EasyChair Preprint

№ 6280

An RDBMS-Only Architecture for Web Applications

Alfonso Vicente, Lorena Etcheverry and Ariel Sabiguero

August 11, 2021

An RDBMS-only architecture for web applications

Alfonso Vicente*, Lorena Etcheverry†, Ariel Sabiguero‡
Instituto de Computación, Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
Email: *avicente@fing.edu.uy, †lorenae@fing.edu.uy, ‡asabigue@fing.edu.uy

***Abstract*—Multi-tier architectures have been the *de facto* standard for web applications, leaving little room for alternative solutions. Despite this, there is diversity in the proposals, especially in the tiers' number, size, and responsibilities. In particular, the database-centric approach aims to implement application logic and behavior within an RDBMS. In this work, we present, model, and propose to extend the database-centric approach into an RDBMS-only architecture, where the whole multi-tiered application is implemented in the database server. We present a characterization and description of the architecture and an early prototype that implements the proposal. It is important to note that both the database-centric and the proposed RDBMS-only architectures are a particular case of a three-layered model that needs to be differentiated from monolithic systems. Our preliminary results show that this approach is not only feasible but also advisable in some cases.**

***Index Terms*—Database systems, Relational databases, Systems architecture, Client-server systems**

I. Introduction

Multi-tier architectures have been the *de facto* standard for enterprise web applications for the last 30 years. They minimally include a *client-tier* and a *data-tier* but often include a *middle-tier*: an additional layer between them to support complex business logic [1]. This *middle-tier* is typically implemented in an application server [2].

Adding specialized tiers is often a way to deal with complexity, but the diversity of technology platforms implies an increase in IT operations and maintenance costs [3] [4] [5].

The three-tier approach provides several architectural advantages but requires interfacing overhead and coding expertise [6]. This architecture is well suited for contexts where the *middle-tier* introduces significant complexity to support business logic not directly bound to presentation or persistence.

To ease the interfacing overhead, relevant Java frameworks like Spring [7], collapse presentation and business layers into a single framework. Different industrial proposals address the need to ease tiering complexity, leading to simpler development processes, adequate for multiple development needs.

Relational Database Management Systems (RDBMS) have been the *de facto* solution for persistence, supporting almost every persistence layer. We claim that the RDBMS can be responsible for additional functionalities, like business logic implemented into stored procedures in many cases. Alongside the mainstream trend, there are proposals for database-centric architectures in which the RDBMS performs different functions in addition to persistence. This work addresses an enterprise architectural proposal that collapses all the layers in the RDBMS. We propose an RDBMS-only architecture, where the three classic web application layers are implemented on top of an RDBMS. In other words, an extreme database-centric architecture based almost exclusively on the capabilities and strengths of RDBMSs: the 50 years old relational model [8]–[12], the Structured Query Language (SQL) [13] and a Database Programming Language (DBPL) [14]. We also present a working prototype and some preliminary results that show that the proposed architecture poses some advantages in certain use cases.

The rest of this document is organized as follows. Section II presents a brief analysis of state of the art on database-centric architectures, while Section III presents the proposed architecture. In Section IV, we discuss some lessons learned during developing a prototype that follows this architecture. We describe the experiments in Section V based on an adaptation of the Transaction Processing Performance Council Web Commerce Benchmark (TPC-W) [15] and present preliminary results. In Section VI, we discuss known limitations and aspects of immediate research. Finally, in Section VII, we conclude and outline some future lines of work on this topic.

II. State of the art and Related work

At a conceptual level, information systems are designed around three layers: *presentation layer*, *application logic layer* or *business logic layer*, and *resource management layer* or *data access layer*. The presentation layer is responsible for managing the display and collecting information to and from the end-user, usually in a Graphical User Interface (GUI). The business logic layer performs the processing related to the particular business application, while the data access layer executes database functions like retrieval, addition, deletion, and modification of records [16].

These layers can be combined and distributed in separate tiers in developed systems, which usually means different servers. Many modern web applications use a three-tier architecture, in which the layers are commonly called *client-tier*, *middle-tier* and *data-tier* or *persistence-tier*. In most cases, the layers coincide with tiers. In particular, the business logic layer matches with a tier that is "in the middle", between the client and persistence tiers, constituting a middle-tier in the physical sense. This situation may explain why the terms "business logic layer," "middle layer," and "middle tier" are sometimes used as synonyms. It is worth mentioning that existing works contribute to clarify of these architectural concepts [6].

There has been some confusion over the term client/server, and it has sometimes been interpreted as synonymous for two-

tier architecture. To clarify the semantics, here we will use the term client/server as a distributed computing model where a client is a process that requests a service, and a server is a process that provides a service [16]. Conceptually, any web application follows the client/server model using HTTP as the service protocol, regardless of its architecture's physical tiers. A web application with persistence can be described as presented in Figure 1.

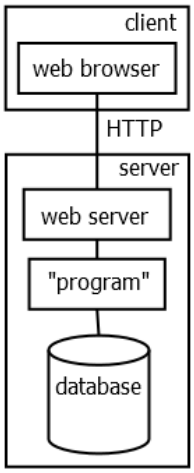

Fig. 1. Conceptual view of a web application with persistence

Web applications are accessed by a web browser, which uses the HTTP protocol to connect to a web server, which delivers the resources requested. The resources are typically web pages, with associated resources like images or code to be executed on the web browser. In this sense, a web application can be seen as a client/server application where the web browser is the client while the web server (along with all the resources it uses in the background) is the server. Enterprise web applications typically use a database managed by a DBMS, often an RDBMS, to implement the persistence tier.

