Introduction

This session is focused on existing sometimes called legacy database-centric applications. By that I mean applications that store data that may include forms, formatted reports, queries, programs, etc. Data may be currently stored in relational, no-sql, flat files, indexed files (ISAM), Spreadsheets or non-relational databases.

It is up to stakeholders to determine if these applications can be of continuing value to the organizations. That said, if these database-centric applications are considered valuable enough for continued use the primary goal of the tools and techniques presented in this paper is to preserve the value in these applications and to ensure that they continue to evolve with business needs.

What users want, need and expect from an application has evolved over time. Our process of ModernizationInPlace™ is used to keep existing applications relevant and preserve as much as possible from the application while modernizing it to meet evolving business needs. Figure 2 graphically describes how applications have evolved from dumb terminals to graphical user interfaces and then to internet and cloud connected and web and mobile accessed multi-dimensional useful entities.

Approach - ModernizationInPlace™

We use an approach that we call ModernizationInPlace™. This allows a stepwise enhancement of the features available to application users with minimal risk, training and business disruption. The approach allows application stakeholders to use a combination of existing application components and new components at the same time. Optionally, at a time of the application owners choosing, all application components can be replaced. However, replacement is a choice and not a requirement.
ModernizationInPlace™ follows the following stages shown in Figure 3:

1. The database (data source) is separated from the other application components if necessary.
2. Next the data source with PostgreSQL and using the existing database forms, reports, etc. with the PostgreSQL database. The database is replaced by PostgreSQL and TransformixTools adapters. This allows application users to continue accessing the application in the familiar way.
3. Next, we use the CUBA-Platform web application development framework to provide an alternate route to access and modernize the application. The CUBA-Platform extends the options for accessing the data and functionality of the application using web browsers, mobile clients, REST and HTTP APIs.
4. Optionally, the original application components are retired and only the CUBA-Platform application is used.

**Milestones**

There are really two major application inflexion points or significant milestones in this approach.

1. PostgreSQL Database replaces existing data source.
2. CUBA-Platform Web Application Development Framework used to build replacement application artifacts.
PostgreSQL Database replaces existing data source
At this point an adapter is introduced that makes the existing application think it is still accessing the original ISAM, Dbase, TurboIMAGE, etc. data source when in fact the data source has been replaced by PostgreSQL and an adapter (interface layer, API). The original application works just like it always has. However, other access to the data has been greatly expanded. This means that standard commercially available off the shelf tools that used SQL can now access this formally difficult to get at data. Also, when a REST API is added to the database, a whole new range of tools can be used to access the data. Moreover, if extending the application to the cloud is desirable. That too is now easier to achieve.

CUBA-Platform Web Application Development Framework used to build replacement application artifacts
Another milestone is reached with the introduction of the CUBA-Platform in stage 3. At this point, the PostgreSQL database allows for rapid development of new application forms, reports, queries, etc. It also automatically creates REST APIs for the database tables and views.

Why PostgreSQL?
The second stage of our approach is to replace the database in an existing application with PostgreSQL. In many cases the original data store is a proprietary or non-relational database. For example, a COBOL application may use ISAM files. PostgreSQL is an object relational extensible database. We have found that we can create adapters that allow us to superimpose other data stores on top of PostgreSQL in order to provide the existing legacy application with the API that the existing programs need to use PostgreSQL instead of ISAM with no change to the exiting COBOL programs. Several features of Postgres help facilitate this.

The IBM Compose site describes many of the features that make PostgreSQL an excellent candidate for use in this role of legacy data source replacement. Below are links to the articles that describe some of
these features. The article is in two parts. In Part I of they describe storing the data - the model, structures, types and size limits. In Part II here, they describe data manipulation and retrieval, including indexing, virtual table features and query capabilities.

https://www.compose.com/articles/what-postgresql-has-over-other-open-source-sql-databases/

https://www.compose.com/articles/what-postgresql-has-over-other-open-source-sql-databases-part-ii/

Why REST API?
We view REST and HTTP APIs as first and foremost a platform and programming language independent application integration solution. In our view, there are two key use cases for REST and HTTP APIs in an application solution. First to connect the application development framework components. For example, an application can consist of a backend built with .NET and a frontend built with JAVA or PHP. Secondly, to either allow access to the finished application components from outside resources or to access external application services from inside the application. An example of allowing access to system components can be the use of a REST API with CRUD callability to access the database. An example of using external APIs can be the use of Google Maps from a COBOL application screen.

