ALBI Investment Group Services (Pty) Ltd t/a AIGS
Byls Bridge Office Park, Building 14, Block B
Corner Olievenhoutbosch Road / Jean Avenue, Centurion
South Africa
PostNet Suite 447, Private Bag X033, Rivonia, 2128
Tel: +27 (0)11 267 8600
Fax +27 (0)11 465 0385

Table Partitioning

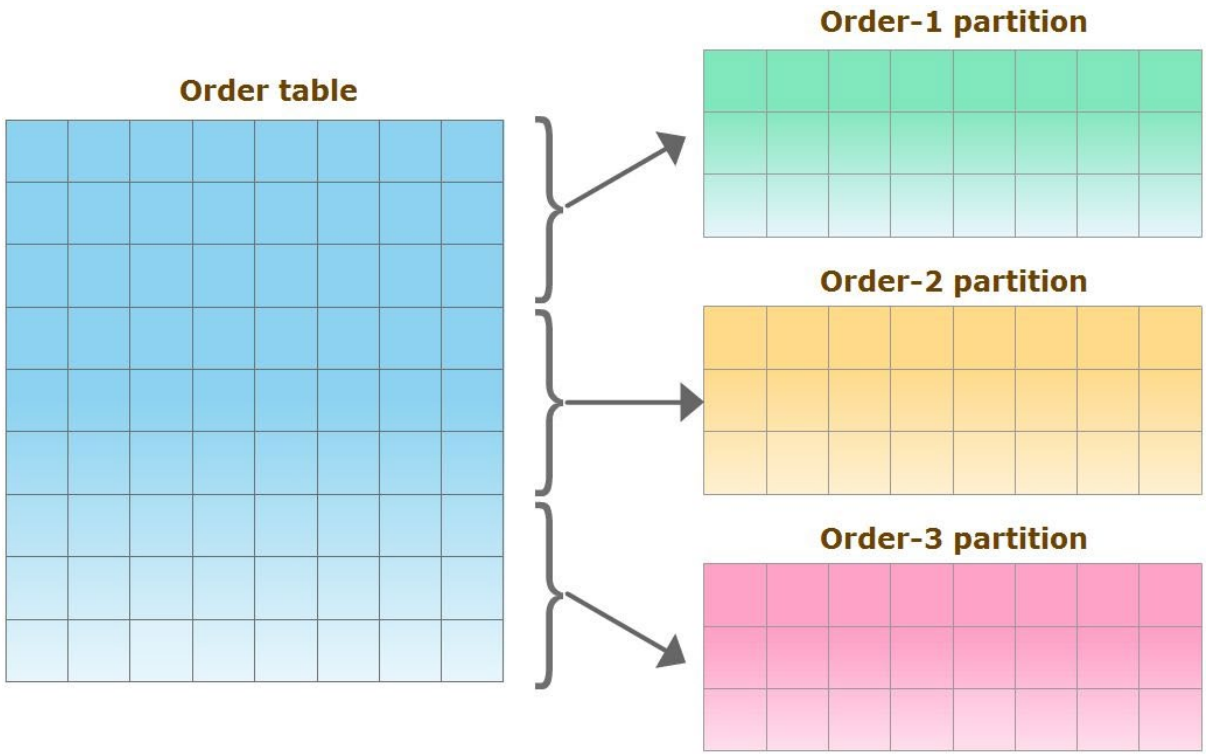
Date: May 2025

Table Partitioning is a separate product for Enterprise databases that allows you to divide large tables into smaller, logically-related parts called **partitions** .

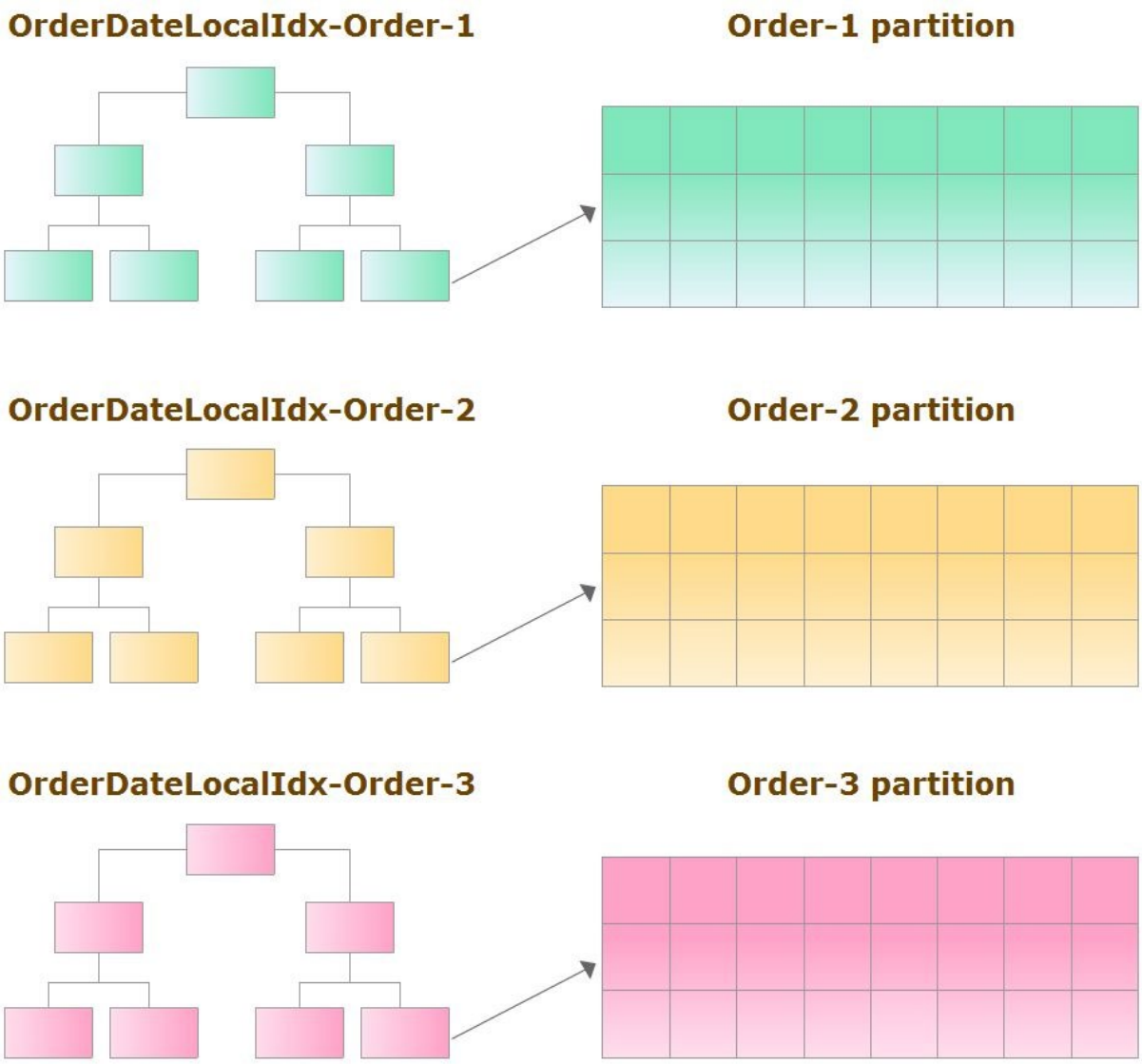
Partitions are implemented at the database level and transparent to the application. Using partitioned tables may require little or no changes to application code.