Some "program" executes queries and manipulation statements between the web server, which knows how to deliver static content and the DBMS. This conceptual entity interacts both with the web server and the DBMS. However, from the logical point of view, it could reside in an independent component, be integrated into the web server or DBMS, or even be distributed among various components that may or may not include the web server and DBMS. The growing demands of building complex distributed applications maintaining a low coupling have led to designing applications using web services and middleware layers among databases and client browsers [17]. In this model, each layer uses the underlying layer services and provides more specialized services. An architecture called Distributed Object Computing (DOC) Middleware has also been proposed, a paradigm that supports flexible and adaptable behavior based on the aggregation of simple interactions through DOC Middleware layers [18].

To organize all the possible scenarios, Koppelaars performs a classification of the eight theoretical combinations of the three-tier systems (client-tier, middle-tier, data-tier) considering for each tier a thin version ("little code") and a fat version ("lots of code") [19]. It is worth noting that there are seven possible combinations in practice because the thin/thin/thin case has no genuine interest. A thin/fat/thin architecture means that the code is mainly in the middle tier, while a thin/thin/fat architecture means that the code is primarily in the data tier. Koppelaars also distinguishes between *User Interface* (UI) *Code*, *Business Logic* (BL) *Code*, and *Data Logic* (DL) *Code*.

According to the Koppelaars classification, most modern web applications have a thin/fat/thin architecture (or Thick Middleware as presented in Figure 2). However, Middleware is not the only tier that could be thickened by business logic. Web applications require thin clients, but the use of a thick tier of Middleware, while common, is neither necessary nor universal. There is an alternative to locate the complex business logic: the data-tier. In this case, and as Koppelaars points out, a thin/thin/fat combination is theoretically possible, and in some cases, it might be convenient. He refers to this case as the *database-centric* architecture.

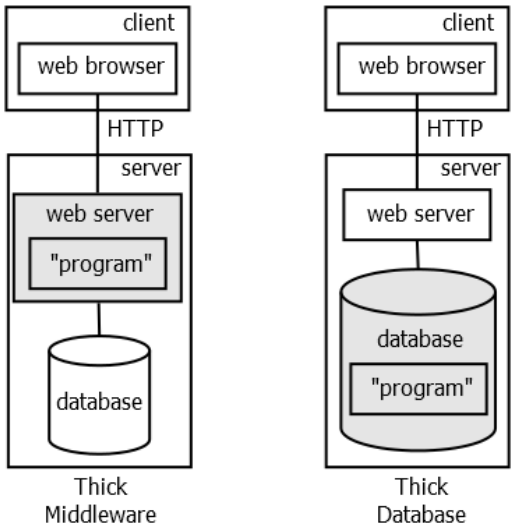

Fig. 2. A logical view of web Applications

In the literature, the terms *database-backed*, *RDBMS-backed*, *database-centric*, and *database-driven* are used to describe web architectures where the RDBMS plays a vital role. Although the surveyed works usually do not provide precise definitions of these terms, we sketch each approach in the following, focusing on the RDBMS responsibilities.

In *database-backed* web sites, the content is dynamically generated querying the database, and there is no application server [20].

In the *database-centric architecture*, the RDBMS plays a central role in the logic of applications [19], [21]–[25]. Simultaneously, the *database-driven* architecture extends the *database-centric* approach, and the data stored in the database is also used to determine the behavior of applications [26]–

[28]. Along with the central role of RDBMS in application logic, the *database-centric* approach also prescribes a methodology for the process of software development. This process begins designing the data model and transactions in an RDBMS, followed by a page flow design, and ending with the implementation of the individual pages [20], [29].

Thickening the data tier with business logic could be convenient if the RDBMS implements a DBPL. This architecture is possible whether we call it thin/thin/fat, database-centric, or Thick database. However, couldn't this database-centric architecture be more extreme? Without the need to use middleware, the thin/zero/fat combination is possible.

In the industry, Application Express (APEX) is an Oracle product that follows this extreme database-centric architecture, and Oracle promotes what it calls the Thick Database Paradigm, or Smart DB [30], [31]. APEX is an example of a thin/zero/fat combination or RDBMS-only architecture, where an entire application is contained in the RDBMS. Even the web server can be contained in the RDBMS if the embedded PL/SQL Gateway (EPG) is used. There are other alternatives to deploy what Oracle calls the *web listener*, an Apache module, or a Java application. Either way, this web listener is a technological component, and in terms of the 4+1 view model, the logical view of the architecture focuses exclusively on the RDBMS. When Oracle says that the APEX architecture is "simple", it refers to the physical view. However, the essential complexity of an application cannot be removed and can only be moved. Although the RDBMS-only architecture's physical view is simplified by bringing all the complexity to a single component, the logical complexity within this component increases, raising design concerns.

How APEX is designed is a valid question. Still, more generally (and because the RDBMS-only architecture is not a monopoly of a technology provider, but a little-explored possibility that could have advantages in certain use cases), a more interesting question is: how an RDBMS-only architecture can be defined? We attempt to answer this question in Section III.

