Understanding the Technical Barriers of Migrating VSAM to RDBMS

White Paper

Notice

Astadia makes no warranty that the content of this document is timely or complete; or is free of omissions, inaccuracies, typographical errors, or other errors. All contents of this document, including but not limited to the text and images contained therein, are made available on an "as is" basis without any warranty, express or implied, of any kind, including the implied warranties of merchantability, title, non-infringement, quality, or fitness for any particular purpose.

Certain sections of this document may contain forward-looking statements that are based on product management's expectations, estimates, projections and assumptions. Words like "plans," "intends," "expects," "believes," "future," "estimates" and variations of these words and similar expressions are intended to identify forward-looking statements. These statements are not guarantees of future performance and involve certain risks and uncertainties, which are difficult to predict. Therefore, actual future results and trends may differ materially from what is forecast in forward-looking statements due to a variety of factors.

Trademarks
Astadia, TestMatch, DataMatch, DataTurn, CobolBridge and CodeTurn are trademarks or registered trademarks of Astadia. These trademarks may not be used without the permission of Astadia. The absence of a product, company, or service name or logo from this list does not constitute a waiver of the trademark or other intellectual property rights of Astadia concerning that name or logo.

Other trademarks that appear in this document are used for identification purposes only and are the property of their respective owners. These marks may not be used without the permission from these owners.

TABLE OF CONTENTS

1. VSAM to RDBMS Migration: an Introduction

Many large organizations still store gigabytes of mission-critical data in VSAM data sets on platforms such as z/OS and BS2000/OSD. Often, these environments will have a variety of software applications (often built in-house or custom made) that access and manipulate the data in these files through different VSAM Data Access Techniques. Frequently, these VSAM-based applications are written in the COBOL programming language.

Database vendors like Microsoft, IBM and Oracle have invested huge sums of money in the development of database management systems. Large open-source products such as MySQL and PostgreSQL, have recently gained a lot of traction with the advent of cloud computing. Generally branded as an RDBMS (Relational DataBase Management System), these products take the concept of "data storage and access" and package it with advanced functionality and standardized access via SQL. They are the working horse of all major corporations and seem like the natural target for VSAM migrations.

Simply planning and executing the switchover of data from VSAM file storage to RDBMS hosting can be a difficult task in and of itself. Not only do large amounts of data need to be migrated, but the applications themselves need to be adapted and tested in order to ensure compatibility with the new RDBMS. The adaptation and testing of large amounts of COBOL code for this purpose can be a lengthy process.

A migration methodology, such as Astadia's Fast Track, offers benefits to companies that wish to retarget from VSAM to RDBMS. The methodology ensures that the application code can be made compatible with the new model both quickly (so that application change requests and maintenance requirements only need to be put on hold for a short while) and completely (so that the entire data set can be moved from one system to another in a single operation).

This implies automation, i.e. working with tools. A conversion software tool will ensure the quality of the produced code by generating modifications to the original code in a consistent way. Creating such a tool is a complicated, costly and lengthy process, for which most IT departments don't have the time or budget.

Astadia has developed a VSAM to RDBMS conversion toolkit, called DataTurn, that automatically retargets applications to an RDBMS architecture.

This article explains the technical difficulties that DataTurn has overcome. The illustration examples use COBOL as the source language example because this is often used in combination with VSAM.

2. Data Storage and Retrieval, in VSAM COBOL and RDBMS

One of the strong points of COBOL has always been its feature to afford programmers the ease of doing two things at once: defining variables and describing the way data is organized within the layout of a data file.

2.1. Defining Variables

In order for applications to process data, they need to be able to manipulate it quickly and ensure that all data conforms to basic validation rules. In order to manipulate data quickly, COBOL developers define variables that are used in the program flow and assign "types" to them in order to ensure basic validation. Through validation, developers can ensure that when an end-user enters a customer's credit card number, for instance, that the data entered contains exactly 16 numerals and no letters.

2.2. Defining the Structure of Data Files

By storage structure definition, we essentially mean the way the program defines, within a file that contains data, at what place a certain piece of information can be found. By defining the structure of a data file, for instance, developers

can say that a customer record consists of the customer's name (say, 30 letters max), followed by the city the customer lives in (say, 15 letters max), followed by the customer's credit card number (16 numerals).

With this definition, programs know that the credit card number of a customer can be found by reading 16 characters from a record in the customer data file, starting at position 46.
In COBOL, programmers have the ability of doing both actions in one single operation, and the VSAM standard supports this. On top of this powerful combination, COBOL offers tools or shortcuts that simplify the definition of file structures and the variables used in the program flow.
These shortcuts, unfortunately, are characteristic to the way COBOL allows this combination of features and are not supported in an RDBMS-mindset.

These shortcuts and their resulting incompatibilities are threefold:

- The ability to introduce structure to data variables or records simultaneously;
- The shortcuts that allow multiple sets of a field to be defined within a single record (the OCCURS clause); and
- The shortcuts that allow a single record to be structured according to different rules without first needing to copy the data from one structure to another (the REDEFINES clause).

