# COBOL APPLICATIONS: A SECOND YOUTH!

An intelligent and "painless" solution can be used to bring new life and usefulness to applications and tools written in the memorable COBOL programming language. These techniques allow you to continue using existing tools, so that evolution does not mean renouncing the reliability and efficiency of systems tested through years of development.

**Requirements**
Knowledge required: COBOL programming language

Needed software: Microsoft Visual Studio, c-treeRTG COBOL Edition evaluation version available from FairCom (www.faircom.com)

Commitment: Moderate

Time of realization: 2 hours

Today there are numerous corporations that rely on "legacy" applications—those written in programming languages that do not have interfaces with the new paradigms the changing market dictates, such as ODBC, ADO.NET, and SQL. The need to modernize has become increasingly pressing, especially because of the strong "push" exerted by the end-users who—in addition to requiring more ergonomic interfaces—increasingly need integration, reporting, business intelligence, and more modern tools.

The value of these existing applications is by no means negligible. Much of that value is due to the maturity derived from the persistent evolution and stability achieved through years and years of development. Rewriting such software is not a feasible option, neither technically nor economically.

What should be done then? The winning trick is to combine the reliability of the existing applications with the new opportunities offered by the latest generation tools. In summary, a quick and "painless" solution is needed to be able to ensure continuity and linearity with the past. Such a solution has been successfully implemented by the company Metodo S.r.l. of Modena (Italy).

This often-overlooked approach can be thought of as an “egg of Columbus”: and idea that seems simple once it is known (Columbus is said to have challenged his critics to stand an egg on its end...after their failures, he simply tapped the egg on the table, breaking the shell so it easily stood on end).

In practice, the solution is to modernize the database (or file system as it is often called in the COBOL world) while maintaining the infrastructure and applications without change and—more importantly—without changing a single line of code.

# A Brilliant Solution

During the modernization process, Metodo S.r.l. had to face various objective problems:

- Existence of applications written in COBOL (tens of thousands of lines of code)
- Daily maintenance of the applications making them a moving target
- Developers were experts in the field of business flow and COBOL, but not in new programming languages
- Need to keep the entire system 100% running
- Evolving user interfaces, communication with the Web, and the need to exchange data with other applications and languages

The solution chosen by Metodo S.r.l. to solve these problems was simple and effective. Above all, the COBOL programs needed to be kept unchanged, including its evolution and maintenance. In addition, it was necessary to gradually integrate new programming paradigms. Let's find out how they managed to achieve those objectives.

**A SOLID BASE**

Metodo S.r.l. (www.metodo.net) is LAPAM Federimpresa (www.lapam.mo.it/Lapam) software house, which since 1995 has started a modernization project of its infrastructure and software. It has been among the first realities to use GNU/Linux as a software platform—both for the servers and clients—creating their own distribution of the powerful Open Source operating system. Lapam Federimpresa is a Federation of Associations, in the province of Modena, which represents the entire business world. The Lapam computer system—developed and maintained by Metodo S.r.l.—includes 800 Unix workstations spread over 45 locations, and access to an archive of 750 GB containing data of approximately 12.000 companies for which it elaborates 9.000 accounting processes, 37.000 monthly pay slips and 45.000 tax returns.


**Fig. 1 - The "modernization" process of a COBOL application**

# Possible Co-Existence

The COBOL language uses an ISAM-type access and writes directly to a buffer containing the data in a proprietary format. Although the data is correctly interpreted by the COBOL application, to other languages it is simply strings of bytes. Moreover, COBOL vendors, eager to protect their market, have been reluctant to push integration with other languages, making this is a closed reality.

Considering all this and the requirements to make the transition as smooth as possible, Metodo S.r.l. has replaced the native COBOL file system with c-treeRTG COBOL Edition developed by FairCom (see http://www.faircom.com/ace/rtg_cobol_t.php). This operation, absolutely simple from a code point of view, has allowed maintaining the ISAM access of COBOL and, consequently, has not required any modification to the source code and not caused any degradation in terms of performance. Indeed, the result has been an obvious improvement of both speed and stability. The FairCom file system, once linked to the runtime, has immediately allowed sharing COBOL data with other programming languages. Metodo's choice has been to use the scripting Tcl language (www.tcl.tk), migrating the entire application module-by-module, allowing simultaneous access and data sharing by the two architectures.

