolvidos por outro nível de indireção", frequentemente expandido pela cláusula humorística "exceto pelo problema de muitos níveis de indireção". Após 30 anos aplicando o primeiro aforismo, as arquiteturas multicamadas (ou seja, arquiteturas com muitos níveis de indireção) tornaram-se o padrão de fato para aplicativos da Web, deixando pouco espaço para arquiteturas alternativas. Mas na indústria existe um produto para desenvolver e rodar aplicações web que segue uma arquitetura diferenciada, centrada no RDBMS ao extremo de não precisar de nenhum outro componente para funcionar. Não há muitos artigos acadêmicos que abordam arquiteturas centradas em RDBMS em geral, e essa arquitetura extrema em particular não foi considerada. Em trabalhos recentes, analisei o caso de uma arquitetura extrema centrada em banco de dados, que chamei de arquitetura RDBMS-only. Este artigo defende a relevância e analisa casos de oportunidade dessa abordagem.*

Palavras-chave: *RDBMS, arquitetura de aplicativos da Web, arquiteturas centradas em banco de dados.*

1. Introduction. -This article defends the hypothesis that a web application architecture physically contained in a Relational Database Management System (RDBMS), which has been called an RDBMS-only architecture, is a valid alternative, with advantages and disadvantages compared to the well-established multi-tier architectures.

It is important to clarify that this hypothesis is not inconsistent with the well-established theory that argues that multi-tier architectures constitute *an* adequate conceptual framework for the design of web applications, since they provide many important advantages. However, the hypothesis would be inconsistent with a theory that argues that multi-tier architectures are *the only* suitable conceptual framework for web application design, leaving no room for alternative architectures. Although this latter theory has not been explicitly affirmed, it has sometimes been implicitly assumed, and the little available literature that presents alternatives will also be discussed.

Furthermore, as this work summarizes previous publications, an article [1] and a master's thesis [2], its focus is to defend the relevance of the approach and analyze the opportunity to use it in certain use cases. It might appear that defending an argument implies bias. However, I believe this is not the case. From an empirical and historical point of view, multi-tier architectures have come to be seen as the only viable alternative. A whole generation of software engineers has been trained according to this idea, and there is almost no dissenting literature.

This article attempts to put the "extreme" database-centric architecture on an equal footing with the "extreme" middleware-centric architecture. The advantages of the latter have been extensively analyzed, but the advantages of the former have not.

Sections 2 and 3 provide a theoretical framework for understanding the validity of a database-centric architecture from conceptual and historical perspectives, respectively. Section 4 briefly presents the extreme database-centric (or RDBMS-only) architecture. Section 5 argues in defense of this architecture, explaining its advantages and the best possible cases of application. Finally, section 6 presents some conclusions and reflections.

2. Web application architectures from a conceptual point of view. - This section will discuss the relationship between layers, understood as portions of code with well-defined responsibilities, and tiers, understood as physical and/or technological components of an information system.

At a conceptual level, information systems are typically designed with three layers: 1) the *presentation layer*, 2) the application logic layer or *business logic layer*, and 3) the resource management layer or *data access layer*. The presentation layer is responsible for managing the display and collecting information from and to the end-user, usually through a Graphical User Interface (GUI). The business logic layer handles the processing related to the specific business application, while the data access layer executes database functions such as retrieval, addition, deletion, and modification of records [3].

In real systems, these conceptual layers can be combined and distributed across separate tiers, often involving different servers and specialized technologies. Many modern web applications employ a three-tier architecture, commonly known as 1) the client-tier, 2) the middle-tier, and 3) the data-tier. In most cases, there is a one-to-one correspondence between layers and tiers. Specifically, the business logic layer typically aligns with the middle-tier, which is positioned "in the middle" between the client and data tiers, constituting a physical middle-tier. This situation explains why the terms "business logic layer," "middle layer," and "middle tier" are sometimes used

interchangeably, and there are even specialized technologies referred to as "middleware". However, it is important to understand that the business logic layer is not necessarily limited to residing in the middle-tier.

How can the layers be distributed across tiers? Since layers are simply code components with specific responsibilities, it could be argued that the question is poorly formulated because the code comprising a single layer could be distributed across multiple tiers. However, for the purpose of modeling the problem, it is simplified by assuming that a layer is atomic.