Note: Partitioned objects must be in a Type II *storage area*.

Partitioning has several important features. Partitions in a table are independent, so they can be accessed simultaneously, improving query performance. If one partition in a table is unavailable, the remaining partitions are still available to the application.



Each record of a partitioned table has the same columns.
Each record can only be in one partition.
Each partition can be in its own storage area.
Each partition can be independently modified and managed without affecting other partitions of the same table.



In addition:
Indexes associated with partitioned tables can also be partitioned (so-called local indexes). Like table partitions, each local index can be in its own storage area and managed independently. In addition to local indexes, there can also be global indexes for all data in a table, regardless of the partitions.

When is it worth using this technology?

For example, for tables containing historical data that must be archived. An interesting feature of partitions is that if a partition contains data that cannot be modified, it can be set **to read-only**. This way, only current data can be modified in the table, not archived data.

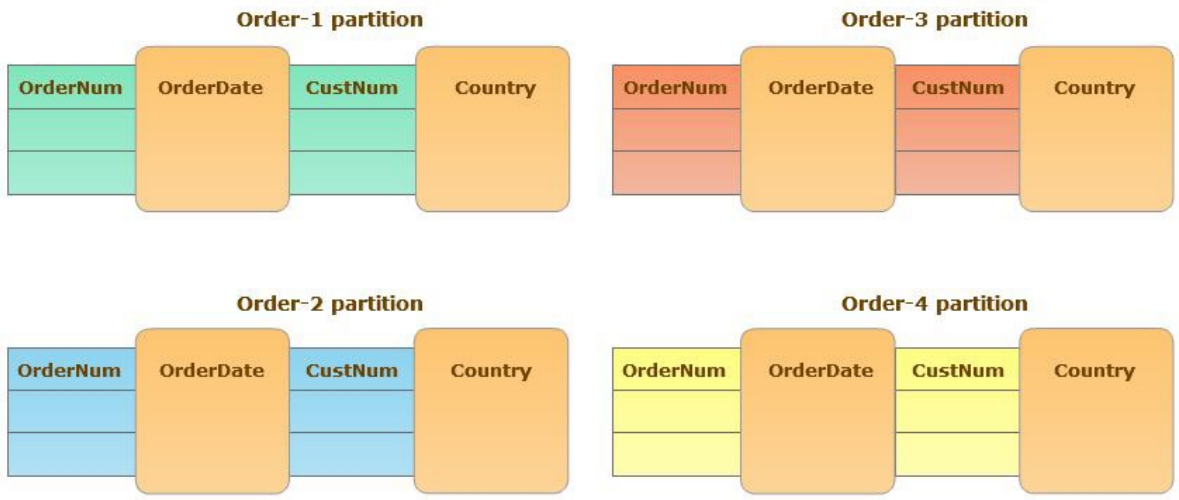
Other examples of causes:

Tables containing data that must be spread across different storage devices.
Large tables that must undergo periodic operations on records and indexes.
Tables containing columns by which data can be logically grouped.
Tables containing data that is frequently queried with the phrase TABLE-SCAN rather than

WHOLE-INDEX.
Tables that will grow to very large sizes.

Partitioning uses one column (**the partition key**) to uniquely identify each partition. The partitioning can be by range (e.g. a date range) or by list (a list of distinct column values, e.g. countries, states, etc.). The *partition key* column cannot have undefined values ("?").

It should be added that the data in the partition can be divided into further partitions (15 levels of division are possible). For example, we divide orders (order table) by date range, and then we divide each partition additionally into sub-partitions by country.



The way tables are divided is described in the so-called partition policy, found in the database meta-schema.

In the previous part we got to know a bit of the theory about OpenEdge table partitioning. Now it's time to do something practical.

We create a new database, e.g. **mytp**, a copy of the **sports2000** database:

prodb mytp sports2000

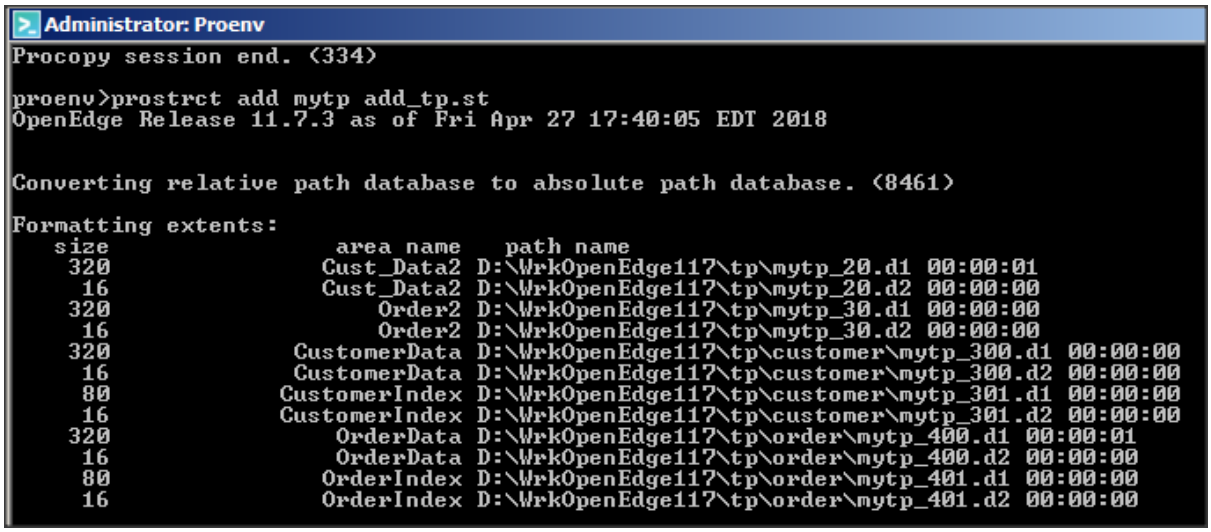
For table partitioning, the data must be in Type II areas. The sports2000 demo database has only Type I areas. We add areas to which we will move **the Customer (Cust_Data2)** and **Order (Order2)** tables, as well as areas where the data will be stored after the partitioning process.