But another valid question arises. Why has Oracle advanced so much in a line of work that seems to go against the general trend? Although we cannot answer this question, it is interesting to note some historical data on the development of web applications. One of the first approaches to generating dynamic web content was an industry proposal formalized in the Common Gateway Interface protocol, or CGI, in 1993 [32]. This protocol specifies how a web server can interact with external programs, called CGI scripts, as well as how in which these return the dynamic pages to the web server. The object-Oriented Programming (POO) paradigm boomed in the 90 when the web emerge [33]. Still, Object-Oriented Database Management Systems (OODBMS) were not mature and were sparsely adopted by the industry [34]. Proposals for web applications after CGI have generally followed the trend of incorporating object-orientation where it was most mature: in the general-purpose programming languages, not in databases. And the typical architecture has had an object-oriented middle-tier and a relational data-tier.

The object-relational impedance mismatch issue has been resolved with a component dedicated to implementing the Object-Relational Mapping (ORM), and these components have advanced to increase productivity by hiding much of the complexity of data access.

Oracle, at least with its products Forms and APEX, against the general trend, has historically adopted *database-centric* architectures that make intensive use of its DBPL called PL/SQL. However, the feasibility of walking this path does not seem to depend on any particularity of the Oracle RDBMS, and any RDBMS that implements a DBLP may be the basis of the RDBMS-only architecture.

III. The RDBMS-only architecture

For the architectural proposal, we set three design objectives: 1) Engine independence, 2) Thick Database Paradigm empowerment, and 3) Separation of concerns. Engine independence means the logical independence between the code that makes the application works (engine) and the user's applications. Thick Database Paradigm empowerment means the architecture will define a series of layers, each of which can only invoke operations from the immediately preceding layer. Furthermore, separation of concerns means that the architecture will prescribe the number of layers and their interests

Figure 3 shows the logical view of the architecture. On the right side are the layers of user applications. Each layer's code only has allowed to know objects and invoke code that resides in the layer immediately lower, and eventually, its 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 presented at the same level as the UI Code layer. Also, the *Interface Wrapper* layer aims to separate the input and output of data from the particularities of both the graphical interface and web services. On the left side is the engine, which does not follow the layered architecture of user applications.

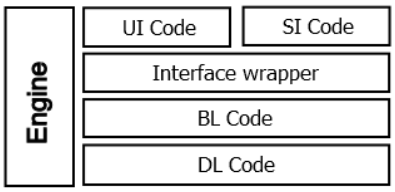

Fig. 3. A logical view of the RDBMS-only architecture

DL Code layer includes the DBPL code where SQL statements are allowed to be used. It offers the top layer an interface to manipulate the base tables as Abstract Data Types (ADT). This layer can be generated and synchronized with the database schema automatically, and a catalog of existing functions could be kept to be used by the top layer. It could also offer a separate standard interface of the RDBMS if there were a standard DBPL, and the fact that such language does

not exist today does not mean that it cannot be developed in the future.

BL Code layer consumes the DL Code layer's operations and contains the logic of the domain. This layer could be subdivided, either in upper and lower sublayers or in segments for cross functionalities, such as reports or authentication and authorization systems. This layer's interface offers high-level operations in procedural format with the specific data types of the RDBMS.

The **Interface Wrapper** layer consumes the BL Code layer's high-level operations. It offers them to the upper layers, but with inputs and outputs in a standard text-based format independent of the RDBMS data types and the interface, such as XML or JSON.

Finally, in the **UI and SI Code layers**, two different output format options are included. One is designed for end customers through a web browser, and another is planned for Web Services. As long as there are no interface elements in any underlying layers, maintaining this interface should be relatively straightforward.

It should be noted that the UI Code layer should only be sufficient to offer interface elements, but this is not enough to generate the browsing experience in a web application. For that, we must have a navigation model. This model could be based on Koppelaars' original idea of an application such as a page network: a directed graph where the nodes are the pages and the edges the valid navigations between them [19].

The application's behavior, including how dynamic pages are generated, the internal structure of the pages, and navigation, are defined in the engine. The general architecture does not prescribe a data model for an application beyond the hierarchy of the entities: application, page, region, item. Our proposal also determines a development methodology. It starts designing the data in the database and continues through the design of the page flows. For example, in a bookstore e-commerce application, entities such as book and author should be modeled, resulting in the logical model of tables. There should be support to generate the DL Code automatically. Besides the structure of the pages, the flow between them should be designed. For example, to go from a "Search Results"page to a "Product Details" page, a book ID must be passed, and a homogeneous mechanism must be provided to perform this passage. These variables must be kept associated with each session, and there may be session variables at the application level or the page level.

In Section II, we mentioned that the web listener is a technological component without much relevance for the logical view of the architecture. Figure 4 shows the web listener function of translating an HTTP request into a DBPL request, and back, translating a DBPL response into an HTTP response. The architecture prescribes to support at least GET and POST methods. Each request is the request of a page, and each response is an HTML page, eventually with other resources like images, CSS and JavaScript. The web listener is entirely stateless, and even the session management is done in the RDBMS. Due to this, there can be multiple web listeners serving HTTP requests for the same RDBMS.