Modern web applications must be accessible through web browsers without the need for plugins, a concept known as thin client. In other words, the majority of the code for the three layers must be distributed across two tiers: the middle-tier and the data-tier. According to the loft principle, at least two layers should coincide in a single tier. However, it is also possible for all three layers to coincide in one tier.

There are several theoretical approaches to analyzing the possible alternatives for organizing the architecture of an application in tiers. One of these approaches is Koppelaars' classification, which categorizes the eight theoretical combinations of three-tier systems, considering both a thin version ("little code") and a fat version ("lots of code") for each tier [4]. It's worth noting that in practice, there are only seven relevant combinations, as the thin/thin/thin case holds no genuine interest. A thin/fat/thin architecture (or Thick Middleware) implies that the code is predominantly in the middle-tier, whereas a thin/thin/fat architecture (or Thick Database) means that the code primarily resides in the data-tier[2].

According to Koppelaars' classification, the majority of modern web applications have a thin/fat/thin architecture (or Thick Middleware). However, Middleware is not the only tier that can be enriched with business logic. Web applications require thin clients, but the use of a thick middleware tier, while common, is neither necessary nor universal. There is an alternative approach to locate complex business logic: the data-tier. In this case, as Koppelaars points out, a thin/thin/fat combination is theoretically possible and, in some cases, may be advantageous. He refers to this case as the database-centric architecture. The terms database-backed, RDBMS-backed, database-centric, and database-driven are typically used interchangeably to describe web architectures where the RDBMS plays a vital role. In database-backed websites, content is dynamically generated by querying the database, and there is no application server [5].

Another way to interpret this classification is as the possible distributions of the layers across levels. Thus, a common distribution aligns each layer with a tier (presentation layer in the client-tier, business logic layer in the middle-tier, and data access layer in the data-tier). Using the notation: P for the presentation layer, BL for the business logic layer, and DA for the data access layer, this architecture can be denoted as the ordered triad (P, BL, DA), where the order corresponds to the client, middle, and data tiers. Following this nomenclature, and adopting the idea of using "thin" when there is little code, two possible distributions of a thin/fat/fat architecture could be (thin, P+BL, DA) and (thin, P, BL+DA). It is also common for the "lots of code" that implement the presentation, business logic, and data access layers to all be situated in the middle-tier (thin, P+BL+DA, thin). Conversely, although less frequent, it is also possible for the "lots of

[2]This classification, however, does not make it clear how the "lots of code" of each layer would be distributed in the middle and data tiers of a thin/fat/fat architecture. For this reason, an alternative that allows these distinctions to be made is presented below.

code" to be located in the data-tier, encompassing the presentation, business logic, and data access layers (thin, thin, P+BL+DA), as presented graphically in Figure I.

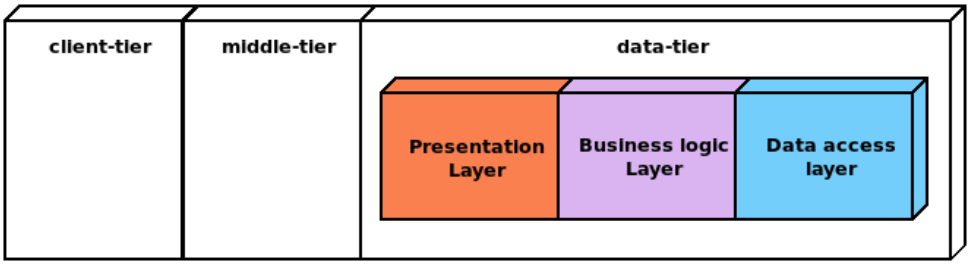

Figure I. A database-centric architecture

Thickening the data tier with presentation and especially with business logic code, could be possible when the RDBMS implements a Database Programming Language (DBPL). This architecture is feasible whether it is referred to as thin/thin/fat, (thin,thin,P+BL+DA), database-centric, or Thick Database. However, couldn't this database-centric architecture be more extreme? Without the need for middleware, the thin/zero/fat combination is possible. This alternative will be addressed in Section 4, following the historical discussion in Section 3.

3. Web application architectures from a historical point of view. - The decade of the nineties was marked by the advent of the Web, which made its debut in 1991. Tim Berners-Lee's invention had three fundamental components: URLs, which enabled access to the entire Web in a "flat" manner; the stateless HTTP protocol, which standardized communication; and the HTML language, which simplified the creation of web pages. This era also witnessed the development of graphical browsers such as Mosaic and Netscape. Netscape, in particular, introduced features like Secure Sockets Layer (SSL) for secure information transmission and cookies for tracking user information [6].