First, we create the **add_tp.st file**

```
d "Cust_Data2":20,64;8 . f 1280
d "Cust_Data2":20,64;8 .
#
d "Cust_Index2":25,32;8 . f 1280
d "Cust_Index2":25,32;8 .
#
d "Order2":30,64;8 . f 1280
d "Order2":30,64;8 .
#
d "CustomerData":300,64;8 ./customer f 1280
d "CustomerData":300,64;8 ./customer
d "CustomerIndex":301,32;8 ./customer f 320
d "CustomerIndex":301,32;8 ./customer
#
d " OrderData " : 400 . 64 ; 8 . / order f 1280
d "OrderData":400.64;8./order
d "OrderIndex":401,32;8 ./order f 320
d "OrderIndex":401,32;8 ./order
#
```

In the database directory, we create subdirectories **customer** and **order**.
Now we run the command:

prostrct add mytp add_tp.st



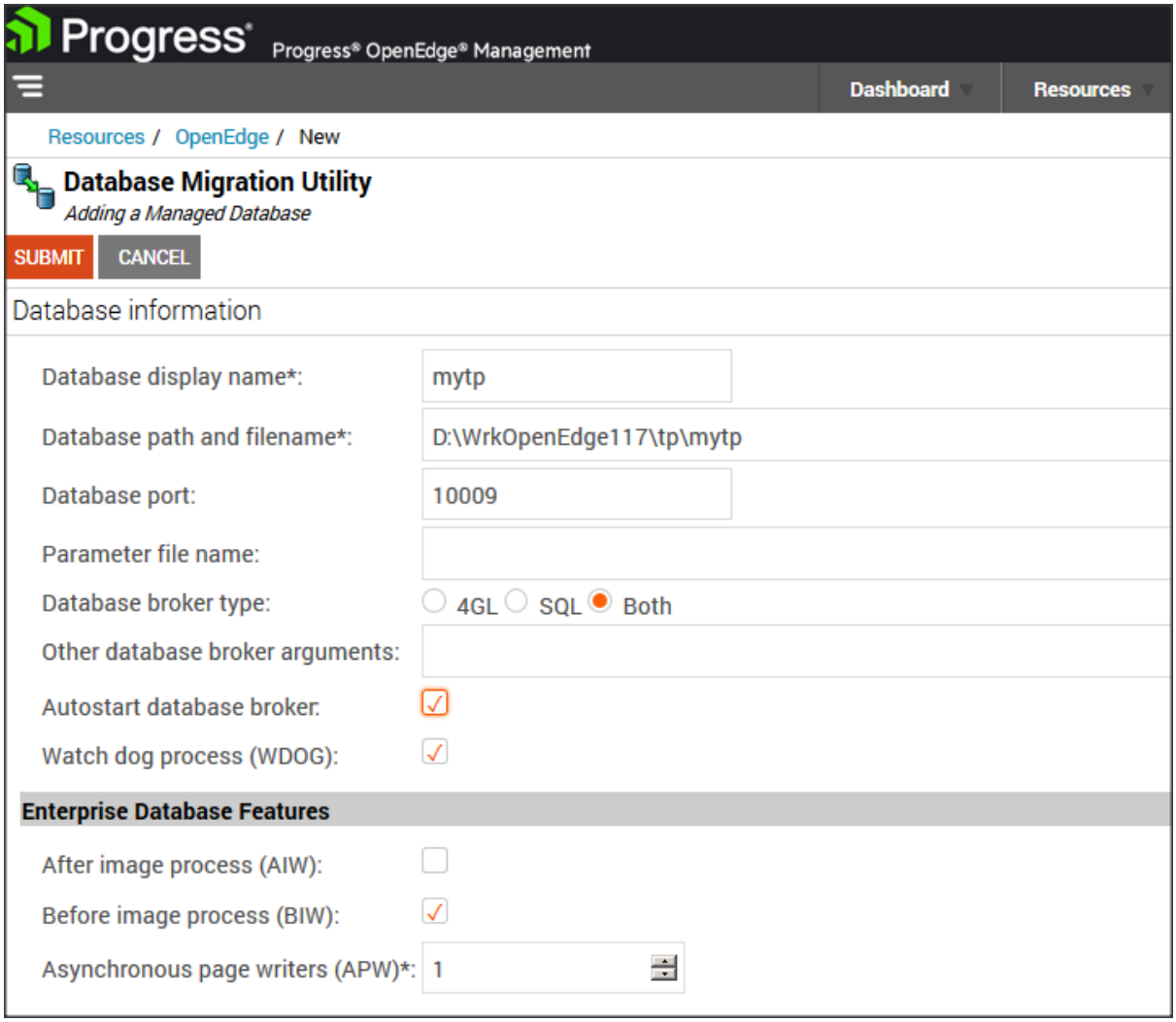
We are moving the Customer and **Order** tables to new Type II areas:

proutil mytp -C tablemove customer Cust_Data2

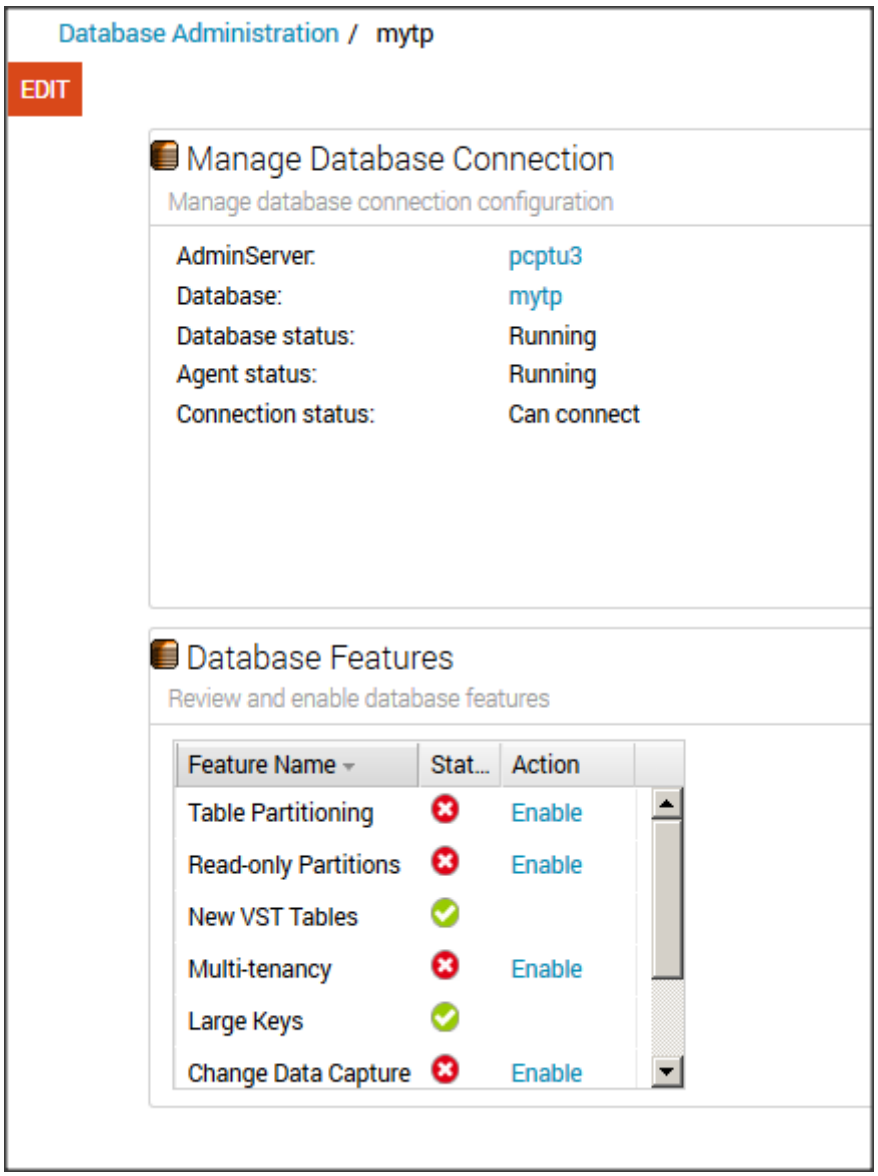
proutil mytp -C tablemove order Order2

Unfortunately, we also need to move all indexes to Type II areas (I won't show this here, but it's also a simple task). *The demo database can be prepared in various ways, e.g. by deleting unnecessary objects or dump and load to a new structure.*

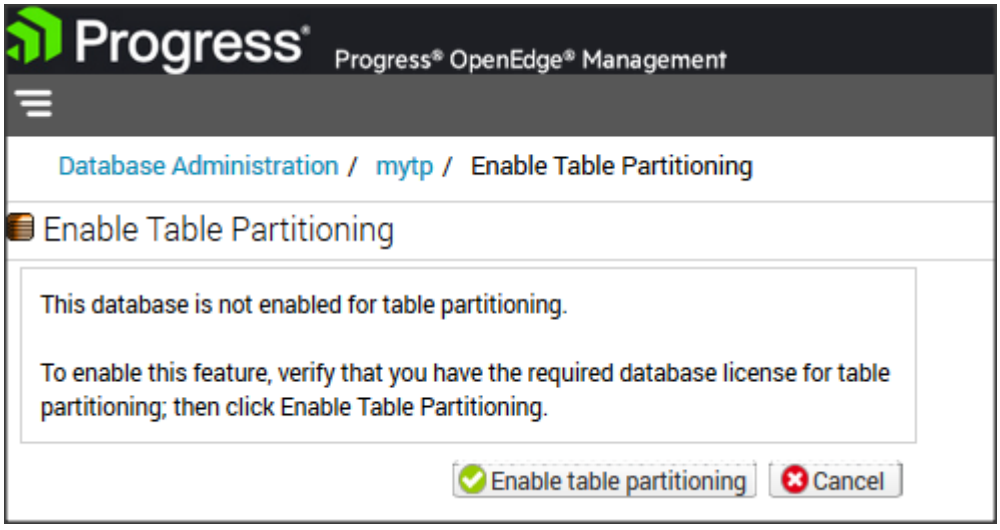
As in the case of CDC, we will add the database server configuration in OE Management (or OE Explorer). After logging in, we select **Resources -> Database. The Database Migration Utility** view appears, in which we provide the parameters of the created mytp database along with the port number, e.g. **1009**. We select **Autostart database broker**.



On the top bar of OE Management (OE Explorer) click **Database Administration** and on the database name: **mytp**. The following screen appears.

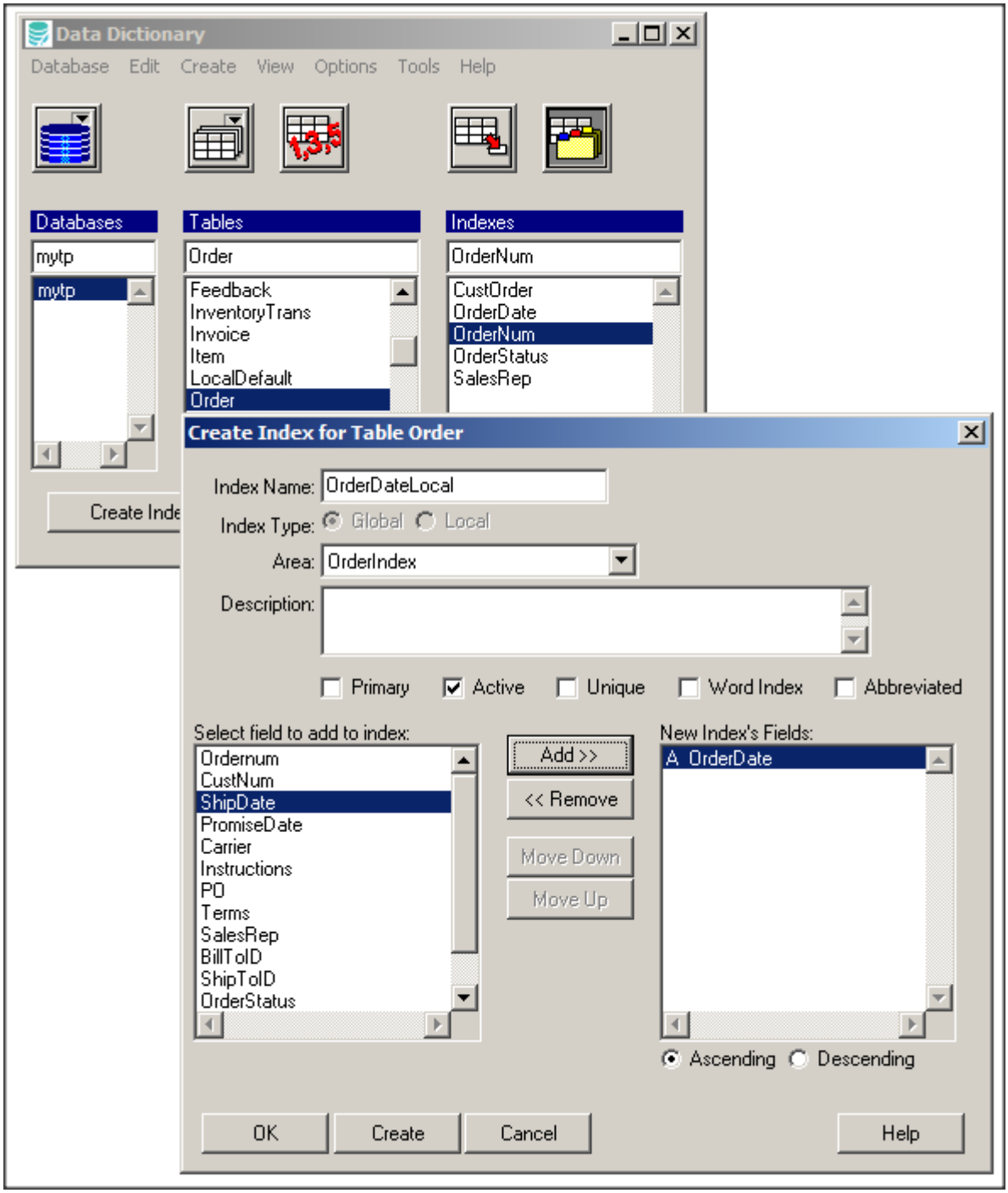


In the **Database Features** section, click Table Partitioning **Enable** ...