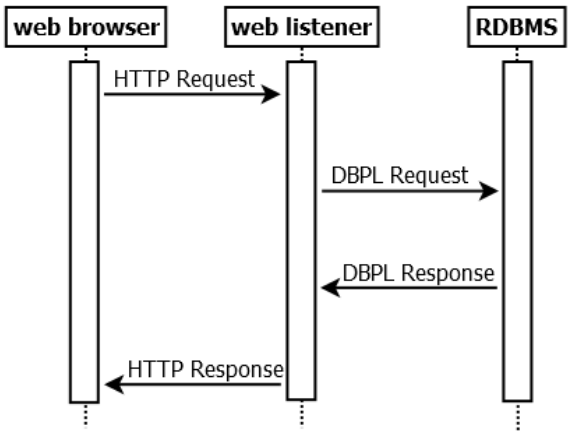

Fig. 4. The web listener as a HTTP / DBPL translator

There are common objections to database-centric architectures that discuss the ability to scale and provide high availability, both related to a supposed monolithic architecture structure of that "same RDBMS". There is an old but current debate about the TP-heavy versus TP-lite debate [35]. We will not enter into that debate because scalability and high availability issues within an RDBMS can be addressed and entirely improved independently and transparently from this proposal. We will only note that there are real-world cases where scalability and high availability were central problems in which a solution based on an RDBMS was chosen [36]–[38]. Also, in recent years much progress has been made in strategies to provide scalability, high availability and extreme performance in both relational and non-relational DBMSs [39]–[44].

In any case, we are not arguing that this proposal is compatible with the most scalable and fault-tolerant architecture of the possible ones. There is a niche for architectures and technologies that don't need extreme scalability and high availability requirements, after all [45]. The core of this work consists in showing the feasibility of an RDBMS-only architecture and technology applicable to any RDBMS with a powerful DBPL. In others words, we question the "one size fits all" trend that seems to rule today's web development practices.

IV. Prototype implementation

A prototype called **webpg** was developed with PostgreSQL as RDBMS and an Apache HTTP Server module called `mod_plpgsql` as a web listener, with the ability to run arbitrary PL/pgSQL code[1]. We intentionally avoided Oracle as RDBMS to check if the feasibility of developing a product like APEX depended on some inherent features of this RDBMS. PostgreSQL was used because it is a Free and Open Source product with a mature and powerfull DBPL. Some ideas were

[1]Source code available at https://gitlab.fing.edu.uy/lorenae/rdbms-only

taken from APEX; nevertheless, in many ways, our prototype differs from it.

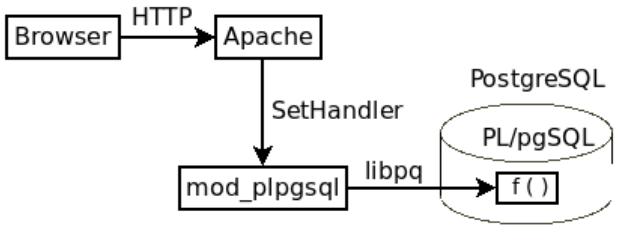

Fig. 5. Physical view of the RDBMS-only architecture

Figure 5 presents the physical view of the architecture. An important design decision is the syntax of the input parameters of HTTP Requests, both in the case of GET and POST. Following APEX, we decided not to pass any parameters with business logic semantics in the requests. This decision has the disadvantage that any navigation that must pass parameters with semantics must use the POST method. However, it offers a homogeneous and straightforward way to process requests while constituting a safe design avoiding the possibility of injections in requests.

Any request like `http://host[:port]/plpgsql/...` that arrives at the host and port where the Apache HTTP serves activates the `mod_plpgsql` module's handler and its code will process the request. The URL of a Request when the GET method is invoked follows the syntax `plpgsql/conf?g&p=app[:page[:ses]]`. The URL of a Request when the POST method is invoked follows the syntax `plpgsql/conf?p`, where the parameters should be passed as a JSON map of the form:

```
{
  "app_id":"app",
  "page_id":"page",
  "session_id":"session",
  ...
}
```

The mod_plsql validates the existence of a configuration called conf in a special file, which must have the connection parameters for the database, and through libpq it executes `select * from g('p=app:page:sess')`. The function g() must exist in the database and must be in charge of processing the GET, just as there must be a function p() in charge of processing the POST. The syntax is arbitrary, but it must be defined *a priori*. When an application is developed, navigation can be implemented following the syntax, using links or HTML Forms, as shown in Figure 6. It is worth noting that other parameters with business semantics such as book_id can be passed.

In the case of the GET, both the page and the session are optional. If a page is not specified, the application assumes that a default application page is requested, and if a session is not specified, the application assumes that it is of the first request of a user.

```
<form action="webpg/p" method="post">
  <input type="hidden" name="app_id" value="1">
  <input type="hidden" name="page_id" value="9">
  <input type="hidden" name="session_id" value="10000012">
  <input type="hidden" name="book_id" value="2459">
  <input type="IMAGE" name="thumb_2459.gif"
         src="_rf/2Q=="
</form>

<a href="webpg?g&p=1:1:10000012">
  <img id="home.gif"
       src="_AAOw=="
</a>
```

Fig. 6. Navigation code example

Figure 7 presents a logical view of the engine. It is relevant to appreciate the basic hierarchy: application, page, region, item since an application will be a set of tuples that will populate the tables derived from this Entity-Relationship (ER) Model.