**EVEN MORE SIMPLE**

Taking as an example Metodo S.r.l., FairCom has developed an even more linear and extended solution while keeping the same simple approach. The support has been extended from FairCom to various COBOL (ACUCOBOL, MicroFocus, isCOBOL, NetCOBOL, COBOLIT etc.), that now—thanks to the FairCom file system—can share data using SQL via JDBC, ODBC, Ado.NET, PHP, Python and with traditional programming languages such as C, C + +, JAVA etc. all using the ISAM level.

# Limits of Alternative Solutions

Various solutions on the market allow you to "modernize" a COBOL application, but often involve disadvantages. For example, translating a COBOL solution to a fully relational system (SQL Only)—such as that provided by Oracle or Microsoft SQL Server—has several disadvantages: the use of expensive bridge components; translation of each COBOL request in a SQL statement (losing the performance of the existing application); change of the existing code because of the different data management. It does not allow the co-existence of the different applications while migrating module-by-module which was necessary to complete the program, given the required changes to the source code. Moreover, this technology is certainly valid, but far from the "file-by-file" management that FairCom maintains, which is more natural for the management and maintenance that COBOL programmers are used to (Fig. 2). Other solutions try to tackle the problem of data sharing with more modern applications, through the use of batch procedures that do not keep synchronized copies of the database accessible through the Web or with reporting tools. This type of solution does not allow you to update or access data in real time with all the limitations that come with it.

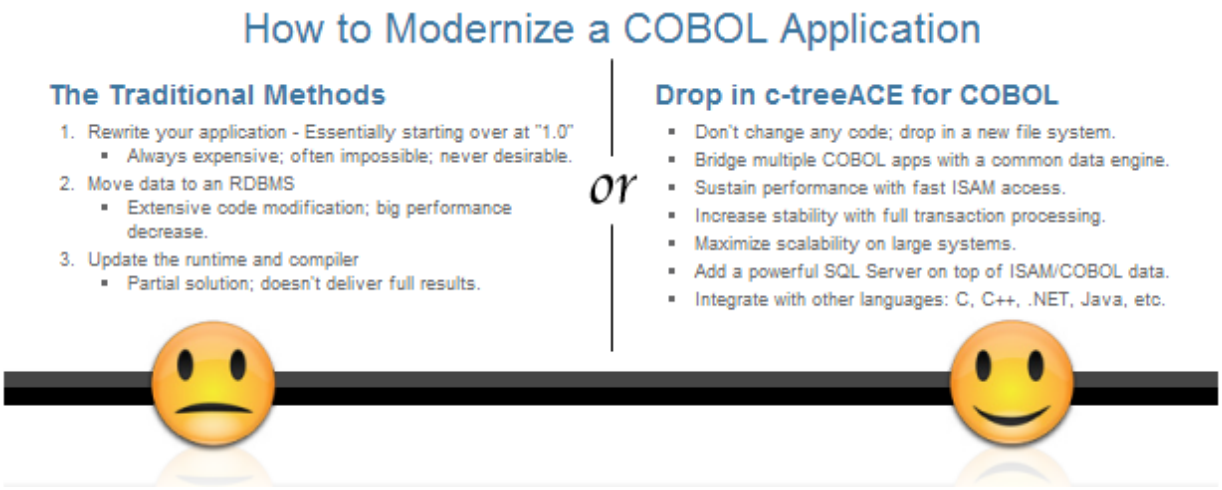


**Fig. 2 –Comparison between traditional methods and methods using the new FairCom c-treeRTG COBOL Edition**

# One for All!

Integrating FairCom technology into the existing applications is extremely simple. Typically those who have used third-party file-handler or written routines for indexed file management have often employed a more programmatic record-oriented approach: ISAM. Today, very often, these realities are coming to terms with the limitations of the systems developed in-house or with the technical/commercial policies of the vendors, which are not always in line with the real needs of the software houses. FairCom provides a valid alternative for all these cases by using an extremely reliable and portable ISAM technology interfaced directly with the SQL world and many other APIs. For the Btrieve world, a direct driver allows the Btrieve library to be replaced with c-treeRTG COBOL Edition, keeping the same calls in the code and thus ensuring a number of both commercial and technical advantages.