From the early days, there was a recognized need for creating database-backed websites, where content could be generated dynamically from information stored in a database. In 1993, the industry proposed a solution called the Common Gateway Interface protocol (CGI), which outlined how web servers could interact with external programs known as CGI scripts, and how these scripts could return dynamic pages to the web server [7].

Although it was technically feasible to develop a "web listener" that would allow the entire application to be implemented in a Database Programming Language (DBPL) as shown in Figure II, this approach was not pursued at the time due to the lack of mature DBPLs. Instead, Object-Oriented Programming (OOP) gained popularity during this period [8]. Object-Oriented Database Management Systems (OODBMS) were not widely adopted by the industry and did not reach maturity [9]. As a result, most web application architectures followed a typical pattern of having an object-oriented middle-tier and a relational data-tier.

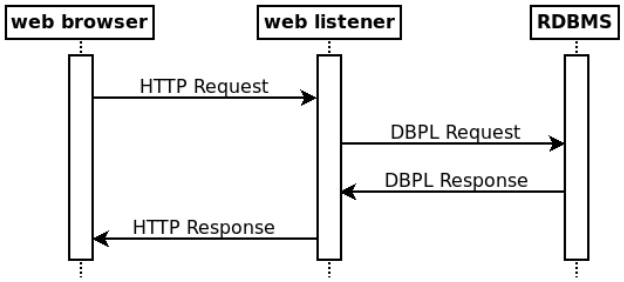

Figure II. The web listener of an RDBMS as a protocol adapter

This introduced a challenge: persisting objects in a relational format was not straightforward due to the object-relational impedance mismatch. To overcome this challenge, a dedicated component called Object-Relational Mapping (ORM) was introduced. These components have evolved over time to simplify data access and hide much of the complexity. In this scenario, the Thick Middleware architecture, where the middle-tier contains the bulk of the code, became the norm for web applications.

Oracle, in contrast to the prevailing trend, historically embraced a database-centric architecture with its products Forms and Application Express (APEX), leveraging its DBPL called PL/SQL [10, 11]. However, the feasibility of this approach does not depend on any particularity of the Oracle RDBMS; any RDBMS implementing a DBPL can serve as the foundation for an RDBMS-only architecture. This notion has been demonstrated in other works [1, 2].

APEX, which follows an extreme database-centric architecture, exemplifies the thin/zero/fat combination, or what I refer to as the RDBMS-only architecture, where an entire application resides within the RDBMS. In fact, even the web server can be contained within the RDBMS using the Embedded PL/SQL Gateway (EPG). Alternatively, the web listener component can be deployed as an Apache module or a Java application. Regardless of the deployment method, the focus of the architecture remains exclusively on the RDBMS. When Oracle describes the APEX architecture as "simple," it pertains to the physical view. However, it is important to note that while the physical view of the RDBMS-only architecture simplifies by consolidating all the complexity into a single component, it introduces increased logical complexity within that component, which raises design considerations.

An interesting historical question arises: why did Oracle choose to pursue a path that seemingly contradicted the prevailing trend, and why did others not follow suit? While definitive answers to these questions may be elusive, several hypotheses can be considered: a) the evolution of APEX was largely unplanned, and b) the APEX architecture went unnoticed by academia and the industry.

The first hypothesis can be justified by recognizing that APEX did not originate as a purpose-built solution for its current objectives. Instead, its conception was incremental. From the outset, the Oracle RDBMS was designed to handle HTTP requests, which laid the groundwork for the architecture, at least in practice. However, initial expectations regarding the capabilities of this technology were modest, limited to the potential use for developing reports and replacing spreadsheets. It was only over time and through experimentation that APEX's capabilities expanded to enable the development of stateful applications and the provision of an Integrated

Development Environment (IDE) built on the same technology. The evolution of APEX can be seen as a spontaneous order—an outcome of human action but not human design. In retrospect, the decision made by APEX architects proved to be fruitful, providing Oracle's clients with an alternative to building enterprise web applications without the need for additional technologies.