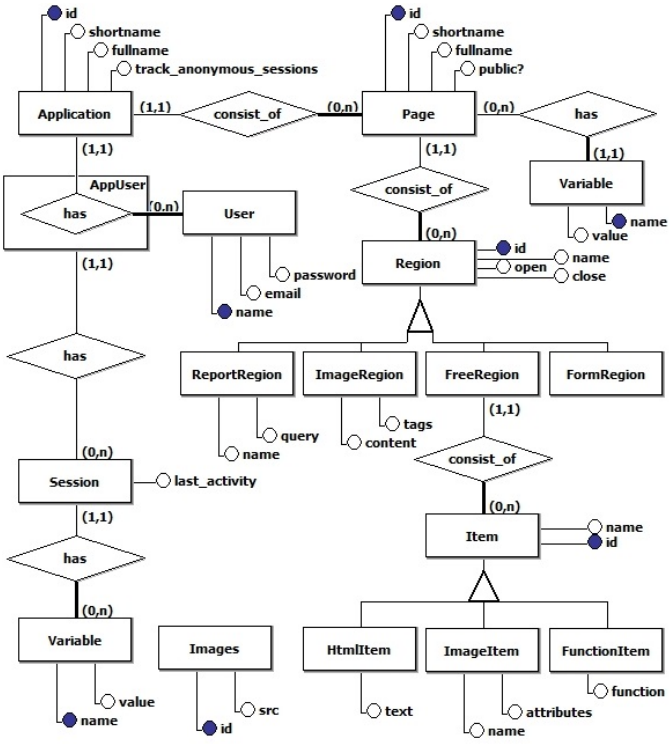

Fig. 7. Logical view of the engine

The central part of the prototype consists of developing a function g() to attend HTTP requests with GET method; and a function p() to attend HTTP requests with POST method. In either case, it should be noted that the fundamental problem is getting a page from an application beyond the technological difference. For this reason, there is a generic getpage() function, which could be used for both GET and POST, as well as intermediate functions process_get() and process_post() to solve particular GET and POST problems,

respectively. Figure 8 presents an excerpt from a process view of the architecture, showing the relationship between the main prototype functions.

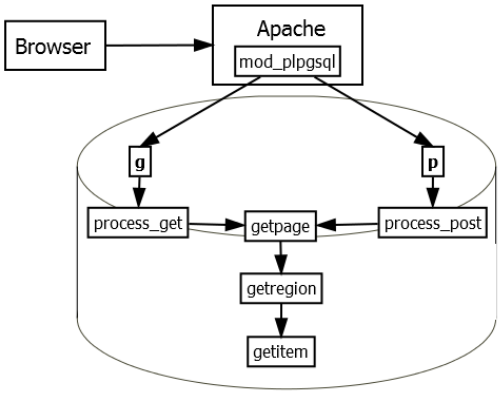

Fig. 8. Main functions of the prototype

The getRegion() and getItem() functions in Figure 8 retrieve varchar values stored in tuples (which can correspond to functions of the UI Code in Figure 3) and are executed dynamically. Figure 9 shows examples of function items that are executed dynamically, using page variables (see Figure 7). The value of a page variable can be a function like get_session_var(), the string "###S###" is replaced at runtime by the session ID, and the get_session_var() function returns the value of a page variable called book_id on page 9 (Product Detail) for that session. Previously, when navigating from another page to page 9, this session variable had to be set in the POST. The model of Figure 7 supports variables at the page level and at the session level. When page 9 is generated for that session, the function items in Figure 9 are instantiated with the values of the title, author and subject of the selected book.

```
webpg=> select item_id as i, function from function_items
webpg-> where application_id = 1
webpg-> and page_id = 9
webpg-> and region_id = 2;
 i |                              function
---+-----------------------------------------------------------
 2 | get_book_title(get_session_var('###S###', 9, 'book_id'))
 4 | get_book_author(get_session_var('###S###', 9, 'book_id'))
 6 | get_book_subject(get_session_var('###S###', 9, 'book_id'))
(3 rows)
```

Fig. 9. Function items for dynamic execution

The development process of the prototype involved two elements. The first one is the development of the database code and structures, focused on the application domain. It includes conceptual modeling, logical modeling, physical implementation, execution of automatic functions that generate the DL Code layer, development of business logic functions at the BL Code layer, and the Interface Wrapper and UI Code functions. The second part involves developing the application pages, which is reduced to populating most of the structure presented in Figure 7.

The prototype development process left several lessons learned. For example, we discovered that it is impossible to navigate and pass parameters with semantics using GET. A workaround is to use the POST method, with a relative increase in the complexity of the HTML code. However, the most important lesson is that we successfully implemented a complete use case described in the following section.

V. Early results

This Section presents an experiment consisting of developing a simple e-commerce application using our prototype from Section IV. The system chosen was TPC-W, a well-known benchmark for e-commerce systems that specifies all the relevant details of an application [15], [46]. A TPC-W implementation with Java Servlets is also available [47], [48]. There are at least two good reasons to develop an equivalent implementation of this application using our prototype.The first is that it is possible to validate if the prototype's implemented application achieves identical interfaces and behavior. The second is that it is possible to benchmark both applications according to quality characteristics, especially performance, since TPC-W prescribes performance metrics.

TPC-W web application consists of 14 pages with various navigation sequences among them. The ER model comprises eight tables where both the schema and each table's data volume are specified. The user interface is also described in the TPC-W specification, offering an HTML code sample for each of the pages.

Development began by defining the most static pages, such as the *"Home"*. Despite being an almost static page, its development presented two different challenges. First, the impossibility of implementing some links with arbitrary parameters as <a> tags. Second, obtaining five books to display their thumbnails, according to a specific random procedure defined in the benchmark's *promotional processing* section.

Once the development of this small application was completed, the feasibility of implementing applications with this architecture was empirically shown. Moreover, it also enables performance comparison between our implementation and Wisconsin's Java Servlet-based implementation.

