Progress®



Create a Database

OpenEdge®

Copyright

© 2021 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.

DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, DataRPM, Defrag This, Deliver More Than Expected, DevReach (and design), Icenium, Inspec, Ipswitch, iMacros, Kendo UI, Kinvey, MessageWay, MOVEit, NativeChat, NativeScript, OpenEdge, Powered by Chef, Powered by Progress, Progress, Progress Software Developers Network, SequeLink, Sitefinity (and Design), Sitefinity, Sitefinity (and design), SpeedScript, Stylus Studio, Stylized Design (Arrow/3D Box logo), Styleized Design (C Chef logo), Stylized Design of Samurai, TeamPulse, Telerik, Telerik (and design), Test Studio, WebSpeed, WhatsConfigured, WhatsConnected, WhatsUp, and WS_FTP 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, Chef Automate, Chef Compliance, Chef Desktop, Chef Habitat, Chef WorkStation, Corticon.js, Corticon Rules, Data Access, DataDirect Autonomous REST Connector, DataDirect Spy, DevCraft, Fiddler, Fiddler Everywhere, FiddlerCap, FiddlerCore, FiddlerScript, Hybrid Data Pipeline, iMail, JustAssembly, JustDecompile, JustMock, KendoReact, NativeScript Sidekick, OpenAccess, PASOE, Pro2, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, Push Jobs, SafeSpaceVR, Sitefinity Cloud, Sitefinity CMS, Sitefinity Digital Experience Cloud, Sitefinity Feather, Sitefinity Insight, Sitefinity Thunder, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, Supermarket, SupportLink, Unite UX, 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.

July 2021

Product version: Progress OpenEdge 12.4

Updated: 2021/06/28

Table of Contents

Preface....7

Physical components of a database....9

Storage areas....10
Types of storage areas....11
Using storage areas....12
Guidelines for creating storage areas....13
General guidelines for creating storage areas for application data....13
Specific guidelines for creating storage areas for application data....14
Extents....15
Types of extents....15
Guidelines for creating extents....16
Clusters....17
How clusters improve performance....17
Record fragmentation....18
Record scatter....19
Guidelines for specifying cluster sizes....19
Database blocks....20
Guideline for specifying database block size....20

Database files....21

Additional files....22
Matching database files with storage areas....23

Methods of creating a database....25

Source databases....26
Existing databases....27
Empty databases....27
Method 1-Creating a database using PRODB....28
Method 2-Creating a database using PROCOPY....30
Method 3-Creating a database using PROSTRCT CREATE and PROCOPY....32
Decisions to make before using PROSTRCT CREATE and PROCOPY....33
Task 1-Creating a .st file....34
Tokens....35
Rules for creating a .st file....36
Example of a .st file....38
Task 2-Validating a .st file....39
What PROSTRCT CREATE -VALIDATE does....39
Using PROSTRCT CREATE -VALIDATE....40

Task 3-Creating a void database using PROSTRCT CREATE.....41
Task 4-Populating the void database using PROCOPY.....42
Task 5-Loading definitions and data into the new database.....43

Deleting a database.....45

Preface

Purpose

This book tells how to create and delete OpenEdge databases, describes database components, and lists physical files that comprise the database.

Audience

This book is intended for database administrators who want to manage OpenEdge databases or application developers who need a basic introduction to . It does not assume OpenEdge experience.

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

Organization

Physical components of a database on page 9

Describes storage areas, extents, clusters, and guidelines for using them.

Database files on page 21

Describes the physical files on disk that make up a database, and their associated storage areas.

Methods of creating a database on page 25

Describes the four ways to create a database, when to use them, and types of source databases.

