Progress®



OpenEdge® Getting Started:
Table Partitioning

OpenEdge®

Table of Contents

Copyright....5

Preface....7

Purpose....7
Audience....7
Organization....8
Using ABL documentation....8
References to ABL compiler and run-time features....8
References to ABL data types....8
Typographical conventions....9
Examples of syntax descriptions....10
Long syntax descriptions split across lines....12
Complex syntax descriptions with both required and optional elements....12
OpenEdge messages....12
Obtaining more information about OpenEdge messages....13

Chapter 1: Introducing Table Partitioning....15

Table partitioning details....15
Table partition types....16
Defining partitions....18
Defining a partition policy....19
Defining a partition policy detail....20
Table partitioning restrictions....21
Indexes for table partitions....21
Getting started partitioning your database....22

Chapter 2: Table Partitioning Planning....23

Initial planning....23
Data Considerations....24
Data access (query) considerations....25
Maintenance considerations....25
Partition column selection....26
Next steps....26

Chapter 3: Table partitioning capabilities 27

Table partition allocation....27
Managing Table Partitioning in OpenEdge Management or OpenEdge Explorer....28
ABL Application considerations 28

Table partitioning API references....29
SQL considerations....30
Maintenance operations....31

Chapter 4: Documentation map....33

Additional table partitioning information....33

Chapter 5: Glossary of Terms....35

Terms....35

Index....39

Copyright

© **2019 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.**

These materials and all Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. The information in these materials is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear therein. The references in these materials to specific platforms supported are subject to change.

Corticon, DataDirect (and design), DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, DataRPM, Deliver More Than Expected, Icenium, Kendo UI, Kinvey, NativeScript, OpenEdge, Powered by Progress, Progress, Progress Software Developers Network, Rollbase, SequeLink, Sitefinity (and Design), Sitefinity, SpeedScript, Stylus Studio, TeamPulse, Telerik, Telerik (and Design), Test Studio, and WebSpeed are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries. Analytics360, AppServer, BusinessEdge, DataDirect Autonomous REST Connector, DataDirect Spy, SupportLink, DevCraft, Fiddler, JustAssembly, JustDecompile, JustMock, Kinvey Chat, NativeChat, NativeScript Sidekick, OpenAccess, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, and WebClient are trademarks or service marks of Progress Software Corporation and/or its subsidiaries or affiliates in the U.S. and other countries. Java is a registered trademark of Oracle and/or its affiliates. Any other marks contained herein may be trademarks of their respective owners.

February 2019

Last updated with new content: Release 12.0

Preface

For details, see the following topics:

- Purpose
- Audience
- Organization
- Using ABL documentation
- Typographical conventions
- Examples of syntax descriptions
- OpenEdge messages

Purpose

The purpose of this book is to provide an introduction to the Table Partitioning capabilities of the OpenEdge® RDBMS. The book provides an overview of the concepts, terms and definitions, descriptions of capabilities, and pointers to other, more detailed documentation.

Audience

This book is intended for anyone interested in a high-level understanding of Table Partitioning.

Organization

This book is organized into the following chapters:

- Introducing Table Partitioning on page 15

  Introduces Table Partitioning, discusses concepts and advantages.

- Table Partitioning Planning on page 23

  Discusses factors to consider when planning to partition an OpenEdge database.

- Table partitioning capabilities on page 27

  Introduces the feature capabilities and some of the tools created or enhanced for managing Table Partitioning.

- Documentation map on page 33

  Provides a map to other locations in the documentation that discuss topics specific to Table Partitioning in greater detail.

- Glossary of Terms on page 35

Using ABL documentation

OpenEdge provides a special purpose programming language for building business applications. In the documentation, the formal name for this language is *ABL (Advanced Business Language)*. With few exceptions, all keywords of the language appear in all `UPPERCASE`, using a font that is appropriate to the context. All other alphabetic language content appears in mixed case.

For the latest documentation updates go to https://docs.progress.com.

References to ABL compiler and run-time features

ABL is both a compiled and an interpreted language that executes in a run-time engine. The documentation refers to this run-time engine as the *ABL Virtual Machine (AVM)*. When the documentation refers to ABL source code compilation, it specifies *ABL* or *the compiler* as the actor that manages compile-time features of the language. When the documentation refers to run-time behavior in an executing ABL program, it specifies *the AVM* as the actor that manages the specified run-time behavior in the program.

For example, these sentences refer to the ABL compiler's allowance for parameter passing and the AVM's possible response to that parameter passing at run time: "ABL allows you to pass a dynamic temp-table handle as a static temp-table parameter of a method. However, if at run time the passed dynamic temp-table schema does not match the schema of the static temp-table parameter, the AVM raises an error." The following sentence refers to run-time actions that the AVM can perform using a particular ABL feature: "The ABL socket object handle allows the AVM to connect with other ABL and non-ABL sessions using TCP/IP sockets."

References to ABL data types

ABL provides built-in data types, built-in class data types, and user-defined class data types. References to built-in data types follow these rules:

- Like most other keywords, references to specific built-in data types appear in all UPPERCASE, using a font that is appropriate to the context. No uppercase reference ever includes or implies any data type other than itself.
- Wherever *integer* appears, this is a reference to the INTEGER or INT64 data type.
- Wherever *character* appears, this is a reference to the CHARACTER, LONGCHAR, or CLOB data type.
- Wherever *decimal* appears, this is a reference to the DECIMAL data type.
- Wherever *numeric* appears, this is a reference to the INTEGER, INT64, or DECIMAL data type.

References to built-in class data types appear in mixed case with initial caps, for example, Progress.Lang.Object. References to user-defined class data types appear in mixed case, as specified for a given application example.