Figure 10 shows what the two implementations share and how they differ. For this comparison, we use the same database with the same schema and data volume. The Java Servlet-based implementation accesses the database through Java Database Connectivity (JDBC) and fetches filesystem images. The RDBMS-only implementation uses the mod_plpgsql module as an adapter between the HTTP and libpq protocols. However, it processes all requests in the database, including the images that are stored in tables.

TPC-W performance tests prescribe three types of usage scenarios on the e-commerce site: Shopping, Browsing and Ordering. For each of these scenarios, it establishes the Requests' frequency to each of the 14 pages of the site. The most relevant performance measure is the Web Interaction Response Time (WIRT), which is the time that passes from when the

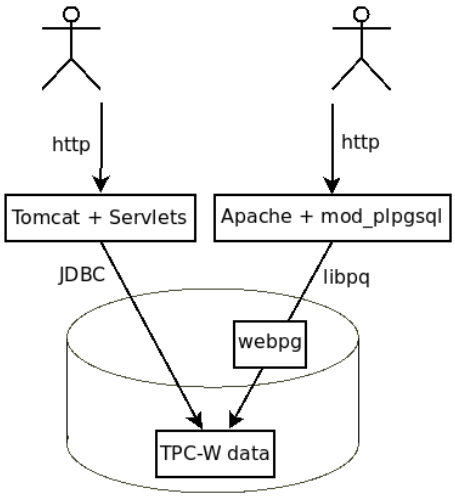

Fig. 10. Implementation comparison

first byte is sent from the browser until the browser receives the last byte.

Figure 11 represents the results of a performance comparison using the WIRT measure. The frequencies of loading times for two of the most accessed pages were compared during Shopping Mix interactions. The results correspond to 10 Emulated Browsers (EB) running concurrently for one hour, and a catch & reply strategy was used. Therefore the sessions performed the same navigations on the two systems.

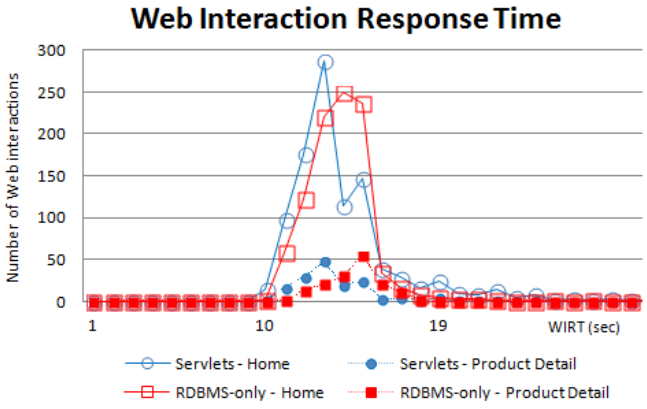

Fig. 11. Performance comparison

Figure 11 shows that there are no significant performance differences between the two implementations. There are also no significant differences from this comparison with the other pages and the other types of Mix interactions. A preliminary observation is that the Java Servlets implementation has a higher frequency of slightly better response times but has higher dispersion than the RDBMS-only implementation. Our prototype seems to have slightly slower response times but is more stable in observed response times. It is worth noting that no database optimization was done.

This result must be interpreted in the light of the objectives: we are not trying to show that our RDBMS-only prototype would have better performance than an alternative based on Java Servlets, but we are only trying to show that it does not have to have worse performance. For more comprehensive performance analysis, the entire prototype code should be optimized. It should be borne in mind that beyond performance, one of the objectives of implementing an alternative to TPC-W was to validate the possibility of developing a small working application.

VI. Open issues and known pain points

Some aspects already identified that are still not addressed must be addressed soon. A key aspect is version control and change management. Modern software delivery methodologies and team collaboration require heavy use of source code versioning. As application logic inside an RDBMS-only system will be stored as DBPL, versioning should be addressed with DBPL versioning tools. Applicability of database schema and data versioning tools like liquibase [49], DbPatch [50], Flyway [51], Evolve [52] and gitSQL [53] must be analyzed. These tools proved adequate for full database versioning in real application scenarios. We believe that they should be specialized for the RDBMS-only requirements.

Another relevant problem is the availability of general-purpose libraries. Although an RDBMS-only technology can produce and consume web services, there are difficulties interoperating and performing specific tasks due to the absence of libraries that solve everyday problems. If we sacrifice some of the design principles to certain levels, particularly Engine Independence, we can select general-purpose DBPL extensions. Our prototype is built on top of PostgreSQL [54], which provides PL/Tcl, PL/Perl and PL/Python procedural languages in the core distribution, in addition to PL/pgSQL [55]. There are additional procedural languages maintained by projects external to the PostgreSQL Global Development Group (PGDG) such as PL/Java, PL/Lua, PL/R, PL/sh and PL/v8 [56]. Beyond PostgreSQL, this diversity of extensions is not common. No similar extension is available, for example, in MariaDB [57]. Just in recent versions of MariaDB, a subset of Oracle's PL/SQL language has been supported in addition to the traditional SQL/PSM-based MariaDB syntax [58]. In Oracle, Java stored procedures are available [59], but Java code stored in one RDBMS is not transportable to another, at least without an abstraction layer of the RDBMS. Given the importance of standard access to libraries, it could also be considered to analyze feasibility of an RDBMS independent extension that would allow uniform access to general-purpose libraries. Detailed analysis and implementation proposals are beyond the scope of this work.

Other development tools like IDEs and debuggers are a must too in order to produce an industrial-grade solution. Despite the analysis done and the confidence in finding a solution, all these aspects are work in progress and must be considered research subjects.