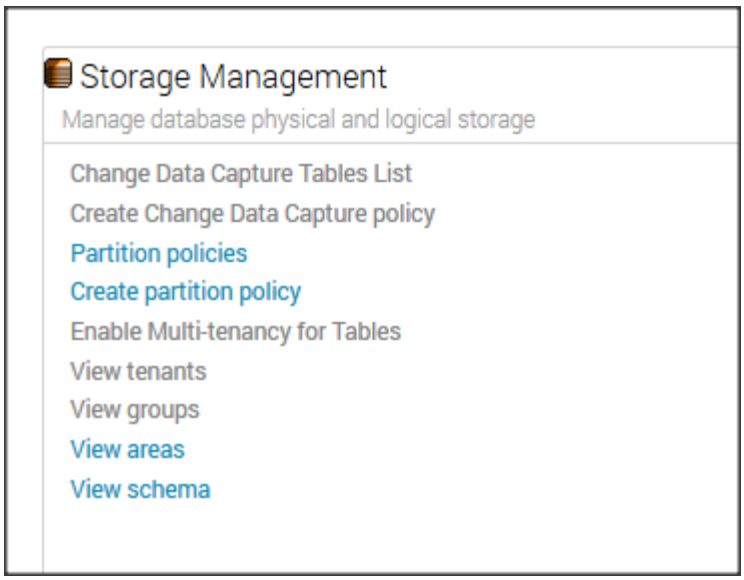


... and also, in **Enable table partitioning**. The function is already enabled in the database.

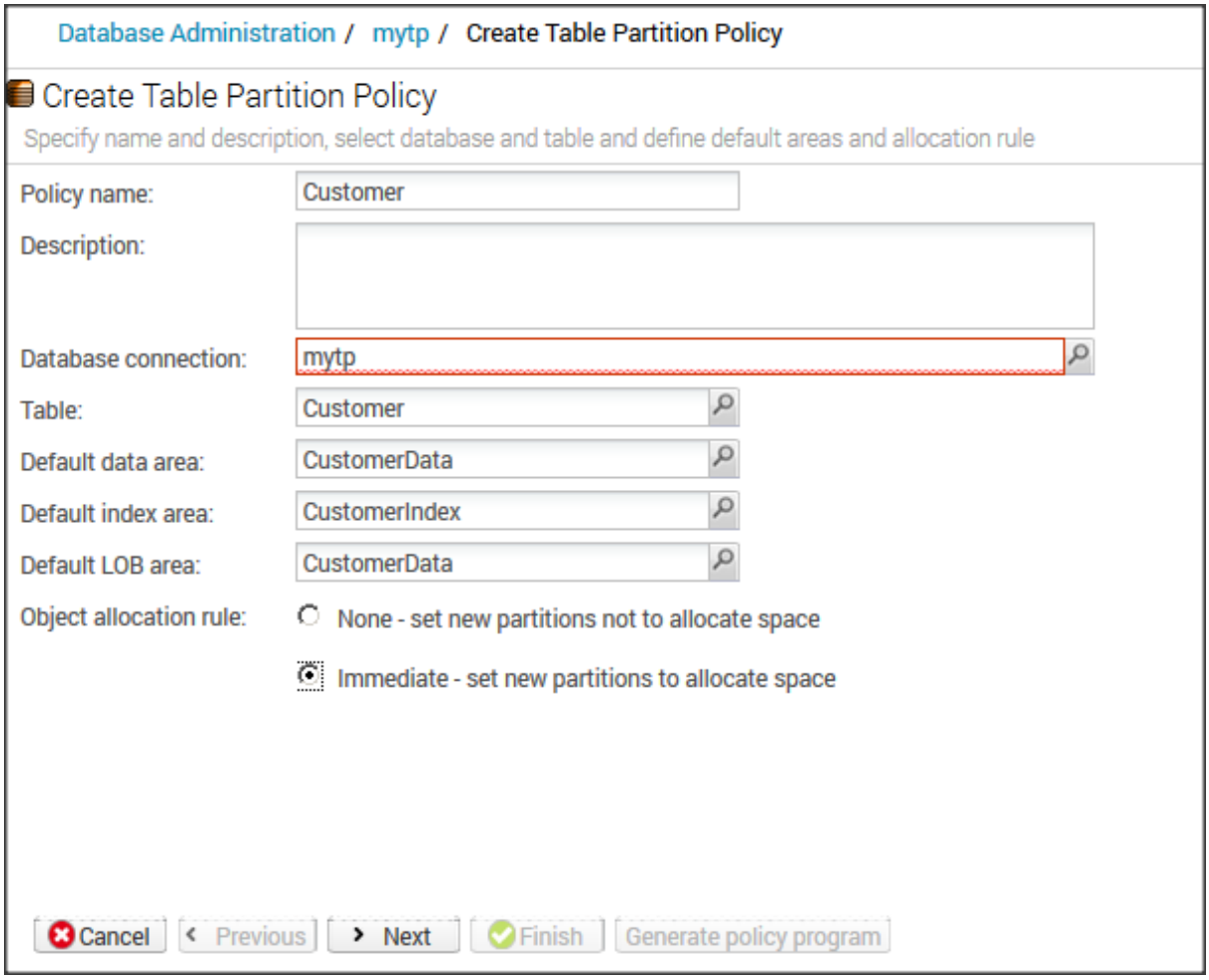
In the database dictionary we add for the **Order** table index: **OrderDateLocal**, area: **OrderIndex** field: **OrderDate**. Similarly for the **Customer** table index: **CustomerCountryLocal**, area: **CustomerIndex**, field: **Country**.



We return to OE Explorer. In the **Storage Management** section, click **Create partition policy**.