**FIRST OF ALL LINEARITY**

Metodo S.r.l., which decided to make use of a procedure provided by a leading software house for payroll handling, did not want to change from the FairCom file system. They simply replaced the COBOL file system directly with FairCom c-treeRTG runtime, ensuring data uniformity and stability. This further confirms the linearity of the solution to the point that an expert end user is able to implement it without having access to the source code.

# From Theory to Practice

Now that we know the benefits of FairCom technology and have the proof that it has been successfully used by Metodo S.r.l., below is a brief guide for the ACUCOBOL users taken from the c-treeRTG COBOL Edition manual. The steps to follow are more or less similar to those of other COBOL dialects, except for Veryant isCOBOL that does not require any operation since the support for c-treeRTG COBOL Edition is integrated.

Let's start! To add the c-treeRTG COBOL Edition support to ACUCOBOL-GT COBOL on Windows, the Microsoft Visual Studio development environment is necessary. To add the support to ACUCOBOL-GT version 9, the use Visual Studio 2008 is needed. Let’s now turn to the changes to be implemented to successfully compile the support. As a first thing, the ctreeacu.c file in the ACUCOBOL-GT lib directory has to be copied:

```
copy win32\Driver\ctree.cobol\Acucobol\ctreeacu.c C:\AcuGT\lib
```

Soon after, it is necessary to copy the **ctutil.exe** utility and the *mtclient.dll* DLL in the bin directory of ACUCOBOL-GT through the following commands:

```
copy win32\Driver\ctree.cobol\Acucobol\mtclient.dll C:\AcuGT\bin
copy win32\Tools\cmdline\ctutil.exe C:\AcuGT\bin
```

At this point, the Visual Studio solution *wrun32.sln* has to be opened and the previously copied *ctreeacu.c* file added to the *wrundll* project. To add the file, just click on the **Project** menu, select **Add to Project** and choose *ctreeacu.c* from the ACUCOBOL-GT lib directory. Soon after, the file management of ACUCOBOL_GT file system called *filetbl.c* (the file is part of the *wrundll* project previously created with Visual Studio) needs to be opened and modified. The *filetbl.c* file

has to be modified in three particular sections. Let's see how to proceed. First, the following three lines need to be identified:

```
#ifndef USE_VISION
#define USE_VISION 1
#endif
```

Once the lines of our interest have been found, immediately after the following statement has to be added:

```
#ifndef USE_CTREE
#define USE_CTREE 1
#endif
```

To proceed with the second change, the line to be found is:

```
extern DISPATCH_TBL v5_dispatch,...;
```

Immediately after this point, the following line has to be added:

```
extern DISPATCH_TBL ct_dispatch;
```

Finally, the third required change can be done. To do this, the following block has to be identified:

```
TABLE_ENTRY file_table[] = {
#if USE_VISION
{ &v5_dispatch, "VISIO" },
#endif /* USE_VISION */
```

Soon after, the following definition needs to be added:

```
#if USE_CTREE
{ &ct_dispatch, "CTREE" },
#endif /* USE_CTREE */
```

At this point, it is necessary to recompile the ACUCOBOL-GT runtime included within the *wrun32.dll* DLL. To do this, select Build from the Visual Studio Build menu. Once this is done, copy the new *wrun32.dll* in the ACUCOBOL-GT bin directory:

```
copy C:\AcuGT\lib\wrun32.dll C:\AcuGT\bin
```

Now, verify that c-treeRTG COBOL Edition support is correctly compiled by running the **wrun32** *-vv* command. This command will show you the various file systems supported by the ACUCOBOL-GT version in use. If c-treeRTG COBOL Edition is properly implemented, the following file system will also be shown among the various others:

```
FairCom c-treeRTG version 10.1.0.xxxxx-yymmdd file system (interface v10.1.0.xxxxx-yymmdd)
```

**A PROVEN TECHNOLOGY**

For many end-users and for some of the experts, FairCom is not a renowned name like Oracle or Microsoft, especially because FairCom technology has always been used by ISVs and software developers as an internal component and not as a commercial product to buy separately. Since 1979 FairCom provides a database known as the c-tree, which evolved from a simple file-handler to a client/server technology in a transactional environment with a SQL relational layer, with the name of c-treeACE (Advanced Core Engine). The implementations of FairCom technology are countless and range from embedded installations in mission-critical financial systems, such as for example VISA Europe. In its VDPS credit card authorization system - based on c-treeACE server - have passed and been approved in 2012 1.23 trillion Euros of purchases with VISA credit cards across Europe - with a goal to 2 trillion Euros in 2015 - and a total of 19 billion transactions all handled by FairCom c-treeACE.