The CUBA-Platform is a full stack web application development framework that is “glued” together with REST and HTTP APIs https://doc.cuba-platform.com/manual-6.5/app_tiers.html. Therefore, applications built with this framework connect the clients to the backend with REST and HTTP APIs.

The CUBA-Platform also allows applications built with it to consume and produce REST and HTTP APIs with relative ease when compared to other approaches. This article discusses the CUBA Platform REST API usage. “REST, APIs, and CUBA Platform's REST API” https://dzone.com/articles/cubas-rest-api-if-only-roy-would-knew-about-it

Further, in addition to the built-in use of REST and HTTP APIs and the automatically generated ones, the CUBA-Platform allows you to build your own clients if you need to do that.

Custom front-ends for your CUBA app


This is a showcase CUBA application that contains a portal module with AngularJS client application.

https://github.com/cuba-platform/sample-portal

Benefits of REST and HTTP APIs
There are many articles available that describe the benefits of using REST and HTTP APIs. We have selected some for you.


http://www.oracle.com/technetwork/articles/olamendy-rest-092504.html

What Do We Need from REST and HTTP API Web Services?
### Why High Level Full Stack Framework?
Frameworks are tools that allow you to implement all the common functionality much faster and easier. We believe new developers should start using frameworks as early as possible. Frameworks will enable developers to get results much faster, and at the same time they teach best practices in a very natural and intuitive way. Further by using frameworks, developers can absorb a huge amount of knowledge incredibly quickly and avoid wasting time on anything unnecessary.

In web application development, there are an enormous number of technologies to choose from, and that choice can often be difficult for the novice. Picking the best framework for developing an application (or a set of frameworks), means trying to find the best trade-off between the following criteria:

- **Range of Applicability** - what part of the application can be developed using the framework? Do I need a set of frameworks or just one?
- **Development Speed** - how fast will development go?
- **Manageability & Flexibility** - what would be the architectural, performance and other limitations?

This article provides an overview of the many different types of frameworks available. [https://www.cuba-platform.com/blog/classification-of-development-frameworks-for-enterprise-applications](https://www.cuba-platform.com/blog/classification-of-development-frameworks-for-enterprise-applications)

### High Level Full Stack Frameworks
This class introduces the next abstraction level over the Narrow-focused Frameworks. So, you already have entire infrastructure to build a full-scale enterprise applications times faster. Taking CUBA Platform as an example, it provides a full-stack architecture for building 3 tier applications and includes widely used common features for enterprise apps such as soft deletion, data-aware components, file storage, as well as high level modules: authentication and authorization, audit of data and others. This class is most appropriate for developing ERP-like systems, line of business applications or custom software for the corporate segment.
At Transformix, since our main interest are in extending the life and functionality of existing database-centric applications we needed a full stack application development framework with the following characteristics:

1. Full function
2. Extensible
3. Database centric

We found that the CUBA-Platform exceeded our requirements.

These articles provide more about the CUBA-Platform:

https://www.predictivanalyticstoday.com/cuba-platform/
https://www.cuba-platform.com/framework

Case Studies
Perhaps the most important points we want to emphasize are:

1. The value of REST and HTTP APIs in modernizing existing applications.
2. The value of PostgreSQL in modernizing existing applications
3. The value of using a full-stack framework
4. How REST and HTTP APIs add value to the full stack framework
5. How REST and HTTP APIs help modernize existing applications.

Our case studies look at complete database-centric applications. By that I mean Database applications consists of forms, formatted reports, queries, programs. We use two different case studies:

1. Case 1: Shows the difference between assembling your own framework and using a full stack CUBA-Platform framework.
2. Case 2: Shows our ModernizationInPlace approach with the well-known Microsoft Northwind Traders Microsoft Access application.

ModernizationInPlace Progressive Modernization
Figure 4 shows how an existing legacy application such as that developed with Microsoft Access can be made progressively more modern in stages at first to provide the benefits of PostgreSQL as a database replacing the MS Access Jet database and then by adding a supplemental set of web, mobile and REST API clients that extend the reach of the application.
CASE 1: Building a Single Table Backend and Front-End REST API Application

Most articles on REST API based applications “get down into the weeds” of coding. That is, even though the developers in most cases are using some sort of combination of frameworks, they still spend a lot of time and energy gluing these things together.