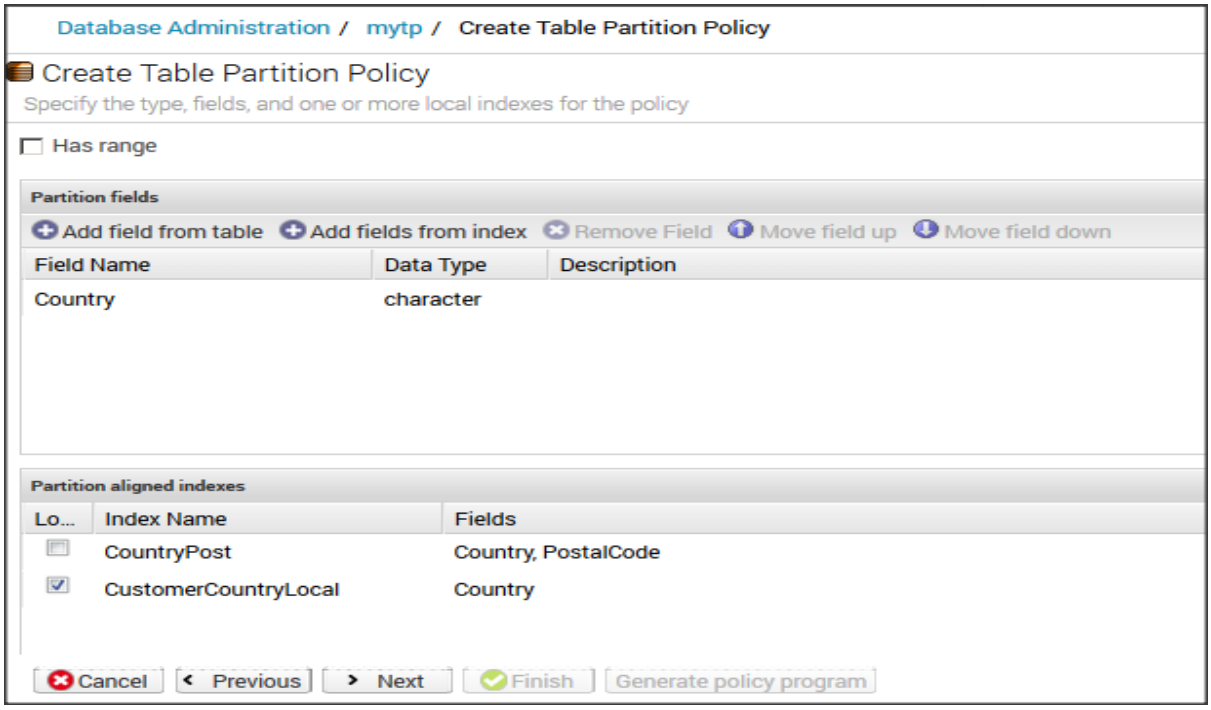


The following screen appears, where we define the settings for our first partition for the Order table. We click **Create partition policy**. We create a partition for the Customer table.

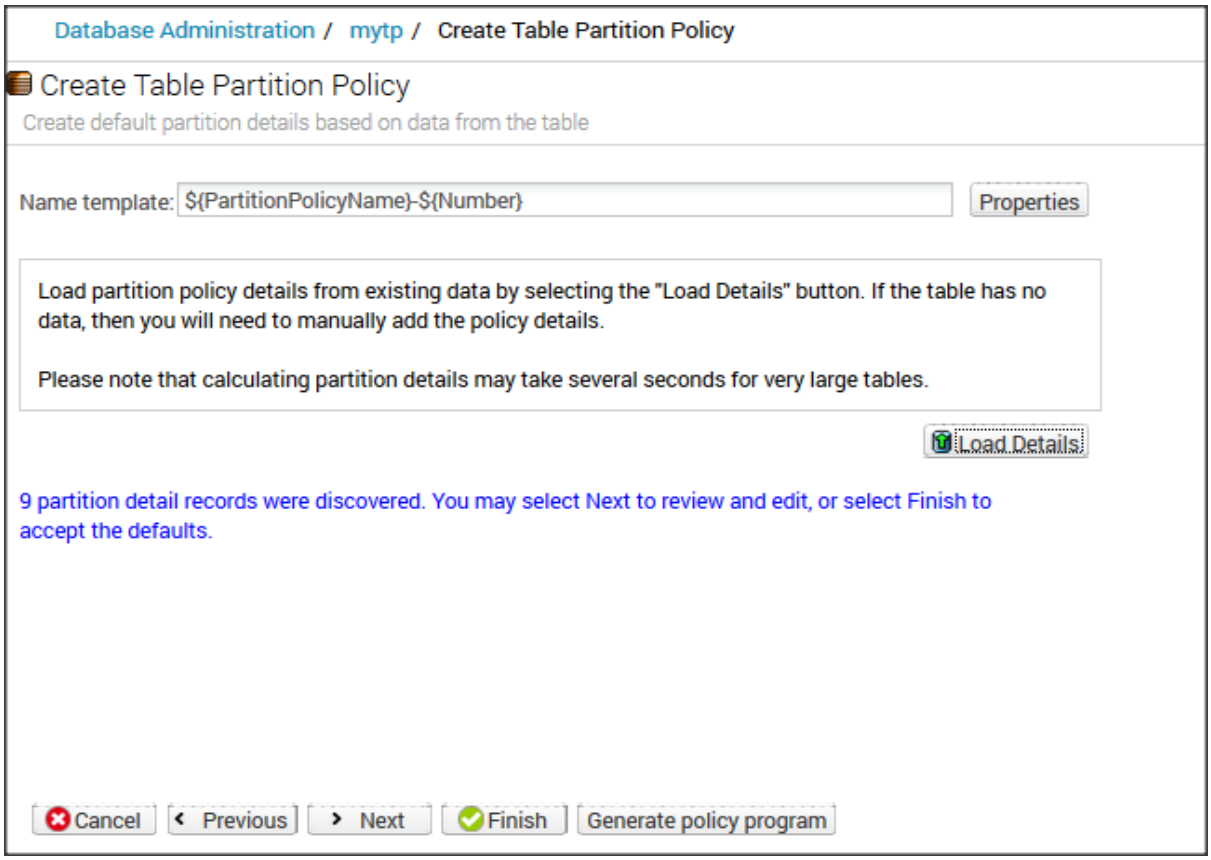


We fill in the data as above (only tables in Type II areas are displayed in lookups), we select: **Immediate – set new partitions to allocate space**. We click **Next**.