Typographical conventions

This documentation uses the following typographical and syntax conventions:

Convention	Description
Bold	Bold typeface indicates commands or characters the user types, provides emphasis, or the names of user interface elements.
Italic	Italic typeface indicates the title of a document, or signifies new terms.
SMALL, BOLD CAPITAL LETTERS	Small, bold capital letters indicate OpenEdge key functions and generic keyboard keys; for example, **GET** and **CTRL**.
KEY1+KEY2	A plus sign between key names indicates a **simultaneous** key sequence: you press and hold down the first key while pressing the second key. For example, **CTRL+X**.
KEY1 KEY2	A space between key names indicates a **sequential** key sequence: you press and release the first key, then press another key. For example, **ESCAPE H**.
Syntax:	
Fixed width	A fixed-width font is used in syntax, code examples, system output, and file names.
Fixed-width italics	Fixed-width italics indicate variables in syntax.
Fixed-width bold	Fixed-width bold italic indicates variables in syntax with special emphasis.
UPPERCASE fixed width	ABL keywords in syntax and code examples are almost always shown in upper case. Although shown in uppercase, you can type ABL keywords in either uppercase or lowercase in a procedure or class.
Period (.) or colon (:)	All statements except DO, FOR, FUNCTION, PROCEDURE, and REPEAT end with a period. DO, FOR, FUNCTION, PROCEDURE, and REPEAT statements can end with either a period or a colon.

Convention	Description
[ ]	Large brackets indicate the items within them are optional.
[ ]	Small brackets are part of ABL.
{ }	Large braces indicate the items within them are required. They are used to simplify complex syntax diagrams.
{ }	Small braces are part of ABL. For example, a called external procedure must use braces when referencing arguments passed by a calling procedure.
|	A vertical bar indicates a choice.
. . .	Ellipses indicate repetition: you can choose one or more of the preceding items.

Examples of syntax descriptions

In this example, `ACCUM` is a keyword, and *`aggregate`* and *`expression`* are variables:

Syntax

```
ACCUM aggregate expression
```

`FOR` is one of the statements that can end with either a period or a colon, as in this example:

```
FOR EACH Customer NO-LOCK:
  DISPLAY Customer.Name.
END.
```

In this example, `STREAM` *`stream`*, `UNLESS-HIDDEN`, and `NO-ERROR` are optional:

Syntax

```
DISPLAY [ STREAM stream ] [ UNLESS-HIDDEN ] [ NO-ERROR ]
```

In this example, the outer (small) brackets are part of the language, and the inner (large) brackets denote an optional item:

Syntax

```
INITIAL [ constant [ , constant ] ]
```

A called external procedure must use braces when referencing compile-time arguments passed by a calling procedure, as shown in this example:

Syntax

```
{ &argument-name }
```

In this example, EACH, FIRST, and LAST are optional, but you can choose only one of them:

Syntax

```
PRESELECT [ EACH | FIRST | LAST ] record-phrase
```

In this example, you must include two expressions, and optionally you can include more. Multiple expressions are separated by commas:

Syntax

```
MAXIMUM ( expression , expression [ , expression ] ... )
```

In this example, you must specify MESSAGE and at least one *expression* or SKIP [(*n*)], and any number of additional *expression* or SKIP [(*n*)] is allowed:

Syntax

```
MESSAGE { expression | SKIP [ ( n ) ] } ...
```

In this example, you must specify {*include-file*, then optionally any number of *argument* or &*argument-name* = "*argument-value*", and then terminate with }:

Syntax

```
{ include-file
    [ argument | &argument-name = "argument-value" ] ... }
```

Long syntax descriptions split across lines

Some syntax descriptions are too long to fit on one line. When syntax descriptions are split across multiple lines, groups of optional and groups of required items are kept together in the required order.

In this example, WITH is followed by six optional items:

Syntax

```
WITH [ ACCUM max-length ] [ expression DOWN ]
  [ CENTERED ] [ n COLUMNS ] [ SIDE-LABELS ]
  [ STREAM-IO ]
```

Complex syntax descriptions with both required and optional elements

Some syntax descriptions are too complex to distinguish required and optional elements by bracketing only the optional elements. For such syntax, the descriptions include both braces (for required elements) and brackets (for optional elements).

In this example, ASSIGN requires either one or more *field* entries or one *record*. Options available with *field* or *record* are grouped with braces and brackets:

Syntax

```
ASSIGN   { [ FRAME frame ] { field [ = expression ] }
           [ WHEN expression ] } . . .
       | { record [ EXCEPT field . . . ] }
```

OpenEdge messages

OpenEdge displays several types of messages to inform you of routine and unusual occurrences:

- **Execution messages** inform you of errors encountered while OpenEdge is running a procedure; for example, if OpenEdge cannot find a record with a specified index field value.
- **Compile messages** inform you of errors found while OpenEdge is reading and analyzing a procedure before running it; for example, if a procedure references a table name that is not defined in the database.
- **Startup messages** inform you of unusual conditions detected while OpenEdge is getting ready to execute; for example, if you entered an invalid startup parameter.

After displaying a message, OpenEdge proceeds in one of several ways:

- Continues execution, subject to the error-processing actions that you specify or that are assumed as part of the procedure. This is the most common action taken after execution messages.

- Returns to the Procedure Editor, so you can correct an error in a procedure. This is the usual action taken after compiler messages.
- Halts processing of a procedure and returns immediately to the Procedure Editor. This does not happen often.
- Terminates the current session.

OpenEdge messages end with a message number in parentheses. In this example, the message number is `200`:

```
** Unknown table name table. (200)
```

If you encounter an error that terminates OpenEdge, note the message number before restarting.

Obtaining more information about OpenEdge messages

In Windows platforms, use OpenEdge online help to obtain more information about OpenEdge messages. Many OpenEdge tools include the following Help menu options to provide information about messages:

- Choose **Help** > **Recent Messages** to display detailed descriptions of the most recent OpenEdge message and all other messages returned in the current session.
- Choose **Help** > **Messages** and then type the message number to display a description of a specific OpenEdge message.
- In the Procedure Editor, press the **HELP** key or **F1**.