This example case shows the difference between assembling your own set of frameworks and using a full stack CUBA-Platform framework. We assume that the goal is to get the benefits of REST and HTTP APIs with the minimal amount of effort. That is, to build applications with web, mobile, desktop or API interfaces with database backends in the least amount of time with the lowest cost while not sacrificing security and reliability and with the smallest amount of vendor lock-in.

This example demonstrates the difference between using Spring Boot + AngularJS + Spring Data + JPA CRUD versus the CUBA Platform full stack framework and PostgreSQL. After the environment is setup, starting from scratch, a skilled JAVA developer can build a simple one table application with the Spring Boot ... set of frameworks in a few hours. A somewhat skilled non-JAVA developer can build the CUBA-Example in about 5 minutes.

Original example: Spring Boot + AngularJS + Spring Data + JPA CRUD App Example


Our example: The CUBA-Platform Approach
Video of the CUBA solution

https://drive.google.com/open?id=1IDQonIkYs5W4mM9n7m6KVqeN5eoMcahj

The CUBA Project you can download

https://drive.google.com/open?id=1r3jniE3V_REPoSveU9MkfrA2evZZAMuD

CASE 2: MS Access Northwind Traders to PostgreSQL and CUBA-Platform

This example is meant to show the power of a full stack web application development platform, specifically, the CUBA-Platform in modernizing a legacy MS Access application. In this case study, we take an existing application that was developed for MS Access on Windows and open it up to platform independence and nearly unlimited growth.

Converting Access Database Objects (AccessToSQL)

It is important to note that Microsoft offers a solution to take MS Access database applications to MS SQL Server. For those of you who are familiar with the Microsoft AccessToSQL solution which is done with the SQL Server Migration Assistant (SSMA) table 1 shows the contrast between the Transformix-CUBA-Platform solution and the Microsoft approach. Note that the Transformix-CUBA solution includes the migration of forms, reports, modules and macros. That is, it is meant to migrate the whole application.

https://docs.microsoft.com/en-us/sql/ssma/access/converting-access-database-objects-accesstosql

<table>
<thead>
<tr>
<th>Access Object</th>
<th>Resulting SQL Server Object</th>
<th>TransformixTools Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>table</td>
<td>table</td>
</tr>
<tr>
<td>column</td>
<td>column</td>
<td>column</td>
</tr>
<tr>
<td>index</td>
<td>index</td>
<td>index</td>
</tr>
<tr>
<td>foreign key</td>
<td>foreign key</td>
<td>foreign key</td>
</tr>
<tr>
<td>query</td>
<td>view</td>
<td>view</td>
</tr>
<tr>
<td></td>
<td>Most SELECT queries are converted to views. Other queries, such as UPDATE queries, are not migrated.</td>
<td>Most SELECT queries are converted to views. Other queries, such as UPDATE queries, are not migrated.</td>
</tr>
<tr>
<td>Access Object</td>
<td>Resulting SQL Server Object</td>
<td>TransformixTools Solution</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>AccessToSQL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SELECT queries that take parameters are not converted, nor are cross-tab queries.</td>
<td>SELECT queries that take parameters are not converted, nor are cross-tab queries.</td>
<td></td>
</tr>
<tr>
<td>report</td>
<td>not converted</td>
<td>Converted</td>
</tr>
<tr>
<td>form</td>
<td>not converted</td>
<td>Converted</td>
</tr>
<tr>
<td>macro</td>
<td>not converted</td>
<td>Replaced</td>
</tr>
<tr>
<td>module</td>
<td>not converted</td>
<td>Replaced</td>
</tr>
<tr>
<td>default value</td>
<td>default value</td>
<td>default value</td>
</tr>
<tr>
<td>allow zero length column property</td>
<td>check constraint</td>
<td>check constraint</td>
</tr>
<tr>
<td>column validation rule</td>
<td>check constraint</td>
<td>check constraint</td>
</tr>
<tr>
<td>table validation rule</td>
<td>check constraint</td>
<td>check constraint</td>
</tr>
<tr>
<td>primary key</td>
<td>primary key</td>
<td>primary key</td>
</tr>
</tbody>
</table>

Table 1 - Microsoft Access2SQL versus Transformix Approach

Modernization InPlace Steps

1. Access 2016: Split a Database

Splitting a database involves separating the database into two files. One file is the back-end database that contains all the tables and data. The other file is the front-end database that contains all other objects, such as queries, reports, and forms.
Users work with a local copy of the front-end database, and the data is sent/retrieved from the back-end database. So, there could be many copies of the front-end database, but only one copy of the back-end.