VII. Conclusions and future work

Existing approaches to ease multilayering in web development often neglect the data tier. As far as we know, extreme database-centric architectures have not been previously addressed or described in the literature. However, the industry offers products like Oracle APEX, and Oracle claims several success stories in several clients [60].

This work proposes RDBMS-only architecture, describes it, and suggests future lines of research. It aims at showing its existence and taking a position on its convenience in specific scenarios. It seems an excellent alternative to convert desktop applications to web applications when the architecture is already database-centric, and the business logic layer is implemented in the RDBMS using a DBPL. Investing in the implementation of business logic in the DBPL of an RDBMS also seems a good option in cases where presentation and middleware technologies are expected to change more frequently in the long term than RDBMS technologies. Besides, technological simplicity could reduce IT operating costs. Despite the promising results of the prototype, the architecture has some aspects that require immediate attention, as discussed in Section VI. To reach a certain maturity, version control and change management need to be addressed. The goal of RDMBS independence introduces limitations on the availability of general purpose libraries that solve common problems. This difficulty is accidental and is explained by the low popularity of this type of architecture. In addition, the maintainability of an RDBMS-only application depends on the functionalities offered by the Integrated Development Environment (IDE).

An in-depth analysis of the internal architecture of APEX can help to refine the essence of the type of architecture that emerges from the specimen specifics, trying to discern what is necessary and what is contingent.

A second exciting line of research, outside the scope of this paper, is the development of the SI code layer to use the RDBMS-only architecture to provide web services. In this case, without all the complexity of GUI elements, it may be even easier to provide data services with an RDBMS-only approach.

The third line of research could be the development of an IDE that allows the development of applications in a productive way.

Finally, progress could be made in the development of such prototypes and their analysis as a software product using a framework such as ISO 25010 [61].

References

[1] W. W. Eckerson, "Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications." *Open Information Systems*, vol. 10, no. 1, 1995.

[2] X. Liu, J. Heo, L. Sha, and X. Zhu, "Adaptive control of multi-tiered web applications using queueing predictor," in *NOMS*. IEEE, 2006, pp. 106–114.

[3] M. Mocker, "What is complex about 273 applications? untangling application architecture complexity in a case of european investment banking," in *2009 42nd Hawaii International Conference on System Sciences*, 2009, pp. 1–14.

[4] S. Scantlebury, W. Thiel, A. Datel, and S. Kimmel, "From it complexity to commonality: Making your business more nimble," *Opportunities for Action in Information Technology*, 2004.

[5] P. Child, R. Diederichs, F.-H. Sanders, and S. Wisniowski, "Smr forum: the management of complexity," *MIT Sloan Management Review*, vol. 33, no. 1, p. 73, 1991.

[6] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, "Web services," in *Web Services*. Springer, 2004, pp. 123–149.

[7] "Spring Framework." [Online]. Available: https://spring.io/

[8] E. Frank Codd, "Derivability, Redundancy and Consistency of Relations Stored in Large Data Banks," IBM Research Report RJ 599, Tech. Rep., Aug. 1969.

[9] ——, "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, Jun. 1970.

[10] ——, "A Data Base Sublanguage Founded on the Relational Calculus," in *Proceedings of the 1971 ACM SIGFIDET (now SIGMOD) workshop on data description, access and control*, Jul. 1971, pp. 35–68.

[11] ——, "Further Normalization of the Data Base Relational Model," *Data Base Systems: Courant Computer Science Symposia Series 6*, vol. 6, pp. 33–64, 1972.

[12] E. Frank Codd *et al.*, *Relational Completeness of Data Base Sublanguages*. IBM Corporation, Mar. 1972.

[13] D. D. Chamberlin and R. F. Boyce, "Sequel: A structured english query language," in *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control*, 1974, pp. 249–264.

[14] J. Schmidt, H. Eckhardt, and F. Matthes, "Dbpl report," 1988.

[15] T. P. P. C. (TPC), "Tpc benchmark w (web commerce) specification," Dec. 2003. [Online]. Available: http://www.tpc.org/tpc_documents_current_versions/pdf/tpcw_v2.0.0.pdf

[16] J. Scourias, "Aspects of client/server database systems," 1995.

[17] P. A. Bernstein, "Middleware: a model for distributed system services," *Communications of the ACM*, vol. 39, no. 2, pp. 86–98, 1996.

[18] R. E. Schantz and D. C. Schmidt, "Middleware for distributed systems: Evolving the common structure for network-centric applications," *Encyclopedia of Software Engineering*, vol. 1, pp. 1–9, 2001.

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

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

[21] J. C. Shafer and R. Agrawal, "Continuous querying in database-centric web applications," *Computer networks*, vol. 33, no. 1, pp. 519–531, 2000.

[22] J. Ploski, W. Hasselbring, J. Rehwinkel, and S. Schwierz, "Introducing version control to database-centric applications in a small enterprise," *IEEE software*, vol. 24, no. 1, 2007.

[23] S. M. Krishna, S. Karnati, A. Biswas, and J. Srinivasan, "Analysis and Modeling of Evolving Database-centric Web Applications." in *COMAD*, 2010, p. 65.

[24] T. Chen, "Improving the quality of large-scale database-centric software systems by analyzing database access code," in *2015 31st ICDE Workshops*, 2015, pp. 245–249.