On UNIX platforms, use the OpenEdge `pro` command to start a single-user mode character OpenEdge client session and view a brief description of a message by providing its number.

To use the pro command to obtain a message description by message number:

1. Start the Procedure Editor:

   ```
   OpenEdge-install-dir/bin/pro
   ```

2. Press **F3** to access the menu bar, then choose **Help** > **Messages**.
3. Type the message number and press **ENTER**. Details about that message number appear.
4. Press **F4** to close the message, press **F3** to access the Procedure Editor menu, and choose **File** > **Exit**.

1

Introducing Table Partitioning

In OpenEdge Release 11.4, the OpenEdge RDBMS introduced horizontal table partitioning. OpenEdge Release 11.5 further enhances table partitioning. Horizontal table partitioning allows Database Administrators (DBAs) to design a physical database layout that aligns storage with specific data values or ranges. The physical separation of data into partitions can improve performance, maintenance, and data availability.

For details, see the following topics:

- Table partitioning details
- Table partition types
- Defining partitions
- Table partitioning restrictions
- Indexes for table partitions
- Getting started partitioning your database

Table partitioning details

Table partitioning enables one table to be divided into multiple self-contained sub-sections (partitions) that can be located in different physical areas. A row from a partitioned table never spans a partition boundary. Prior to OpenEdge 11.4, a table, regardless of size, was physically stored in only one storage area. It was also treated as a single unit at the application and database levels. Consequently, for a very large table, it could be very slow and costly to perform certain operations, either run-time queries or maintenance activities.

Partitioning tables can provide advantages, including the following:

- Improved performance
  - If querying with a partition-aligned index, whole partitions can be quickly removed from consideration. This is called pruning.
  - Concurrency is increased in random activity.
- Improved availability

  Maintenance can be performed on a single partition, leaving other partitions from a table available.
- Improved maintenance

  Utilities can operate on one partition, reducing the amount of time that data is unavailable (locked by the utility). Partition maintenance includes partition management to split, merge and purge partitions; partition-level index management such as index check, fix and rebuild; and partition-level data activities such as binary dump and load.

Partitioning tables also has the risk associated with these disadvantages:

- Correct partitioning requires knowledge of table layout and application deign to be aligned with the physical layout of the database. Re-partitioning is expensive, requiring a dump and load of data.
- Updating a partition-aligned column can cause a row to be moved from one partition to another.

Table partition types

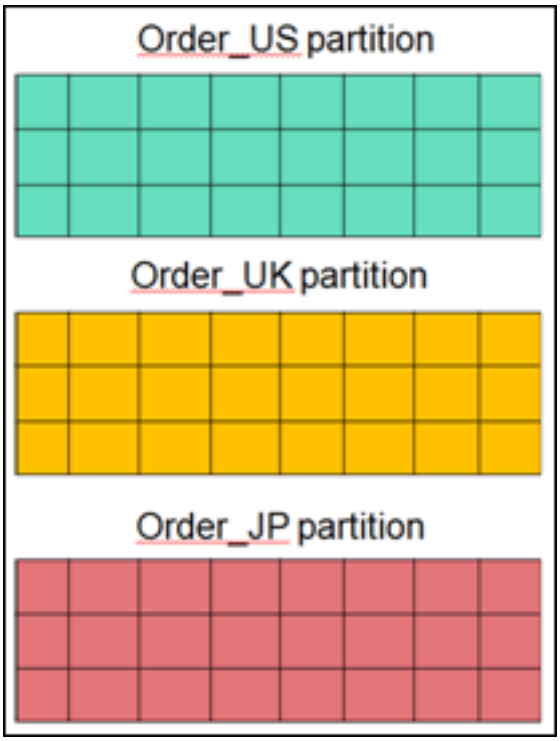
Table partitioning enables you to divide the rows of large tables into smaller units, called partitions, and manage each partition individually. The table appears as one single unit to the application, but it is physically divided into multiple partitions in the database. OpenEdge table partitioning supports several different kinds of partitions, as follows:

- List

  For a list partition, the partition definition is based on a single value, and the values for the column that define the partition define a complete set. You cannot create a row that does not match the partition definition.

For example, suppose you had a table of orders containing a country column, and you wanted to partition the column based on the three possible country values, JP, UK and US, the list-partitioned table would look like the following figure:

Figure 1: Example list partition

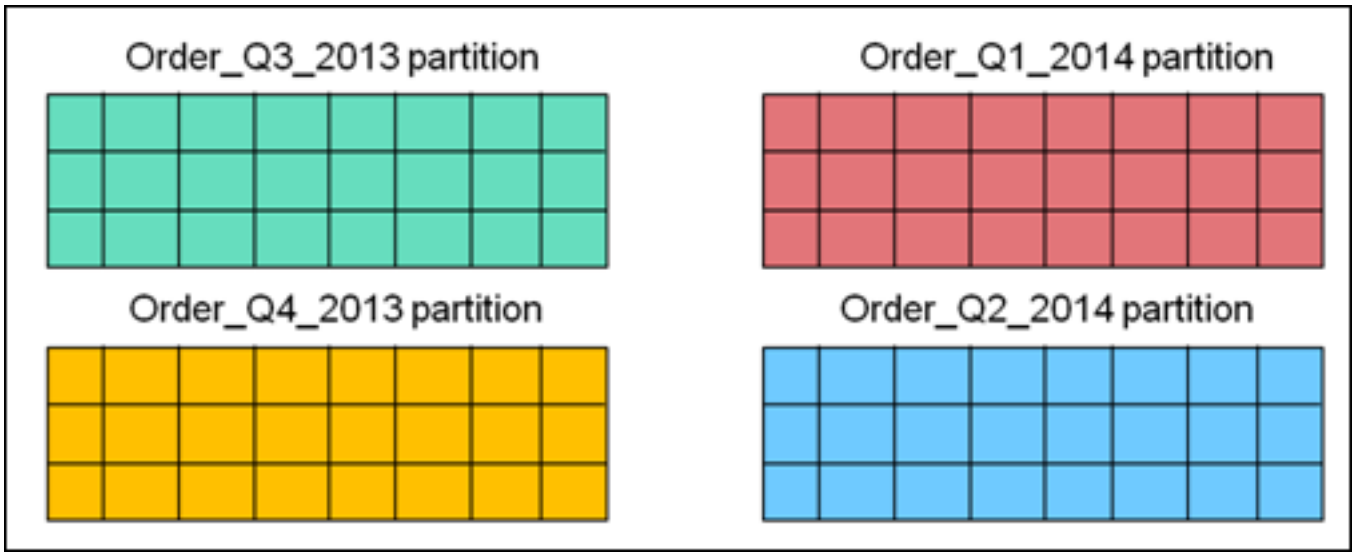


- Range

  For a range partition, values for the column that define the partition match particular ranges. Ranges cannot overlap. For example, if you are partitioning based on a date field and defining partitions for a year's worth of data, your partitions define ranges of January 1 through December 31, for each year.

  For example, suppose you had a table of orders containing an order date column, and you wanted to partition the column based on quarters of the calendar year. The range-partitioned table would look like the following figure:

Figure 2: Example range partition



- Sub-partitions

  You can partition a partition, to further refine the subset of a table.

  - **List-range** — A list-range partition first defines a partition by a list, and then further partitions the data based on a range.
  - **List-list** — A list-list partition first defines a partition by a list, and then further partitions the data based on a subsequent list.

  You can create up to 15 levels of sub-partitions. You can have 15 list sub-partitions, or you can have 14 list sub-partitions, and finally a last sub-partition a range partition.

  For example, taking the two previous examples of a list partition and a range partition on a table of orders, combining them into a list-range partitioned table would look like the following figure:

Figure 3: Example list-range partition



- Composite

  A composite partition is a partition that contains the data for multiple partition definitions in the same physical partition. Intended as a migration tool, when initially partitioning an existing table, all data resides in the *composite initial* partition.

Defining partitions

Partitions are defined in your database meta-schema in partition policy and partition policy detail records. You define one partition policy per table, and then one partition policy detail record for each partition. You can create policy (`_partition-policy`) and detail (`_partition-policy-detail`) records with ABL, SQL, or through the OpenEdge Explorer or OpenEdge Management interface.

Defining a partition policy

Defining a partition policy record requires you to define the table being partitioned and the attributes of the partition listed in the following table:

Table 1: Partition policy record

Property	Description
Policy name	A unique name for the partition policy. Do not use spaces or special characters such as the asterisk (*), the ampersand (&), and the period (.).You can use the underscore (_) character in addition to alphanumeric.
Description	A description of the partition policy.
Database connection	The database connection name of a table-partitioning-enabled database.
Table	The name of the table.
Default data area	The default storage area for the table's data.
Default index area	The default storage area for the table's indexes.
Default LOB area	he default storage area for the table's LOBs.
Allocation	Object allocation rule • None — Set new partitions not to allocate space (default). Use this option if you want to allocate space individually to new partitions when you create them later by adding partition policy details to the policy. • Immediate — Set new partitions to allocate space. Use this option if you want to allocate space globally to new partitions when you create them later by adding partition policy details to the policy.
Has range	The table partition type • No (unchecked in the OpenEdge Management wizard) — Indicates that the partition is a list partition or all list sub-partitions. • Yes (checked in the OpenEdge Management wizard) — Indicates that the partition or your last sub-partition is a range.

Property	Description
Partition fields	Identifies the partition fields. You can easily select the fields with the **Use Add field from table** or **Add fields from index** to add partition fields. If a local index uses the selected field, the index appears in the Partition aligned indexes area.
Partition aligned indexes	Identifies partition-aligned indexes to use in the partitioning.

Defining a partition policy detail

Defining a partition policy detail record requires you to define the attributes of one specific partition. The properties of the partition policy detail are listed in the following table:

Table 2: Partition policy record

Property	Description
Name/Description	A unique name of the partition policy detail; it corresponds to the partition name. Do not use spaces or special characters such as the asterisk (*), the ampersand (&), and the period (.).You can use the underscore (_) character in addition to alphanumeric.
Values	Specifies a column value. • If the partition column is based on list, the comparison operator is EQ (equal to). Provide a value that matches exactly a column value. • If the partition column is based on range, the comparison operator is LE (less than or equal to). Provide a valid upper bound.
Allocation	There are three possible values: • Allocated • Composite • Split-target. Composite and Split-target are used in migrating existing table data. For new partitioned tables, specify Allocated to indicate that space is to be allocated to the partition, if you are ready to allocate space.
Default data area	The default value is inherited from the parent partition policy definition. Use this property to assign another area for the partition’s data.

Property	Description
Default index area	The default value is inherited from the parent partition policy definition. Use this property to assign another area for the partition's indexes.
Default LOB area	The default value is inherited from the parent partition policy definition. Use this property to assign another area for the partition's LOBs.

Table partitioning restrictions

Table partitioning restrictions include the following:

- Table partitions must reside in Type II areas.
- Partitions can not span storage areas.
- A maximum of 32,768 partitions (one of which is the composite initial partition) are allowed per table.
- A maximum of 15 sub-partitions are allowed per table.
- Only one range column per table may be specified as defining a range partition; it can be either the only partition or the last sub-partition.
- Overlapping range partitions are not supported. Ranges are contiguous, without gaps. Data in each range partition is all data less than or equal to (<=) the partition definition value down to the previous partition definition or down to the beginning of time if there is no smaller value.
- Partition-aligned columns in a local index must be ascending.
- Once you partition a table, if you need to re-define partitions for that table, you must dump and load the data.
- Once you enable table partitioning for your database, you must remove all partitions to in order to disable the feature.
- Table partitioning is not supported for multi-tenant tables.