To justify the second hypothesis, it is helpful to distinguish between two types of novelty: true novelty, which entails genuinely new ideas, and novelty of arrangements, where the novelty arises from the combination of existing ideas. In this context, APEX represents a novelty of arrangements by combining preexisting capabilities of a DBPL within an RDBMS with the use of HTTP as a communication protocol. While this form of novelty of arrangements is indeed novel in practice, it does not rely on any published original idea. Additionally, compacting an entire enterprise application into a single physical tier appears to contradict the current trend of multi-tier separation, service-oriented architectures, and decoupled components running on containerized platforms. Furthermore, Oracle has not made any efforts to publish specific details about the APEX architecture at an academic level, limiting its description to aspects relevant to application administrators, developers, and its technical circles. Consequently, due to its status as a novelty of arrangements, its divergence from prevailing trends, and Oracle's limited interest in conducting and publishing research papers on the subject, it is understandable why the APEX architecture has gone unnoticed by academia and the wider industry.

4. The extreme database-centric architecture. - This section presents the proposed extreme database-centric architecture, which can be seen as a simple shift in emphasis from multi-tier architectures. Instead, a multi-layered architecture is proposed, but with these layers being logical layers within an RDBMS. The simplification is physical and technological, but the complexity does not disappear and is evident in the development and process views of the architecture [2].

Figure III illustrates the logical view of the architecture. On the right side are the layers of user applications. Each layer's code is only allowed to interact with objects and invoke code residing in the immediate lower layer. The Data Logic (DL), Business Logic (BL), and User Interface (UI) Code layers follow the Koppelaars classification. However, a new Service Interface (SI) Code layer is introduced at the same level as the UI Code layer. Additionally, the Interface Wrapper layer aims to separate the input and output of data from the specificities of both the graphical interface and web services. On the left side is an "engine" that does not adhere to the layered architecture of user applications. The engine is responsible, among other things, for executing the application flow.The innermost layer is not depicted in Figure III, as it represents the layer of the RDBMS itself, where database objects such as tables and indexes are located, and where the specific SQL dialect of the RDBMS executes SELECT, INSERT, UPDATE, and DELETE statements.

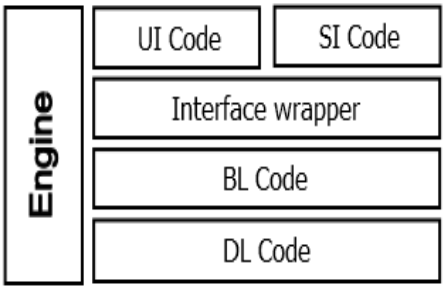

Figure III. Logical view of the extreme database-centric architecture

The DL Code layer encompasses stored procedures and/or functions that allow the use of SQL statements. This layer provides an interface to the upper layer for manipulating database relations as if they were an Abstract Data Type (ADT). It is worth noting that this layer can be automatically generated and kept synchronized with the database schema, and a catalog of existing functions can be maintained for use by the outer layer. It could also provide a standard RDBMS-independent interface if a standard DBPL existed. The absence of such a language at present does not imply that it cannot be developed in the future.

The BL Code layer represents the domain logic, which utilizes the functionalities provided by the previous layer. This is the most complex layer responsible for implementing use cases and ensuring transactional design. This layer can be further subdivided according to specific needs, either into more internal and external layers or into segments for cross-cutting functionalities. The interface of this layer to the outside provides high-level operations in a procedural format with RDBMS-specific data types.

In the Interface Wrapper layer, high-level operations are consumed and offered back to the upper layers, but with inputs and outputs in a standard format independent of both the RDBMS data types and the interface itself, such as Extensible Markup Language (XML) or JavaScript Object Notation (JSON).

Finally, the UI Code and SI Code layers include two different output formatting options: one designed for end clients through a web browser, and the other designed for web services. It is important to note that in this context, "interface" does not solely refer to graphical interface elements. This allows the treatment of domain layer functionalities in a symmetrical manner, whether it is a human user requiring an HTML interface or another system invoking a web service. As long as there are no interface elements in any of the underlying layers, maintaining the interface in these layers should be easier since it is confined to these layers. It should be noted that the architecture is a theoretical proposal, and variants may make sense depending on implementation decisions. For example, if it is decided to use PostgreSQL as the RDBMS and adopt a product like PostgREST to simplify the automatic provision of a REST API from the stored procedures of the BL Code layer, the Interface Wrapper layer could be eliminated.