# Running a COBOL Program

Now that the ACUCOBOL-GT runtime supports c-treeRTG COBOL Edition, let's see at once how to run a program written in COBOL using this efficient tool. First, copy the source code—in our case *cobol_Tutorial1.cbl*—into the ACUCOBOL-GT bin directory:

```
copy \FairCom\V10.x.x.COBOL\win32\ctree.cobol\tutorials\cobol_Tutorial1.cbl C:\AcuGT\bin
```

The ACUCOBOL-GT compiler needs to be started as shown below:

```
cd C:\AcuGT\bin
ccbl32.exe -fx cobol_Tutorial1.cbl
```

Specifically, the -fx option forces the compiler to generate an XFD file that defines the data structure used by *cobol_Tutorial1*. Subsequently, we will use the XFD file just generated to show the features of the SQL support of c-treeRTG COBOL Edition. At this point, before executing the *cobol_Tutorial1*, it is necessary to define the DEFAULT_HOST environment variable. This variable selects which file system will be used by the ACUCOBOL-GT runtime:

```
set DEFAULT_HOST=CTREE
```

Finally, the *cobol_Tutorial1* program can be run by invoking the ACUCOBOL-GT runtime as follows (Fig. 3):

```
wrun32.exe cobol_Tutorial1.acu
```

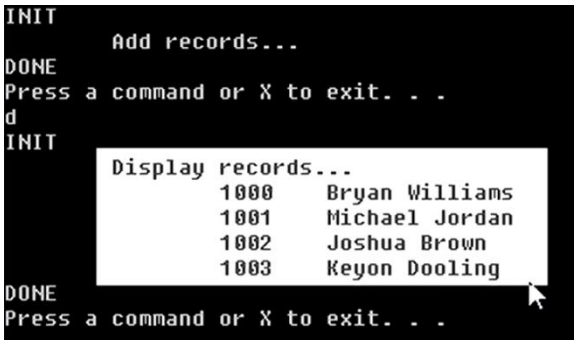


**Fig. 3 –The COBOL program running**

# Access to Data through SQL

The *cobol_Tutorial1* program creates an indexed data file called *CUSTMAST*. This file can be opened through simple queries using c-treeRTG SQL. To open *CUSTMAST* via c-treeRTG SQL follow the steps below. This method uses a command-line interactive tool—specifically **isql**—which allows to execute the query on the data managed by c-treeRTG SQL. Let's see how to proceed. First, import the *custmast.dat* data file and the previously generated XFD file through the **ctutil** tool. In particular, invoke **ctutil** with the *-sqlize* parameter, this will convert the XFD file in XDD format and put it in the header of the data file, then, the latter will be automatically imported into the database specified in the parameters. Here is the command to execute:

```
ctutil.exe -sqlize custmast custmast.xfd ADMIN ctreeSQL
```

At this point, we have to start ISQL:

```
\FairCom\V10.x.x.COBOL\win32\tools\cmdline\isql.exe -u admin -a ADMIN ctreeSQL
```

Finally—from the ISQL interface—the data in the custmast table can be visualized by executing a simple SQL SELECT instruction (Fig. 4):

```
ISQL> SELECT * FROM CUSTMAST;
custmast.xdd
<?xml version="1.0" encoding="US-ASCII"?>
<table name="CUSTMAST-FILE" maxRecLen="157" minRecLen="157">
 <key segCount="1" duplicate="false">
  <segment offset="0" size="4"/>
  <part name="CM_CUSTNUMB" offset="0" size="4"/>
 </key>
 <schema name="CUSTMAST-FILE" size="157">
  <field name="CM_CUSTNUMB" size="4" type="Alphanum" digits="4" scale="0"/>
  <field name="CM_CUSTZIPC" size="9" type="Alphanum" digits="9" scale="0"/>
  <field name="CM_CUSTSTAT" size="2" type="Alphanum" digits="2" scale="0"/>
  <field name="CM_CUSTRTNG" size="1" type="Alphanum" digits="1" scale="0"/>
  <field name="CM_CUSTNAME" size="47" type="Alphanum" digits="47" scale="0"/>
  <field name="CM_CUSTADDR" size="47" type="Alphanum" digits="47" scale="0"/>
  <field name="CM_CUSTCITY" size="47" type="Alphanum" digits="47" scale="0"/>
 </schema>
</table>
```