Note: Note you can have both multi-tenant and partitioned tables in the same database, but you cannot use both features on the same table.

Indexes for table partitions

Two types of indexes support locating data within a partitioned table, as follows:

- Global

  A Global index on a partitioned table spans all partitions. There is one index b-tree for all of the data, and provides uniqueness for non-partitioned columns.

- Partition-aligned local

  A partition-aligned local index has one index b-tree for each partition.

  Note: Word indexes can not be local indexes.

Getting started partitioning your database

With an understanding of table partitioning, you can begin the process of partitioning your database. While a specific process is unique to every situation, the following is a high-level process outline:

- Understanding what tables you will partition.

  You will need to evaluate the layout and content of your database to determine the best tables to partition and how they are partitioned (list, range, sub-partition). Use the considerations outlined in, Table Partitioning Planning on page 23 to plan your partitioning.

- Understanding your physical layout.

  You will need to evaluate the current storage layout of your database. As you partition your database, do you need new storage areas to hold your new partitions? Do you want to move tables and indexes from their existing storage areas? Use the existing PROSTRCT and PROUTIL tools to make these changes.

- Enabling partitioning.

  Prior to creating partitions, you will need to enable your database. You can use PROUTIL or OpenEdge Management/Explorer to enable your database.

- Defining partitions

  Once you have a plan, you have several options to define your partitions:

  - OpenEdge Management/Explorer provides a graphical interface to define partitions. See Managing Table Partitioning in OpenEdge Management or OpenEdge Explorer on page 28.
  - OpenEdge SQL provides DDL for partitioning. See SQL considerations on page 30.
  - OpenEdge ABL provides APIs for partitioning. See Table partitioning API references on page 29.

- Migrating data

  When you first create partitions on an existing table, the data remains in the composite initial partition. You can use the PROUTIL PARTITIONMANAGE SPLIT option to move data out of the initial partition and into other partitions. If you have a new partitioned table, loading data through any of the standard load options (command line or tools) the data is automatically populated into the correct partitions.

- Programming considerations

  See ABL Application considerations on page 28 for an understanding of the impact of table partitioning on ABL applications.

- Ongoing database management.

  One advantage of partitioning your tables, is that certain database maintenance can be performed at the partition level. See Maintenance operations on page 31 for the updated utilities.

2

Table Partitioning Planning

Horizontal table partitioning allows Database Administrators (DBAs) to design a physical database layout that aligns storage with specific data values or ranges. The physical separation of data into partitions can improve performance, maintenance, and data availability. Your initial migration to a partitioned table architecture can be performed without moving any data, but changing an existing partition scheme requires a dump and load. For this reason, it is wise to plan your partitioning scheme well. The best table partition design depends on many factors specific to your data and database.

For details, see the following topics:

- Initial planning
- Data Considerations
- Data access (query) considerations
- Maintenance considerations
- Partition column selection
- Next steps

Initial planning

To roll out table partitioning in your deployment, you need to identify the tables to partition, and then identify the column or columns to partition on. You then need to determine the partition type to use, list or range, and the values for the columns defining what data will be directed to the different partitions. Additionally, you need to identify the location to store the data for each of the defined partitions.

There are several things to consider when identifying a table as being a good candidate for partitioning. Partitioning a table should not be based solely on the size of the table. Additionally, your partitioning scheme will serve you better if you spend a little time to investigate partitioning deeper than simply employing the all too common “just partition on the date field” strategy. The sections that follow ask general questions about your data to help you identify tables that could be good candidates for table partitioning. The more aspects in the sections below that pertain to your existing tables, the better candidate the table is for partitioning.

Data Considerations

As you decide to partition a table, consider the following questions regarding the data:

- **Does the data have a logical grouping to it?**

  Partitioning by logical data grouping provides the best maintenance and performance experience.

- **Does the data have a historical aspect to it?**

  In other words, does the data age over time and become less important to the day to day OLTP activities of the deployment? If so, it is a good candidate for partitioning since historical data can be marked read only, more easily deleted, archived or moved to a different storage device if the table were partitioned.

  Historical data can more easily be moved to less expensive storage in a partitioned environment. The alternative but just as unique to partitioning is that historical data can age on its existing aging storage device while new data can be easily directed to new storage devices. Either way, partitioning allows you control over where each data grouping (partition) is stored.

- **Does the data have an organizational aspect to it?**

  For instance, is the data broken down by regions or some other relatively static grouping? If so, it is a good candidate for list partitioning since there are specific values that identify each logical grouping of data that can be used to define each partition.

- **Does the data have order to it; either a numeric order or a time stamp ordering that would logically imply a grouping of the data?**

  For example, do you logically think of your data as being organized by quarter or an annual basis? If so, it is a good candidate for range partitioning since there are value ranges that identify each logical grouping of data that can be used to define each partition.

- **Do you have more than one column that can be used to uniquely identify a group of data in your table?**

  The more refined the partition definition, the more you will experience the virtues of partitioning. For example, partitioning by date alone will not do much to improve block level concurrency since all new data is most likely being inserted with the same date value in the same partition for any particular day. However, sub-partitioning by region as well as by date may improve concurrency between regions.

- **Do the values of the partitioning columns remain the same over time?**

  You want to pick a column that does not change frequently since the changing the value in a partition aligned field internally requires a delete followed by an insert operation to move the record to its new partition. This also has the side effect of changing the ROWID which can be troublesome if your application uses ROWIDs to directly access data.