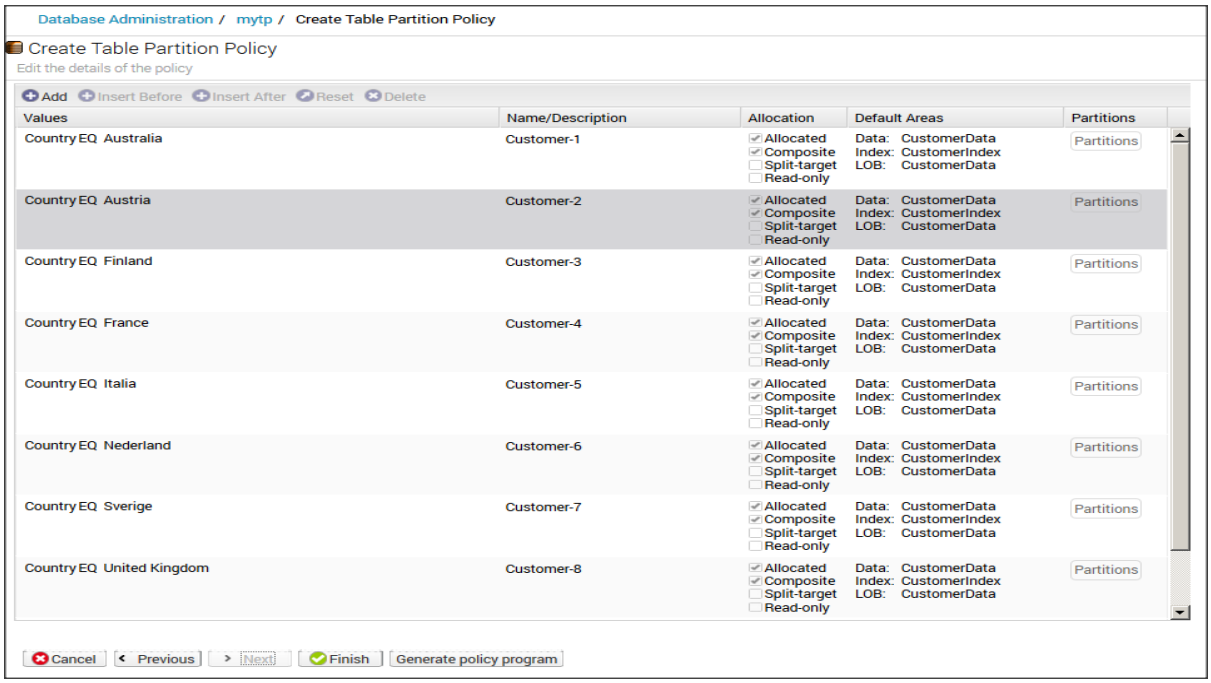
This will be a partition of type **List**, so we **do NOT check the Has range** option. We click **Add fields from index**, select the newly created index **CustomerCountryLocal**. The partition will divide the data by the **Country** field.



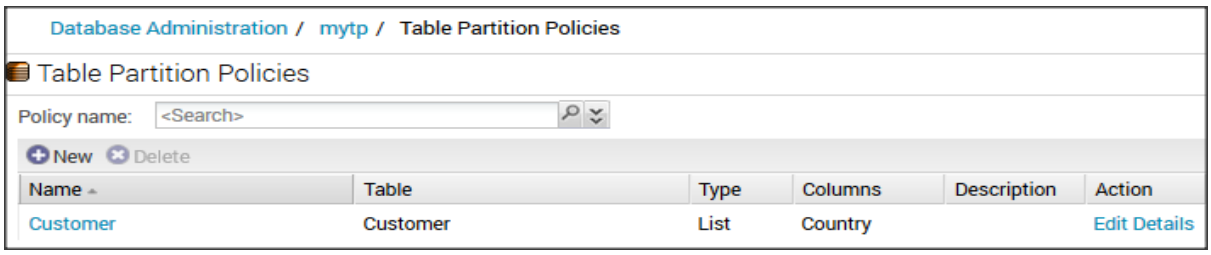
We click **Next** and **Load Details**. We get information that 9 subareas were found for our partition (this is related to the values of the Country field). We click **Next**.



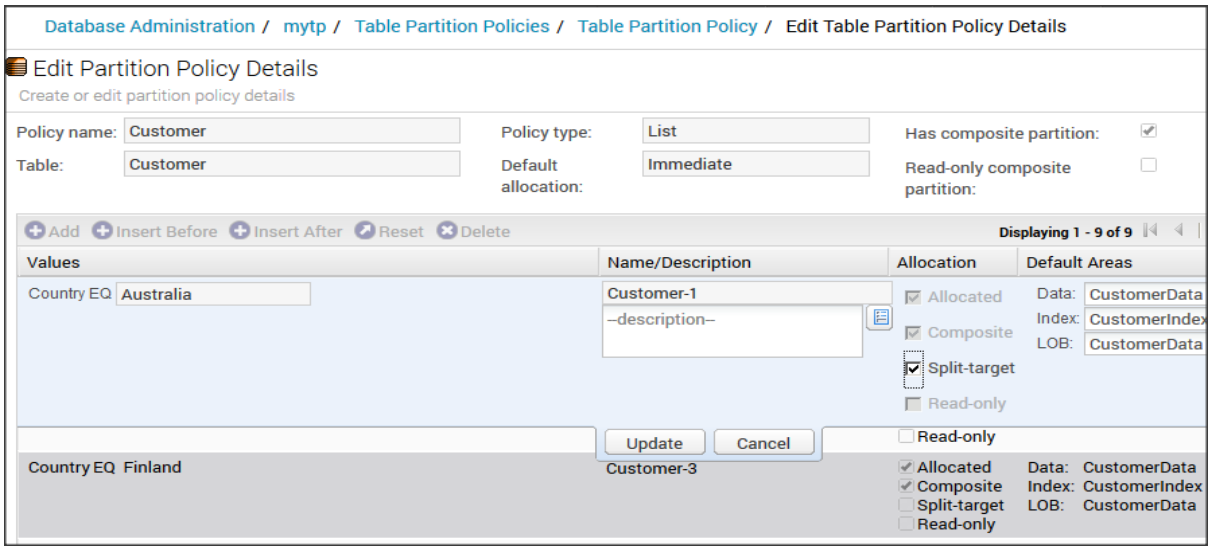
We see the details of **Table Partition Policy**. Click **Finish**.



A new rule called **Customer** is created. However, the data in the table is not yet split.



We prepare the data for migration. Click on **Edit Details,** entering the partitioning rule details again. Double-click on each detail, e.g. Austria, and select **Split-target**. Only the selected elements will be split. I select **Split-target** for all 9 elements (i.e. for all countries).



I choose **Commit**. Everything is ready to move data from the Composite partition to the Split-target partitions.

In the proenv command window, enter the command:

proutil mytp -C partitionmanage split table customer composite initial useindex

CustomerCountryLocal recs 100

```
proenv>proutil mytp -C partitionmanage split table customer composite initial useindex
ryLocal recs 100
OpenEdge Release 11.7.3 as of Fri Apr 27 17:40:05 EDT 2018
BEGIN: Split Operation For Table customer (17384)

             Source Partition initial[0]
             Target Partition Customer-3[1]
             Target Partition Customer-1[2]
             Target Partition Customer-2[3]
             Target Partition Customer-4[4]
             Target Partition Customer-5[5]
             Target Partition Customer-6[6]
             Target Partition Customer-7[7]
             Target Partition Customer-8[8]
             Target Partition Customer-9[9]

Index CustomerCountryLocal has been identified as the scanning index (useIndex).

A non-unique index has been selected as the useindex index.
Additional locking is required with the use of this index CustomerCountryLocal.

Number of Records per Transaction (recs): 100

Do you want to continue (y/n)?
y
       Target partition: Customer-1[2], records moved: 1.
       Target partition: Customer-2[3], records moved: 3.
       Target partition: Customer-3[1], records moved: 28.
       Target partition: Customer-4[4], records moved: 7.
       Target partition: Customer-5[5], records moved: 2.
       Target partition: Customer-6[6], records moved: 3.
       Target partition: Customer-7[7], records moved: 2.
       Target partition: Customer-8[8], records moved: 12.
       Target partition: Customer-9[9], records moved: 1059.
       Source partition: initial[0], contains no records.
       Total records processed: 1117.

END: Split Operation For Table customer[0]
Split Operation finished successfully. (17359)

proenv>
```

Data was split using the defined local index. Number of records in transaction: 100.