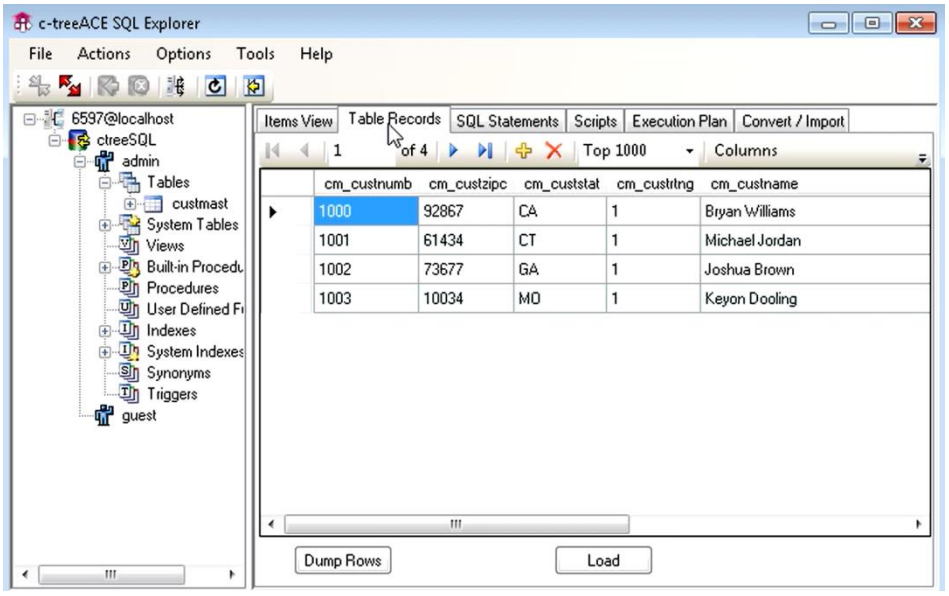


**Fig. 4 – Viewing table data via c-treeACE Explorer**

# A Particular Method

Below is a file descriptor example—typically included within the source of a COBOL program—that uses the redefines method to handle both the header (i.e. ORDINI_TESTATE) and the lines (i.e. ORDINI_RIGHE) of a command in the same table:

```
 FD  ORDINI
     LABEL RECORD IS STANDARD.
 01 REC-ORDINI.
     02 ORDINE-CHIAVE.
         03 ORDINE-CLIENTE   PIC  9(04) USAGE IS COMP-1.
         03 ORDINE-SEQ       PIC  9(04) USAGE IS COMP-3.
         03 ORDINE-RIGA      PIC  9(03) USAGE IS COMP-3.
$xfd  WHEN ORDINE-RIGA = 0 TABLENAME="ORDINI_TESTATE"
     02 ORDINE-TESTA.
         03 ORDINE-INDIRIZZO  PIC X(200).
         03 ORDINE-DATA       PIC X(8).
$xfd  WHEN ORDINE-RIGA != 0 TABLENAME="ORDINI_RIGHE"
     02 ORDINE-RIGA REDEFINES ORDINE-TESTA.
         03 ORDINE-ARTICOLO       PIC X(04).
         03 ORDINE-DESCRIZ        PIC X(40).
         03 ORDINE-QTA            PIC  9(4) USAGE IS COMP-6.
         03 ORDINE-PREZZO         PIC  9(5)V9(2) USAGE IS COMP-3.
         03 ORDINE-IVA            PIC  9(02) USAGE IS COMP-6.
```