- **Does an index containing the partition columns in order, as its leading components, exist already or must one be added? What is the cost of adding such an index if it does not already exist? Can you recompile the application to take advantage of this new index?**

The columns chosen to partition on must exist as the leading component of an index. This is referred to as a partition aligned local index and is required when enabling partitioning on a table.

- **Are the partition aligned columns well known at record creation time?**

  Partition aligned columns do not support the UNKNOWN “?” value. If the partition aligned fields are not well known at record creation time, the columns may not be the correct columns to be using to define your partitioning.

  Partition aligned columns must be assigned in the first assign statement at record creation time, or have default values that locate the record in a defined partition. Some applications create records and populate the fields of the record across several assign statements. If one of those assign statements requires the record be physically created (as opposed to deferring the record creation) then the default values of the partition aligned column will be used to initially create the record if they are not already assigned. This has an adverse performance side effect when updating the partition aligned fields, as mentioned earlier. Worse could be that the record is prevented from being created altogether if a partition definition for locating records with the default value combinations does not exist.

Data access (query) considerations

As you decide to partition a table, consider the following questions regarding how you access the data:

- **Do you want to query data by logical grouping without using an index?**

  If so, this is a good candidate for partitioning. When the table is not partitioned, a table scan needs to traverse the entire table to retrieve the requested data since there is no index to restrict the data retrieved. However, for a partitioned table, a partition scan can be performed without an index and the data retrieved restricted to one or multiple partitions based on the query.

- **Do the queries of the application typically use the partition aligned columns to choose data?**

  If so, it is a good column to partition on since this will facilitate partition pruning which is where much of the performance advantage of partitioning originates. Not only does it restrict the data that needs to be scanned, but it can also lessen contention at the block level.

- **Do you typically search or sort on non-partition aligned fields?**

  If so, a global index is usually created to improve performance by avoiding the need to perform a sort operation on the retrieved data. However, maintaining a global index has a higher maintenance cost since its concurrency during certain maintenance operations affects the entire table, not just a partition of the table. Global indexes are also larger since they contain index data for all partitions of the partitioned table making maintenance operation take longer.

Maintenance considerations

As you decide to partition a table, consider the following question regarding the maintenance of your tables:

- **Is the cost of doing maintenance too high to perform at the table level either because of the amount of time it takes to perform the operation on the entire table or because of the down time required?**

If so, it is a good candidate for partitioning since the amount of maintenance required is restricted to the partition level, there is less to do in one maintenance “chunk”. Additionally, data that does not change does not require as much maintenance as data that is constantly in flux and maintenance operations are more efficient at the partition level and allows concurrent access to a partition while a different partition for the same table is under maintenance. All these benefits increase the availability of the data.

Partition column selection

After considering tables for partitioning, you need to select the most appropriate column or columns to define your partition. Consider the following guidelines:

- The partition column selected must be the leading component of an index. Tables can only be partitioned on indexable columns (excluding RECIDs)
- The partition selected should be frequently used as a filter criterion in queries frequently run against the table to take advantage of partition pruning.
- The partition column selected should contain values that are relatively static over time. Static values reduce row movement between partitions when partition column values are changed, eliminating the performance cost of an expensive internal operation.
- The partition column selected must have known values. You can not partition a table using a column that accepts the OpenEdge ABL UNKNOWN “?” value.

Next steps

After reviewing your data in light of the questions asked, you can now select the best tables in your database to partition. Understanding how the data is accessed and updated helps you design partitions that will yield the best performance.

3

Table partitioning capabilities

This chapter highlights the capabilities of the tools and language support of table partitioning.

For details, see the following topics:

- Table partition allocation
- Managing Table Partitioning in OpenEdge Management or OpenEdge Explorer
- ABL Application considerations
- Table partitioning API references
- SQL considerations
- Maintenance operations

Table partition allocation

When allocating space for your partitions, keep in mind the following:

- Partitions must reside in Type II storage areas.
- Table, index, and LOB partitions for the same table can be stored in separate areas.
- Partitions can be defined without allocating space.

Managing Table Partitioning in OpenEdge Management or OpenEdge Explorer

OpenEdge Management and OpenEdge Explorer Table provide an interface to enable, define, and manage your table partitions. You can perform table partitioning configuration from the Database Administration Console by:

- **Establishing connections with table-partitioned databases** — You can set up local and remote connections to managed databases with table partitioning. A list of these managed databases automatically appears in OpenEdge Management or OpenEdge Explorer.

  You can also set up unmanaged local or remote database connections. Depending on the type of connection you set up, you can edit, copy, and delete it.

- **Converting a database that is not enabled for partitioning into a database enabled for table partitioning** — You can enable a database to support table partitioning.

- **Creating a table partition policy** — You can create and view a policy that includes details, such as default storage areas and an object-allocation rule, so that you can reuse the policy to create another.

For details on using OpenEdge Management or OpenEdge Explorer to manage your partitions, see *Management and OpenEdge Explorer: Managing Table Partitioning in Databases,* or the online help for these tools.

ABL Application considerations

Generally speaking, there are no requirements to change existing ABL applications when migrating to a table partitioned database. However, consider several issues when running an existing ABL application on a partitioned table:

- The order in which records are returned may differ for a newly partitioned table.

  A partitioned table must have at least one local index, which is based on the partition fields. When partitioning a table, you can make existing indexes local if they are aligned with the partition fields, but you may need or want to add a new index to satisfy the requirement for a local index. Anytime a new index is added to a table and you recompile, the new index may be used by queries that previously used a different index, and as a result, it is possible that the records will be returned in a different order than they were before.

- Partition fields must be set before record creation.

  By the time a physical record is created in the database, all of the partition fields must be set. A field is "set" either because the code explicitly assigns it a value or because the field has an initial value. The actual record creation does not happen on the `CREATE` statement. The creation often happens in the following cases:

  - The AVM is about to release the buffer, which can happen as a result of any of the following:
    - An explicit `RELEASE` or `VALIDATE` statement
    - The reading of a new record into the buffer
    - The record going out of scope
    - The end of the transaction

- You set a LOB field.
- You set a field that an index is based on.