In this analysis, the discussion of the specificities of the engine and the Integrated Development Environment (IDE) is beyond the scope of this document. However, perhaps I should mention one aspect of the interface design that Koppelaars and the APEX designers seem to have arrived at independently: the conception of a GUI as a network of pages, where each page contains a hierarchy of interface elements and navigation rules to other pages that are triggered by certain events. An essential element of this navigation is that the status of each session must be maintained in the database.

5. Defense of the extreme database-centric architecture. - In another work, I conducted a systematic comparison between two technologies using two versions of the same application. One version was implemented using the RDBMS-only architecture, while the other version utilized a middleware, with the ISO 25010 standard serving as the framework [2].

In this section, I aim to defend the extreme database-centric architecture itself, rather than advocating for a particular technology. I will identify the theoretical advantages it offers over a multi-tier architecture. Although this focus on advantages does not imply that the approach is without disadvantages, the software engineering community will readily and fully identify the

latter.

The most apparent savings can be observed in terms of technology. The RDBMS is the only required component, eliminating the need for a middleware technology. This can result in savings in computing and communication time in many cases, regardless of whether it leads to savings in licenses. Moreover, this technological simplicity reduces the number of components, thereby minimizing potential points of failure or vulnerability. These technology savings subsequently translate into process and personnel savings.

Process savings manifest in various forms, as multiple processes must be performed within each component of a solution. By reducing the number of components, there are fewer elements to monitor, update, back up, recover in the event of an incident, or integrate into configuration and change management processes. Many routine and exceptional processes are simplified by confining them to a single technology type.

Personnel savings primarily involve reducing the requirement for specific skills. Some companies or departments may possess individuals with advanced database skills but lack expertise in middleware technologies. In such cases, a database-centric approach can serve as an enabler. In the event of an incident, for instance, there is no need to determine the involved components initially. The troubleshooting process remains consistent, starting with the analysis of stored procedures and proceeding to the analysis of other stored procedures and eventually SQL statements.

However, it is important to note that this architecture may not be suitable for every case. For example, I am not suggesting its use for e-commerce sites with high write concurrency. Nonetheless, many systems do not have such demanding write concurrency requirements. Additionally, most RDBMSs offer relatively simple and transparent solutions to achieve high read scalability.

Therefore, the database-centric architecture could be applied in scenarios such as ticketing systems in small and medium-sized companies, in-house applications, or systems with low write but high read activity, like documentation systems or web logs. Furthermore, it can be an excellent alternative for converting desktop applications to web applications, especially when the existing architecture is already database-centric, and the business logic layer is implemented using a DBPL within the RDBMS.

Investing in the implementation of business logic in the DBPL of an RDBMS also proves to be a favorable option when presentation and middleware technologies are expected to change more frequently in the long term compared to RDBMS technologies. Moreover, the technological simplicity offered by this approach has the potential to reduce IT operating costs.

6. Conclusions. - The APEX case could represent an anomaly for following an architecture that does not meet expectations about how a web application *should* be structured. Those expectations are usually tacit, arising from continuous exposure to papers, textbooks and technologies that propose, describe and follow very different architectures.

When the web was invented, object orientation already existed. After the first experiences with technologies such as CGI, objects were a dominant trend in web applications, and along with objects at least three-tier architectures and middleware. These trends could be seen as a true

paradigm in the sense of Kuhn's scientific paradigm[3], where a new paradigm replaces the previous one [12]. The limitations of CGI could have been seen as evidence of a crisis of the "structured programming paradigm", and in that sense the new "object-oriented paradigm" was not simply an alternative but an overcoming. These ideas, explicitly or implicitly, may have dominated in both the academic and industry communities, and may perhaps explain the scarcity of works focused on database-centric architectures, and the uniqueness of APEX as an extreme database-centric architecture.

As far as I know, no one has postulated the following historical conjectures: 1) the trend of using object-oriented middleware was historically conditioned (I don't say determined) by the difference in maturity between OOP and DBPLs, 2) that trend was never explicitly challenged theoretically, even as DBPLs matured into a reasonable alternative, 3) Oracle's development of APEX was largely unplanned and is little known outside the APEX specialist community, and 4) no one has cautioned that APEX represents in practice a theoretically valid alternative.

While it is understandable historically why the alternative of an extreme database-centric architecture was not considered, conceptually it is a valid alternative, and there is at least one product in the industry that provides at least a glimpse of its possibilities. Moreover, there is no reason to think that Oracle has exploited all the possibilities of the architecture, or that the internal design of its product is the best possible.