After compiling using the -fx option seen before and after running the utility included in the distribution of c-treeRTG COBOL Edition (**ctutil** with the *-xfd2xdd* parameter), a file with a XDD extension is obtained and it defines the structure of the records for SQL by dividing it in two separate tables (ORDINI_TESTATE and ORDINI_RIGHE), both managed by SQL:

```
<?xml version="1.0" encoding="US-ASCII"?>
<table name="ORDINI" minRecLen="215"
maxRecLen="215">
 <key duplicate="false" primary="true">
  <segment offset="0" size="7"/>
  <part name="ORDINE_CLIENTE" offset="0" size="2"
/>
  <part name="ORDINE_SEQ" offset="2" size="3" />
  <part name="ORDINE_RIGA" offset="5" size="2" />
 </key>
 <filters>
  <field name="ORDINE_RIGA" offset="5" size="2"
type="PackedPositive" digits="3" scale="0" />
  <filter number="1" table="ORDINI_TESTATE">
   <eq>
    <field>ORDINE_RIGA</field>
    <value>"0"</value>
   </eq>
  </filter>
  <filter number="2" table="ORDINI_RIGHE">
   <neq>
    <field>ORDINE_RIGA</field>
    <value>"0"</value>
   </neq>
  </filter>
 </filters>
 <schema name="ORDINI_TESTATE" size="215" filter="1">
  <field name="ORDINE_CLIENTE" indexed="true" size="2" type="BinarySigned" digits="4"
scale="0" />
  <field name="ORDINE_SEQ" indexed="true" size="3" type="PackedPositive" digits="4"
scale="0" />
  <field name="ORDINE_RIGA" indexed="true" size="2" type="PackedPositive" digits="3"
scale="0" />
  <field name="ORDINE_INDIRIZZO" size="200" type="Alphanum" digits="200" scale="0" />
  <field name="ORDINE_DATA" size="8" type="Alphanum" digits="8" scale="0" />
```

**COBOL STILL IN VOGUE!**

Another example of the use of c-treeACE is found in the world of COBOL and regards the implementation of the file system in the solution FairCom isCOBOL of Veryant (www.veryant.com). isCOBOL is an Italian technology that offers an innovative solution, made entirely in JAVA, able to "rejuvenate" legacy COBOL applications, ensuring portability, scalability, efficiency and low cost of development.
Veryant is having considerable success, especially in the United States—where several hundreds of customers and tens of thousands of installations are counted—or as in northern Europe—where hundreds of completely renovated software applications are operating at thousands of companies.

```
 </schema>
 <schema name="ORDINI_RIGHE" size="215" filter="2">
  <field name="ORDINE_CLIENTE" indexed="true" size="2" type="BinarySigned" digits="4"
scale="0" />
  <field name="ORDINE_SEQ" indexed="true" size="3" type="PackedPositive" digits="4"
scale="0" />
  <field name="ORDINE_RIGA" indexed="true" size="2" type="PackedPositive" digits="3"
scale="0" />
  <field name="ORDINE_ARTICOLO" size="4" type="Alphanum" digits="4" scale="0" />
  <field name="ORDINE_DESCRIZ" size="40" type="Alphanum" digits="40" scale="0" />
  <field name="ORDINE_QTA" size="2" type="PackedUnsigned" digits="4" scale="0" />
  <field name="ORDINE_PREZZO" size="4" type="PackedPositive" digits="7" scale="2" />
  <field name="ORDINE_IVA" size="1" type="PackedUnsigned" digits="2" scale="0" />
  <field name="FILLER58" hidden="true" size="157" type="Alphanum" digits="157" scale="0"
/>
 </schema>
</table>
```

This single COBOL table, in the c-treeRTG SQL Explorer will be displayed as two separate tables (Fig. 5 and Fig 6). The modernization of COBOL program ends here! For those who want to dive in deeper, the complete documentation is available on the FairCom website (see http://www.faircom.com/ace/rtg_cobol_replace_your_file_system_t.php).