Each of these incompatibilities will form a barrier to the retargeting of an application system from VSAM to RDBMS, as none of these "shortcuts" (which constitute some of the main strengths of the COBOL language) are supported within the RDBMS architecture.

Four technical issues are relevant to the retargeting of VSAM COBOL applications to an RDBMS architecture. These can be categorized into two main groups:

- Incompatibility issues arising from a shift of the programming model (cursor-based versus index-based), and
- A shift of the way you think about data (incompatible record structures and data types)

In the section that follows, we will examine each of these issues in more detail.

2.2.1. Data Structure

COBOL offers the ability to introduce structure to records within a data file. Structure is introduced through the definition of relevant information that can be attained from the combination of other fields. For instance, a customer's telephone number could be defined as a combination of up to four separate fields - the country code, area code, prefix, and subscriber number. The last two could be regrouped and defined as the "local number" of the customer, which must be dialed from within the same area code.
In COBOL, such a structure could be defined as follows:

```
01  customer-record.
    03 telephone-number.
       05 country-code pic 999.
       05 area-code pic 999.
       05 local-number.
          08 prefix pic 999.
          08 subscriber-number pic 9999.
```

Within COBOL, the structure of the data file that stores the customer records is defined and at each level the programmer is left with a variable that can be used to manipulate or retrieve the underlying data. In this example, further on in the program's processing, the programmer could write an instruction that accepts screen input and writes the telephone number directly to the variable "local-number." However, it is equally valid for the COBOL programmer

to write two separate instructions to accept screen input and to write the contents of two separate fields on the screen to the two variables "prefix" and subscriber-number" separately.

In an RDBMS, it is impossible to store such a structure, as database management systems have a flat record buffer that does not allow the "nesting" of fields within other fields. Within the RDBMS philosophy, each field is an element in its own right, and fields cannot be split or grouped into a level that gives the data an aggregate meaning. For this reason, any automated solution to retarget VSAM to RDBMS will have to achieve the "flattening out" of data structures in one way or another.

2.2.2. OCCURS

OCCURS is a mechanism within VSAM COBOL to define an array of data that can be nested within a record. If we return to our customers example, suppose we would wish to store up to three telephone numbers for each customer. With the OCCURS clause, COBOL provides developers with a shortcut to indicate that the "telephone number" is repeated twice in the data file in each customer record. This can be achieved very easily, by adding the text "OCCURS 3" behind the definition of the "telephone-number" variable in the file description, as follows:

```
01  customer-record.
    03 telephone-number occurs 3.
       05 country-code pic 9.
       05 area-code pic 999.
       05 local-number.
          08 prefix pic 999.
          08 subscriber-number pic 9999.
```

In strict RDBMS thinking, the definition of arrays as shown in the above example is inefficient and is therefore disallowed. The RDBMS reasoning behind this is that (one) you may one day come across a customer that has four telephone numbers, in which case, your array of three would work restrictively; and (two) for the majority of the customers that may have only one or two telephone numbers, the allocation of a fixed amount of space for three numbers wastes space.

A relational database management system solves this through allowing the definition of a separate table within the database, related to the customer table via a key field, in which as many telephone numbers per customer can be defined as are necessary.

To solve this, any automated solution to retarget VSAM to RDBMS will have to strike a balance between the storage efficiencies and elegance that can be achieved through the strict adherence and conformity to the RDBMS philosophy on the one hand, and the performance gains that can be achieved by not doing so on the other. This way, the solution would either automatically create related "sub" tables within the database (a solution that deserves consideration, especially when the OCCURS clause is followed by the "depending on" clause), or work with a similar "flattening out" of the array, repeating the definition of each element for each reoccurrence.

2.2.3. REDEFINES

The COBOL REDEFINES clause gives developers the ability to organize the layout of a data file in an inconsistent way, in effect allowing the data, stored in a specific record, to be interpreted in different ways. The REDEFINES clause allows this while at the same time allowing the different record layouts to be stored in a single file.
Back to our example for telephone numbers and the customer record file, suppose we had to deal with customers in the United States (where the area code is 3 digits and local numbers are 7 digits long) and in Denmark (where an area code is not used, and all local numbers are 8 digits long). Within VSAM COBOL, it is perfectly acceptable to define a "double layout" for customer telephone number data in a single file as follows:

```
01  customer-record.
    03 tel-number.
       05 country-code pic 9.
       05 area-code pic 999.
       05 local-number.
           08 prefix pic 999.
           08 subscriber-number pic 9999.
    03 tel-number-denmark redefines tel-number.
       05 country-code pic 99.
       05 local-number.
           08 subscriber-number pic 99999999.
       05 filler pic x.
```

Such definitions are completely impossible in an RDBMS architecture. There are a few possible solutions to this gap in architectures as concerns the REDEFINES clause, however, any automated conversion tool will have to solve this problem. By defining one of the concurrently incompatible definitions as the master reference, or having all definitions available in the constructed table, but only using 1 actually depending on the actual data.

2.2.4. Index-Based versus Cursor-Based Program Flow

One of the strong points of VSAM COBOL has always been its feature to afford developers a high degree of control and transparency over the physical location within data files where data gets read from or written to. However, as is often the case in life, this benefit of control comes at a price, and in this case the price is developers' discipline.