In my master thesis I hope to have demonstrated that the development of a technology such as APEX does not depend on any particularity of the Oracle RDBMS. With any RDBMS that has a mature DBPL it is possible to build a similar technology, and the architecture of web applications developed with such technology have a number of inherent advantages that make them attractive for some use cases.

If I am correct in the aforementioned, it would be more appropriate to consider the extreme database-centric architecture not as a paradigm superseded in the sense of Kuhn, but rather as an underexplored yet progressive research programme in the sense of Lakatos[4] [13].

It would be reasonable to invest efforts in this alternative research programme to ascertain whether the theoretical development of the programme precedes empirical development that could yield numerous benefits. This is the main reason why, in this scenario, a defense seemed timely.

[3]Many have quoted Kuhn applying his idea of paradigms to technology, without realizing that Kuhn was referring to science. The guiding principle of technology is not truth, but utility, and for a "technical paradigm" to replace another, the latter should constitute an overcoming according to all possible criteria. To what extent one can speak of "technical paradigms" and what scope they might have, is a work that, I believe, remains to be done.

[4]Again, it should be noted that Lakatos speaks of scientific (and not technological) research programmes. It is far from obvious what, *mutatis mutandi*, can be preserved by moving from the realm of science to the realm of technology. In any case, I think the idea I am trying to convey is clear: if in the realm of science where (generally) truth is considered the guiding principle, it is reasonable to work in parallel on alternative research programmes, how much more reasonable should it be to do the same in the realm of technology where the main guiding principle is utility?

1. References

[1] Vicente, A., Etcheverry, L. and Sabiguero, A; An RDBMS-only architecture for web applications, *2021 XLVII Latin American Computing Conference (CLEI)*. IEEE, 2021.

[2] Vicente, A; La arquitectura RDBMS-only: una arquitectura database-centric para aplicaciones Web, Tesis de maestría. Universidad de la República (Uruguay). Facultad de Ingeniería, 2021. [Online]. Available: https://hdl.handle.net/20.500.12008/31620

[3] Scourias, J; Aspects of client/server database systems, University of Waterloo, 1995.

[4] Koppelaars, T; A Database-Centric Approach to J2EE Application Development, Oracle Development Tools Users Group (ODTUG), 2004.

[5] Greenspun, P; Database Backed Web Sites: The Thinking Person's Guide to Web Publishing. Ziff-Davis Publishing Co., 1997.

[6] Ceruzzi, P; Computing: a concise history, MIT press, 2012.

[7] T. A. S. Foundation; Rfc 3875 - the common gateway interface (cgi) version 1.1, 2004. [Online]. Available: https://tools.ietf.org/html/rfc3875

[8] Nielsen, J; Noncommand user interfaces, Communications of the ACM, vol. 36, no. 4, pp. 83–99, 1993.

[9] Kim, W; Object-Oriented Database Systems: Promises, Reality, and Future, in VLDB, vol. 19, 1993, pp. 676–692

[10] Cimolini, P; Oracle Application Express by Design: Managing Cost, Schedule, and Quality. Apress, 2017.

[11] Llewellyn, B; NoPlsql versus ThickDB, 2016. [Online]. Available: https://web.archive.org/web/20170909164923/https://blogs.oracle.com/plsql-and-ebr/noplsql-versus-thickdb

[12] Kuhn, T; *The structure of scientific revolutions*. University of Chicago press, 2012.

[13] Lakatos, I; Falsification and the methodology of scientific research programmes, in Lakatos I. and Musgrave A. Criticism and the growth of knowledge, Cambridge University Press, 1970

[14] Lakatos, I; History of science and its rational reconstructions, *PSA: Proceedings of the biennial meeting of the philosophy of science association*. Vol. 1970. D. Reidel Publishing, 1970.

Nota contribución de los autores:

1. Concepción y diseño del estudio
2. Adquisición de datos
3. Análisis de datos
4. Discusión de los resultados
5. Redacción del manuscrito
6. Aprobación de la versión final del manuscrito

AV ha contribuido en: 1, 2, 3, 4, 5 y 6.

Nota de aceptación: Este artículo fue aprobado por los editores de la revista Dr. Rafael Sotelo y Mag. Ing. Fernando A. Hernández Gobertti.