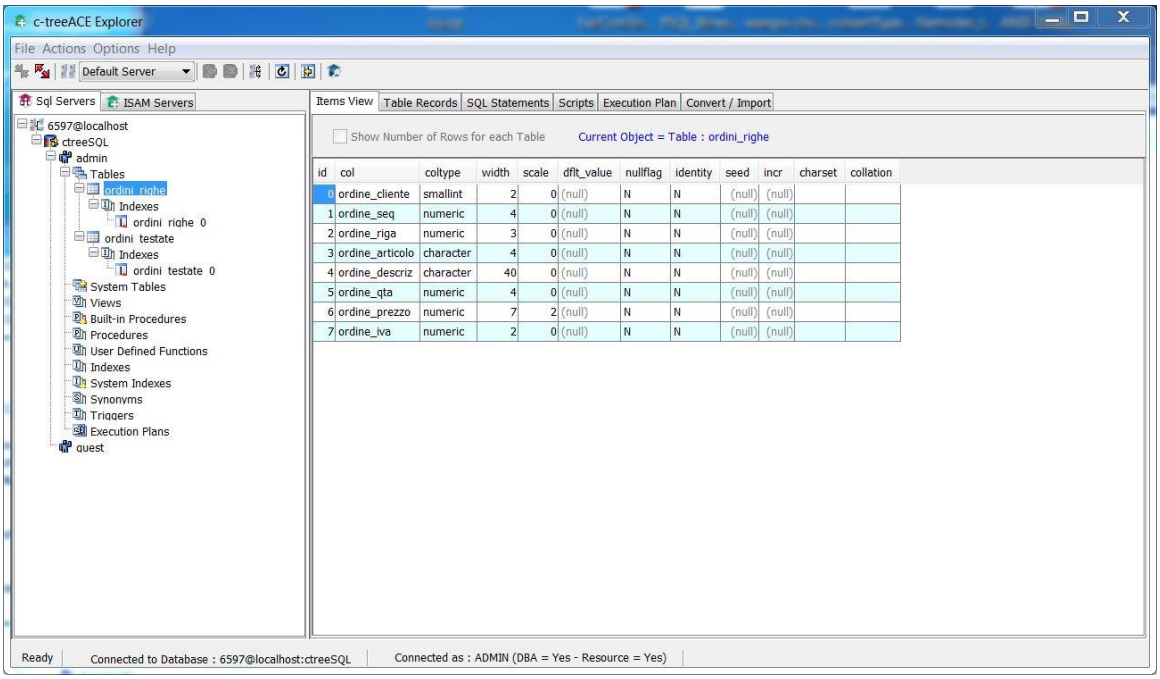


**Fig. 5 – The ORDINI_TESTATE table contents**

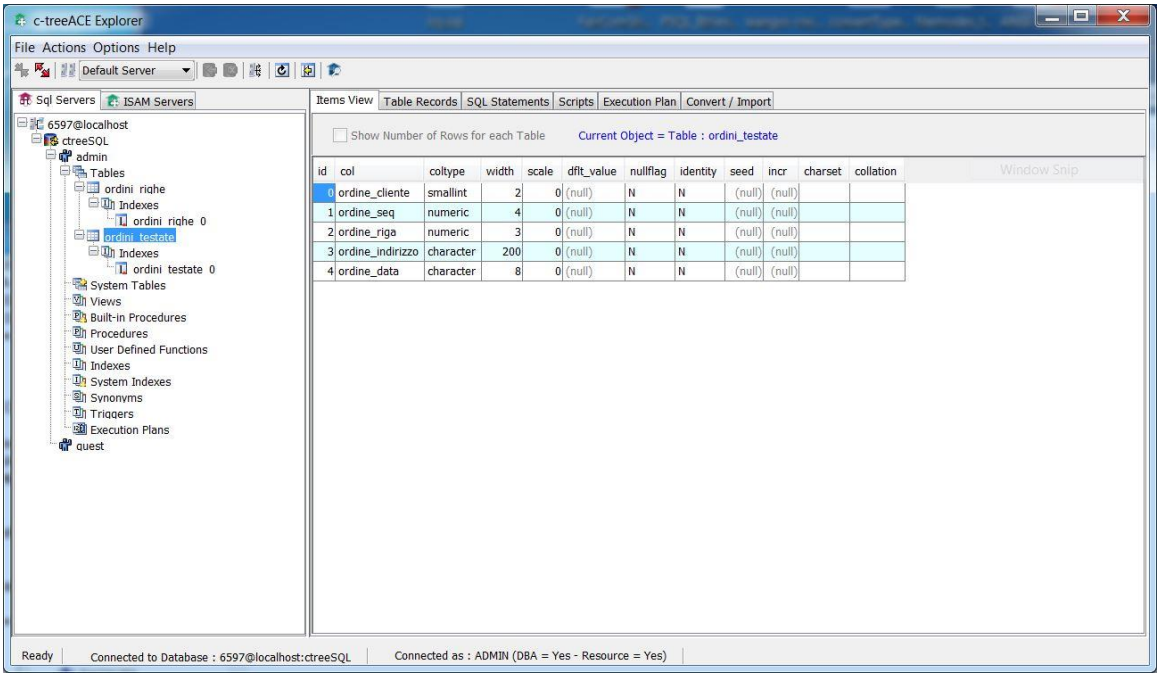


**Fig. 6 – The ORDINI_RIGHE table contents**