By doing this, only the data needs to be sent across the network. With a non-split database, all database objects need to be sent across the network, which typically results in a slower experience for the user.

https://www.quackit.com/microsoft_access/microsoft_access_2016/tutorial/split_a_database_in_microsoft_access.cfm

---

**MS Access File:** SplitNorthwindmdb3.accdb  
https://drive.google.com/open?id=1EC46kG3Hd6GZHeX9Z_NBrKHTtl5JL95x  
SplitNorthwindmdb3_Backup.accdb  
https://drive.google.com/open?id=1NSwCxlVvdlWoG4hybU4YspUUziakhuj4

---

2. USING MS ACCESS WITH POSTGRESQL

Many in the PostgreSQL community use Microsoft Access as a front-end to their PostgreSQL databases. Although MS Access is strictly a windows application and PostgreSQL has its roots in Unix, the two go well together. A large part of that reason is because the PostgreSQL ODBC driver is well maintained and has frequent updates. You can expect one new ODBC driver release every 4-6 months. There exist only 32-bit production quality drivers. The 64-bit driver is of alpha quality. In addition to other front-ends to
PostgreSQL that utilize the ODBC driver used by Windows developers, there is VB 6 (VB.NET/C# use the ADO.NET driver also very well maintained), Visual FoxPro, Delphi, to name a few).

This is an article that describes the process used to use MS Access as a client to a PostgreSQL database using ODBC.

**The Frontend**
MS Access File: *SplitNorthwindmdb3.accdb*

[https://drive.google.com/open?id=1wlHv85c5MvYkzK2U2bO8aOXLWN56bnkz](https://drive.google.com/open?id=1wlHv85c5MvYkzK2U2bO8aOXLWN56bnkz)

**The Backend**

PostgreSQL Northwind database backup

[https://drive.google.com/open?id=1wlHv85c5MvYkzK2U2bO8aOXLWN56bnkz](https://drive.google.com/open?id=1wlHv85c5MvYkzK2U2bO8aOXLWN56bnkz)
Create user “northwind_user” with password “northwind_user”

Grant login and superuser capability

Create PostgreSQL database “northwindmdb3” with owner “northwind_user”

Restore the database.

Create an ODBC DSN named “PostgreSQL30” and point to the PostgreSQL database northwindmdb3.

3. Create CUBA-Platform Clients

The next stage of application modernization involves selectively converting forms, reports, functions, procedures, persistent relations, etc. Using PostgreSQL as a database the CUBA-Platform can ease the level of difficult of this set of tasks. Moreover, with this approach it becomes possible to scale up desktop applications well beyond the capacity their previous limitations and move them to the cloud if that is desirable.

CUBA-Project Video

https://drive.google.com/open?id=1flbEmIPnphkBCommTlTeWbcrSwSjT6l19

Eclipse Video

https://drive.google.com/open?id=1gNAhvTxnl0ap7FUnPUs-3OLZopB8XjK5
CUBA Project

https://drive.google.com/open?id=16sWIIFRKJYFwJFK_GpygiJ2W-xHWUfW4

Backup

The database backup should have been restored in the previous step. It is the same database used by the MS Access client.

4. Optionally retire the original MS Access client.

After all the functionality has been recreated using the CUBA-Platform. There might come a time when it is no longer desirable to keep the MS Access client. The good news is that the application stakeholders get to decide if and when this happens.

Summary and Conclusions

What users expect from their applications has changed and is changing. Yet applications have been and must be written with existing tools. When these applications are valuable it is usually impractical to replace them with the latest and greatest thing. REST APIs, PostgreSQL and a full stack web application development framework like the CUBA-Platform with a few added tools from Transformix and others can be used to extend the long-term usefulness of these applications. This paper shows how this can be done in small steps using an approach called ModernizationInPlace™.