In VSAM COBOL applications, actual data is not stored in the same place as the descriptions of the layout (the "maps" of the data or the so-called meta data) of the data files. With VSAM, data is stored in multiple, physically separated data files. In COBOL, the layout of files is recorded in the file descriptions of the programs themselves, not in the data files or in the index files that give VSAM its superb performance.

Because of this, VSAM COBOL developers always must be careful that the file descriptions they use in their programs are of the latest and most recent version. Whenever the layout of a file needs to be changed, the developers need to ensure that the changes are introduced in all programs where the data file in question was used. The danger is, if they forget to keep programs and data files in sync, it could be many days or weeks later that the discrepancy would surface in VSAM COBOL as a corrupt file. The reason why it could sometimes take so long is because in VSAM, the data is dead, and does not give errors or generate warnings.

RDBMS products bring data to life, by uniting the data with the description of its contents and layout, effectively giving the data self-awareness and the ability to prevent itself from becoming corrupt. With RDBMS products, since data is alive and can take care of itself to a large degree, somewhat less discipline is required of developers than was the case with VSAM COBOL. Naturally, this relaxation of developers, as a benefit, also comes at a price. This time the price is reduced control and transparency of the developer over the physical location on the disk where the data gets read from or written to. With most RDBMS systems, developers are unable to find out where or how, physically, a record or any piece of information is stored on a disk.
This shift in benefits and costs to developers (more control vs. more discipline, and less discipline vs. less control) is the source of a serious challenge in the retargeting of VSAM COBOL applications to an RDBMS like Oracle.

2.2.4.1. Control

VSAM COBOL developers exercised a lot of control in their programs by working with the indexes stored in KSDS files (Key Sequenced Data Sets). With the ability to work with multiple indexes, VSAM COBOL developers could create applications that could neatly browse through large files of data, without placing heavy loads on memory resources (be they volatile or non-volatile) in the process. With the usage of index files alone, these applications could change the direction in which they read data from the data files instantaneously. Furthermore, the use of index files alone allowed developers to browse through a data file using one index, and then switch to another index just as instantaneously.

Within the RDBMS mindset, indexes and especially keys play a very similar role as they did within the VSAM mindset. However, indexes in an RDBMS are not things that are accessed as such. They are used by the RDBMS engine in order to boost the performance of queries, but it is here where the similarity ends.

Within RDBMS, data is retrieved from a database table or set of tables through a query. The query has two main functions – to specify which fields within the table or set of tables that must be retrieved (horizontal qualification of the queried data) and to specify which records of the table or set of tables must be retrieved (vertical qualification of the queried data). Said differently, if you want to get anything out of an RDBMS you have first to ask it a question (submit a query), and the only answer you can expect to receive will be in the form of a two-dimensional matrix of fields and records.

This "answer" being a matrix of fields and records is called a record set. Once the record set is made, the RDBMS will supply the data to the programs one row at a time. This is called the "fetching" of the data and happens with the help of a cursor.

2.2.4.2. Retargeting Issues

The concepts of "record sets," "fetching," and "queries" are all concepts that are foreign to the VSAM mindset, and COBOL applications that get retargeted from VSAM to RDMBS will have to have all of them introduced somehow. While technically this may be easy to achieve, it remains a challenge to ensure that these introduced concepts can be implemented in a consistent way, and that the solution always assures acceptable levels of performance of the programs, once retargeted.

These issues can get tricky, and there are two issues that easily come to mind. The first issue regards the hopping of indexes (caused by two consecutive READs on different keys) or the reversal of their direction (caused by using consecutive READ PREVIOUS and READ NEXT operations). While hopping or changing indexes in VSAM is easy, within the RDBMS mindset we generally will be confronted with another cursor that must be active in another record set, each time a hop takes place. Any retargeting solution will have to provide a standardized answer to this so that performance is not too heavily impacted or that the resulting code is not too cumbersome to maintain.

In addition, when COBOL programs reverse the direction of their queries, for instance, by scrolling upwards in the data file after having browsed downwards in the data file, the choice of target RDBMS system may influence the ease with which the cursor's ability to invert can be retargeted.

Some modern RDBMS products implement "scrolling cursors" that can change direction within the record set, but not all do. If the target RDBMS does not provide support for "scrolling cursors" then the solution may, in the worst case, must rely on the definition of double record sets in the database each time a query is made. The cursors in these double record sets will always scroll forward, but the order of the records in the record set will have to be opposite.

The second issue, of course, regards the ability of the retargeting solution to reliably produce record sets that are minimal in size.

For instance, if in a particular area of an application, it is only necessary to retrieve one or two records from a very large table, the creation of queries that include the full table of information in the record set will cripple application performance. The solution will have to find some way to generate the appropriate "where" clauses in the resulting code that serve as the vertical qualification of the query, or where this is impossible (as it most often is) devise a mechanism that allows full tables to be scanned without necessitating the creation of whopping entire record sets.

More information on the implementation of such a mechanism is available in the DataTurn toolkit.

Get in Touch

info@astadia.com
www.astadia.com
+1 877 727 8234