The timing of the record creation is important in the case that the partition fields do not have initial values. If, for example, the AVM executes a CREATE statement and then immediately sets a field that an index is based on, an error occurs. This is because setting the field causes the underlying record creation to take effect, and at that point none of the partition fields have been set. An error also occurs at the end of a DO TRANSACTION block if there is a CREATE statement inside the block but not all of the partition fields are set.

- ROWID values may change as a result of changes to a partition field.

  In the past, an update to a record would never cause the ROWID of the record to change. Now, if the value of a partition field changes such that a record now belongs in a different partition, the record's ROWID will change because the ROWID identifies the location and partition of the record. If you store a ROWID value for some reason, the stored ROWID may need to be updated in this scenario.

- ROWID values encapsulate partition information, but RECID values do not.

  You cannot use a RECID value to identify the partition of a record in a partitioned table. Using a stored RECID value to retrieve a record in a partitioned table will not work unless the record is in the initial composite partition. However, the ROWID of a record in a partitioned table identifies the location and partition of the record. Applications that need to store and reuse record locations in partitioned tables should use ROWID instead of RECID.

  When changing RECID functions or attributes to ROWID in your application, you should keep the following differences in mind:

  - A RECID value is an integer, whereas a ROWID value is a variable-length set of bytes.
  - You can display a RECID directly to the screen or store it directly to a file, but you cannot do so with a ROWID. You must first convert the ROWID value to a string (using the STRING function, for example).
  - Because RECID values are integers, you can compare them using greater than (with the GT or > operator) or less than (with the LT or < operator), but you cannot do the same with ROWID values. You can, however, check ROWID values for equality.

- To retrieve partition information, a new function, BUFFER-PARTITION-ID, has been added. Two new attributes, BUFFER-PARTITION-ID and IS-PARTITIONED, have been added to the buffer object handle.

Table partitioning API references

Table partitioning related APIs provide access to table partition entity data and operations typical of the Data Dictionary, for example creating, reading, updating, deleting, enabling and allocating. This API can be used by developers who want or need to use ABL code to manage table partition information. The API is provided as a set of ABL user-defined classes and interfaces for accessing database entities. The user-defined classes include a single API service class that handles all communication between the database and entity objects. The API provides user-defined interfaces to access all these entity objects, most of which the service class creates or retrieves for a given database using factory methods.

For information about the table partition API references, see *OpenEdge Development: Programming Interfaces*.

SQL considerations

OpenEdge SQL DDL and DML support table partitioning.

- The following OpenEdge SQL DDL statements are enhanced to support table partitioning:
  - `CREATE TABLE`
  - `ALTER TABLE`
  - `CREATE INDEX`
  - `DROP TABLE`
  - `DROP INDEX`
- OpenEdge SQL DML supports table partitioning as follows:
  - Updating records

    You can use the existing UPDATE statement to update records of a partitioned table. You must have the WRITE permission on the partitioned table to update its records.

    Updating a row in a partitioned table may result in one of the following:

    - If unpartitioned key columns are updated, then the updated new record remains in its original partition. In this case, there is no change in the behavior of the UPDATE statement.
    - If only one range partition key column is updated and the updated value is in the existing partition range, then the record remains in the existing partition.
    - If one or more list partition key columns are being updated (or the range partition key column is updated and the new value are not in the existing partition), then the updated new record is moved to a different partition. The new partition ID is determined with the newly updated record's partition key columns.
    - If the new record's partition key columns cannot determine any partition while updating the record, then the UPDATE statement returns an error stating that the updated row does not belong to any partition.
  - Inserting rows

    You can use the existing INSERT statement to insert rows into the specified partition of a partitioned table. You must have the WRITE permission on the partitioned table to add rows to it.

    Executing the INSERT statement to insert rows into a partitioned table may result in an error in the following cases:

    - If all partitioned columns of the partitioned table are not specified in the INSERT statement or if they do not have any default values when the table is created.
    - If the specified values of the partitioned columns do not determine in which partition the row should be inserted.
    - If there is no space allocated in the partition to which the inserted row belongs.

For more information on OpenEdge SQL syntax, see *OpenEdge Data Management: SQL Development* and *OpenEdge Data Management: SQL Reference*.

Maintenance operations

Database utility changes in support of table partitioning fall into three categories: unchanged, enhanced to support partition-aware output, enhanced to operate on a partition, newly added specifically for table partitioning.

Unchanged database utilities continue to operate across the database, and are not impacted by table partitioning. These utilities include:

- Backup and restore
- After-imaging and Replication
- Failover clusters
- PROSTRCT
- Transparent Data Encryption
- Auditing

Utilities that are enhanced to support partition-aware output include:

- PROUTIL Analysis tools (CHANALYS/DBANALYS/IDXANALYS/TABANALYS)
- PROUTIL VIEWB2
- PROMON

Utilities that are enhanced with the ability to operate on a partition scope include:

- BULKLOAD
- DUMP/DUMPSPECIFIED
- IDXACTIVATE
- IDXBUILD
- IDXCHECK
- IDXCOMPACT
- IDXDEACTIVATE
- IDXFIX
- IDXMOVE
- LOAD
- TABLEMOVE
- Create and toss limits (DISPTOSSCREATELIMITS, SETAREACREATELIMIT, SETAREATOSSLIMIT, SETLOBCREATELIMIT, SETLOBTOSSLIMIT, SETTABLECREATELIMIT, SETTABLETOSSLIMIT)
- DBTOOL
- PROMON

Utilities that are specific to table partitioning include:

- Enable and disablement

    - ENABLETABLEPARTITIONING
    - DISABLETABLEPARTITIONING
    - DISABLEREADONLYPARTITIONS
    - ENABLEREADONLYPARTITIONS
- Partition management
    - PARTITIONMANAGE ALTER
    - PARTITION MANAGE MERGE
    - PARTITIONMANAGE RENAME
    - PARTITION MANAGE SPLIT
    - PARTITION MANAGE TRUNCATE
    - PARTITIONMANAGE VIEW
- Single partition index rebuild
    - TPIDXBUILD

Startup parameters that are enhanced with the ability to operate on a partition scope include:

- -IndexCheck
- -TableCheck

For details on these utilities and startup parameters, see *OpenEdge Data Management: Database Administration*.