We can check where the Customer table data is now by running the well-known command: **proutil mytp -C tabanalys**

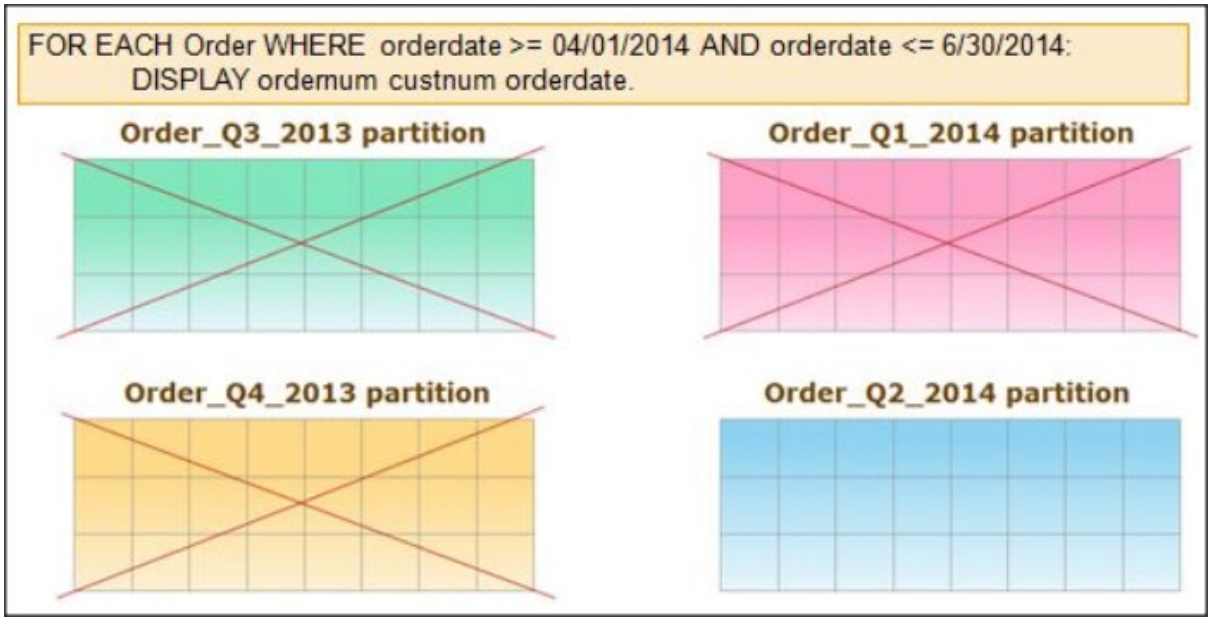
```
RECORD BLOCK SUMMARY FOR AREA "CustomerData": 300
------------------------------------------------------


RECORD BLOCK SUMMARY FOR SHARED TABLES
----------------------------------------------


                                                                  -Record Size (B)-
---Fragments--- Scatter
Table                                    Records    Size   Min   Max  Mean
Count Factor  Factor
PUB.Customer.Customer-1                        1  175.0B   175   175   175
1    1.0     1.0
PUB.Customer.Customer-2                        3  413.0B   132   145   137
3    1.0     1.0
PUB.Customer.Customer-3                       28    4.3K   140   198   157
28    1.0     1.0
PUB.Customer.Customer-4                        7    1.1K   129   199   164
7    1.0     1.0
PUB.Customer.Customer-5                        2  257.0B   127   130   128
2    1.0     1.0
PUB.Customer.Customer-6                        3  406.0B   133   137   135
3    1.0     1.0
PUB.Customer.Customer-7                        2  250.0B   121   129   125
2    1.0     1.0
PUB.Customer.Customer-8                       12    2.0K   139   225   168
12    1.0     1.0
PUB.Customer.Customer-9                     1059  138.2K    98   175   133
1059    1.0     1.0

                                  ----------------------------------------------
--------------------------
Subtotals:                                  1117  147.1K    98   225   134
1117    1.0     1.0
```

The analysis for **the CustomerData** area shows the table divided into 9 partitions.

Now that we finished dividing the table based on the values of the Country field (local index). I mentioned that partitioning is transparent for the application, but there may be situations when the query returns different records as before partitioning. One example is using RECID/ROWID address search in the code. Because partitioning moves records, their physical addresses change. In the case of using this solution, appropriate corrections must be made in the application code.

Another more common example is using a filter on a given field (WHERE) in a query. Since a new local index is usually added to a partitioned table, the search can be performed based on it. To be sure that the index has not been changed, and to maintain the same order of records, you may need to add the phrase USE-INDEX.

It is necessary to mention an important feature of partitioning technology, the so-called **pruning**.



The pruning process automatically analyses the query and (if possible) does not take into account records in partitions that do not meet the query conditions. The figure shows an example of dividing records into quarters according to the OrderDate field. If the conditions in the query apply only to records from the second quarter, then only this one partition will be used. This technology can significantly improve performance.

The next issue is the use of the TABLE-SCAN phrase. It causes the records to be retrieved without access to indexes and displayed in the order in which they are in the database blocks. The order of the records will therefore change after partitioning, due to their transfer to other blocks. Before retrieving, the pruning operation is performed. Please see the following example:

```
FOR EACH order WHERE country = "USA" AND
  OrderDate >= 10-01-2014 AND
  OrderDate <= 12-31-2014 TABLE-SCAN:
DISPLAY Order No...
```

As you can see, table partitioning can affect the order in which records are processed. Now let's see what the situation is with creating new records and editing them.

It is important to remember that records must have specific values for the fields according to which partitioning is performed. Therefore, it is worth using the ASSIGN statement, which groups the value settings for the record, right after the CREATE statement. The ASSIGN statement forces the physical creation of a record in the database, and thus the operation of the porting mechanism. The following example will cause an error (partitioning by the Country field).

```
CREATE order.
ASSIGN ordernum = NEXT-VALUE(next-ord-num). /* ERROR! */
ASSIGN country = "USA".
```

This example can easily be improved to:

```
CREATE order.
ASSIGN ordernum = NEXT-VALUE(next-ord-num)
       country = "USA".
```

Editing the record field according to which the table is divided involves moving the entire record to another area This involves deleting the record, creating it in a new area, updating indexes. Too frequently, such operations can result in reduced performance. This may be the result of an incorrectly selected partition key, which is worth discussing and possibly choosing a different key.