[25] M. Linares-Vásquez, B. Li, C. Vendome, and D. Poshyvanyk, "Documenting database usages and schema constraints in database-centric applications," in *Proc. of the 25th International Symposium on Software Testing and Analysis*, 2016, pp. 270–281.

[26] S. Ceri, P. Fraternali, and S. Paraboschi, "Data-driven, one-to-one web site generation for data-intensive applications," in *VLDB*, vol. 99, 1999, pp. 7–10.

[27] K. S. Candan, W.-S. Li, Q. Luo, W.-P. Hsiung, and D. Agrawal, "Enabling Dynamic Content Caching for Database-driven Web Sites," in *SIGMOD*. New York, NY, USA: ACM, 2001, pp. 532–543.

[28] M. Muji, "Application Development in Database-Driven Information Systems." *Acta Universitatis Sapientiae-Electrical & Mechanical Engineering*, vol. 2, 2010.

[29] Eve Andersson, Philip Greenspun, and Andrew Grumet, *Software Engineering for Internet Applications*, 2006. [Online]. Available: http://philip.greenspun.com/seia

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

[31] B. Llewellyn, "NoPlsql versus ThickDB." [Online]. Available: https://blogs.oracle.com/plsql-and-ebr/noplsql-versus-thickdb

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

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

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

[35] J. Gray, "Why tp-lite will dominate the tp market." in *HPTS*, 1993, p. 0.

[36] J. Shute, M. Oancea, S. Ellner, B. Handy, E. Rollins, B. Samwel, R. Vingralek, C. Whipkey, X. Chen, B. Jegerlehner *et al.*, "F1-the fault-tolerant distributed rdbms supporting google's ad business," 2012.

[37] J. Shute, R. Vingralek, B. Samwel, B. Handy, C. Whipkey, E. Rollins, M. Oancea, K. Littlefield, D. Menestrina, S. Ellner *et al.*, "F1: A distributed sql database that scales," 2013.

[38] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild *et al.*, "Spanner: Google's globally distributed database," *ACM Transactions on Computer Systems (TOCS)*, vol. 31, no. 3, pp. 1–22, 2013.

[39] M. Stonebraker and R. Cattell, "10 rules for scalable performance in'simple operation'datastores," *Communications of the ACM*, vol. 54, no. 6, pp. 72–80, 2011.

[40] R. Cattell, "Scalable sql and nosql data stores," *Acm Sigmod Record*, vol. 39, no. 4, pp. 12–27, 2011.

[41] I. Rae, E. Rollins, J. Shute, S. Sodhi, and R. Vingralek, "Online, asynchronous schema change in f1," *Proceedings of the VLDB Endowment*, vol. 6, no. 11, pp. 1045–1056, 2013.

[42] S. Loesing, M. Pilman, T. Etter, and D. Kossmann, "On the design and scalability of distributed shared-data databases," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015, pp. 663–676.

[43] S. Harizopoulos, D. J. Abadi, S. Madden, and M. Stonebraker, "Oltp through the looking glass, and what we found there," in *Making Databases Work: the Pragmatic Wisdom of Michael Stonebraker*, 2018, pp. 409–439.

[44] G. A. Schreiner, R. Knob, D. Duarte, P. Vilain, and R. d. S. Mello, "Newsql through the looking glass," in *Proceedings of the 21st International Conference on Information Integration and Web-based Applications & Services*, 2019, pp. 361–369.

[45] C. Nance, T. Losser, R. Iype, and G. Harmon, "Nosql vs rdbms-why there is room for both," 2013.

[46] D. A. Menascé, *TPC-W: A Benchmark for E-commerce*, 2002.

[47] H. W. Cain and R. Rajwar, "An architectural evaluation of Java TPC-W," in *Proc of the 7th Int. Symposium on HPC Architecture*, 2001, pp. 229–240.

[48] T. Bezenek, T. Cain, R. Dickson, T. Heil, M. Martin, C. McCurdy, R. Rajwar, E. Weglarz, C. Zilles, and M. Lipasti, *Java TPC-W implementation distribution*, 2011.

[49] "Liquibase." [Online]. Available: https://www.liquibase.org/

[50] "DbPatch." [Online]. Available: https://github.com/dbpatch/DbPatch

[51] "Flyway." [Online]. Available: https://flywaydb.org/

[52] "Evolve." [Online]. Available: https://evolve-db.netlify.app/

[53] "gitSQL." [Online]. Available: https://gitsql.net/

[54] "Postgresql." [Online]. Available: https://www.postgresql.org

[55] "PostgreSQL: Documentation: 13: Chapter 41. Procedural Languages." [Online]. Available: https://www.postgresql.org/docs/current/xplang.html

[56] "PostgreSQL: Documentation: 13: Chapter h.3. External Projects - Procedural Languages." [Online]. Available: https://www.postgresql.org/docs/13/external-pl.html

[57] "Mariadb enterprise open source database." [Online]. Available: https://mariadb.com

[58] "SQL_MODE ORACLE - MariaDB Knowledge Base." [Online]. Available: https://mariadb.com/kb/en/sql_modeoracle/

[59] "Java Programming in Oracle Database." [Online]. Available: https://docs.oracle.com/en/database/oracle/oracle-database/21/jjdev/Java-application-strategy.html

[60] "Oracle success stories." [Online]. Available: https://apex.oracle.com/en/solutions/success-stories/

[61] ISO/IEC, "ISO/IEC 25010 System and software quality models," ISO/IEC, Tech. Rep., 2010.