4

Documentation map

This chapter provides a documentation map that directs you to additional sources of more detailed information regarding table partitioning in OpenEdge.

For details, see the following topics:

- Additional table partitioning information

Additional table partitioning information

Use the table below to locate specific details regarding table partitioning enhancements.

Table 3: Additional information

For additional information on...	See...
Database Administration	The manual, *OpenEdge Data Management: Database Administration* and the Database Administration Utilities online help.

For additional information on...	See...
Configuring and managing table partitioning with the Database Administration console in OpenEdge Management and OpenEdge Explorer	The manuals: • *Getting Started: OpenEdge Management and OpenEdge Explorer* • *OpenEdge Explorer: Managing Table Partitioning in Databases* The OpenEdge Management and OpenEdge Explorer online help.
Enhancements to and behavior of ABL	The manuals: • *OpenEdge Development: ABL Reference* • *OpenEdge Development: Programming Interfaces*
Enhancements to and behavior of OpenEdge SQL	The manuals: • *OpenEdge Data Management: SQL Development* • *OpenEdge Data Management: SQL Reference*

5

Glossary of Terms

The following section contains a listing of terms related to table partitioning.

For details, see the following topics:

- Terms

Terms

`Composite partition`

A composite partition is a physical partition where more than one logically defined partition resides in the same physical storage allocation. Prior to the migration of data in an existing table into partitions, the partition containing the table in its pre-partitioned state is a composite partition.

`Global index`

A global index in an index that contains key entries for data across all partitions of the partitioned table. Global indexes are used when a sort order other than the partitioned aligned sort order is required. For example, an index for customer name when the table is partitioned by region.

`Horizontal data/table partitioning`

Horizontal data/table partitioning is the partitioning of a table based on values of specified columns. Each row of a partition contains all the columns of the table but only contains those rows where the column value meets a particular partition policy established by the DBA.

Index of indexes

The index of indexes is an index b-tree that is special for table partitioning. The index of indexes is built and scanned like a regular index but the entries in the leaf nodes point to the root block of a local index, making it an “index of indexes” rather than an index of record pointers. The index of indexes is used to facilitate partition pruning for the languages without the need for special knowledge of the partitions being scanned. The index of indexes is also used to provide partition information for table scans of partitioned tables.

List partitioning

List partitioning partitions data from your table based on a defined set of values. For example, partitioning a table by country or state. If you have list partition based on the value of a country field, it requires a partitioning policy that lists the partition information for data storage of each valid country field value such as USA, Canada, Mexico, etc.

Local index (and local –prefixed (or aligned) indexes)

A local index is an index that contains key entries only for one particular table partition.

Multi-column partitioning

Multi-column partitioning is equivalent to sub-partitioning.

Partition aligned indexes (or partition prefixed)

Partition aligned indexes are indexes that include as the leading component(s) the column(s) associated with and in order of the table’s partition policy (the partition key).

Partition id

A partition id is database-wide unique identifier associated with each partition policy detail record in the database which is used to locate the physical data associated with the partition policy.

Partition Key

The partition key is the ordered column or columns that uniquely identify the table partition. The partition key is described in the partition policy.

Partition name

A partition name is a (database wide) unique name given to a table partition policy. A partition name can be either system generated implicitly or specified by the user explicitly. A valid partition name follows the same rules as a valid table name.

Partition pruning

Partition pruning is the process of analyzing a request for data such that only the partitions containing data associated with the request are examined, For instance, for each sales where quarter GT EQ 3 and year EQ 2011: only looks at 3 partition out of the all the partitions associated with the particular table. Partition pruning is performed by the storage engine when no specific partition is requested by the language engines.

Physical data partition

The physical data storage partition is the “container” associated with a particular table partition policy.

`Range partitioning`

Range partitioning is the partitioning of a table based on a value range associated with a particular column. The column can be any indexable data type that you can divide with a greater than or less than (<=) evaluation. Simple examples include partition definitions based on a range of values for the given data type such as a historical (date) range or a salary (numeric) range.

`Sub-partitioning`

Sub-partitioning is partitioning of more than one column of the same table (multi-column partitioning). Some examples are range-list partitioning or list-list partitioning. It is the combination of the values of each of the participating columns which determine the table partition to use for the data.

`Table Partition Policy`

A table partition policy is a policy record stored as database meta-schema that defines how a particular table is partitioned. A partition policy includes the partition name, the partition type, the column or columns involved and various attributes associated with the partition.

`Table Partition Policy Detail`

The partition policy detail is a policy record stored as database meta-schema that defines the list of values or ranges for each column defined in the table partition policy.

`Table Partitioning`

Table partitioning is the act of splitting data associated with a table definition into separately managed pieces, based on the value or range of values for a specific column or columns.

Index

A

ABL applications
    use with partitioned tables *28*
ABL partitioning APIs *29*

D

Database Administration Console
    Table partitioning features *28*
Database administration of table partitions *31*

M

Managing table-partitioned databases *28*

P

Partition
    Allocation *27*
    Column selection *26*
    Defining *18–19*
Partition policy *19–20*

T

Table Partitioning
    Capabilities *27*
    Database administration *31*
    Documentation map *33*
    Glossary *35*
    Introduction *15*
    Using ABL APIs *29*
    Using SQL *30*