Deleting a database on page 45

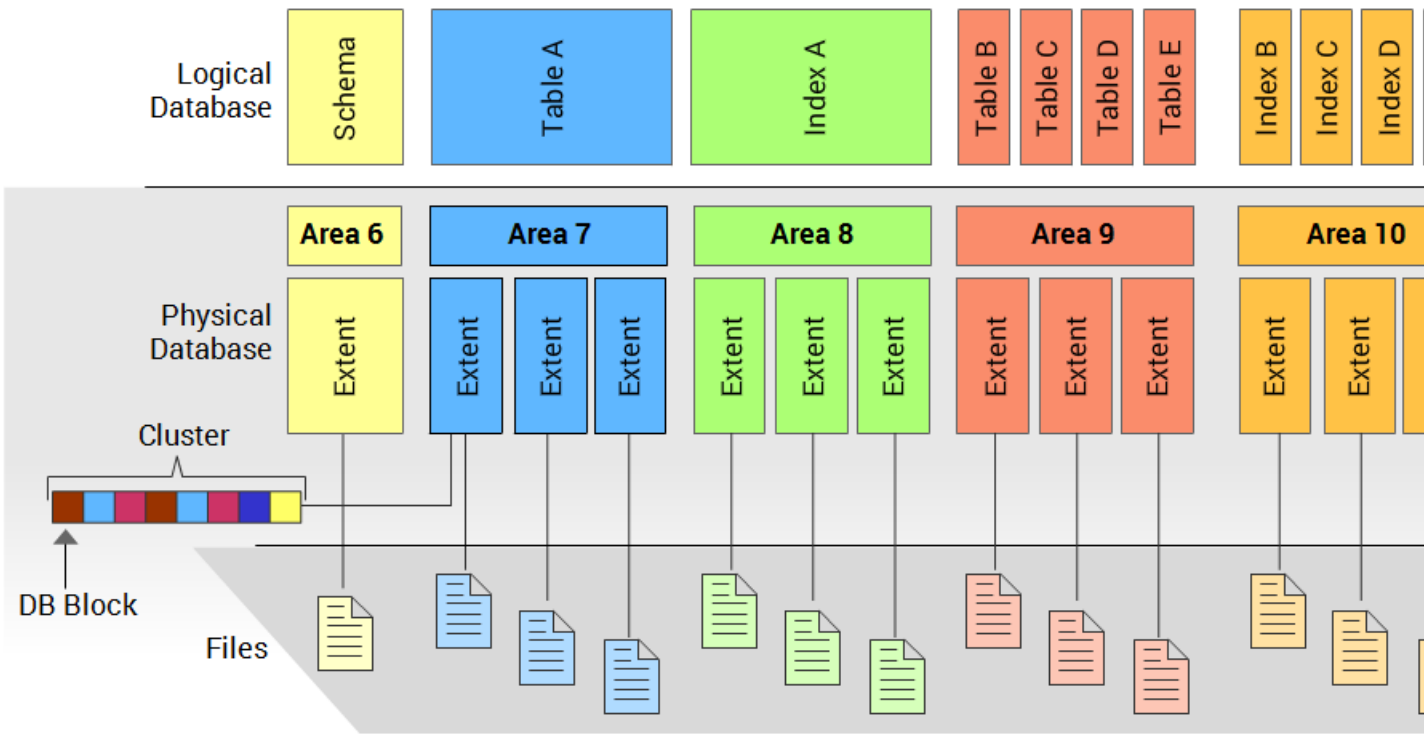
Describes how to delete a database and its structure description file.

Documentation conventions

See Documentation Conventions for an explanation of the terminology, format, and typographical conventions used throughout the OpenEdge content library.

1

Physical components of a database



A disk-resident database can be viewed as both a logical and a physical database.

The logical database consists of database objects such as tables, indexes, sequences, and SQL views. Typically, a logical database is the domain of application developers who are responsible for creating and maintaining logical database objects.

The physical database consists of files that contain permanent data on disk. A physical database is the domain of database administrators who are responsible for creating and maintaining database files. A physical database consists of the following components:

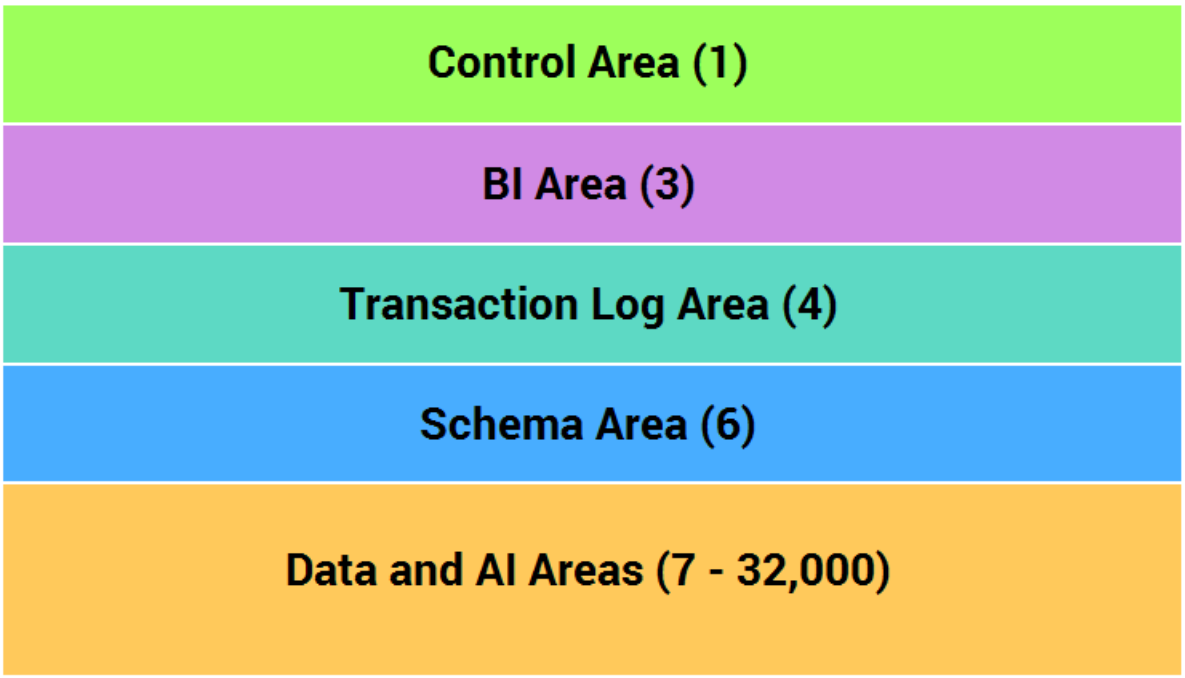
- Storage areas
- Extents
- Clusters
- Database blocks

You will learn about each of these components next.

For details, see the following topics:

- Storage areas
- Extents
- Clusters
- Database blocks

Storage areas



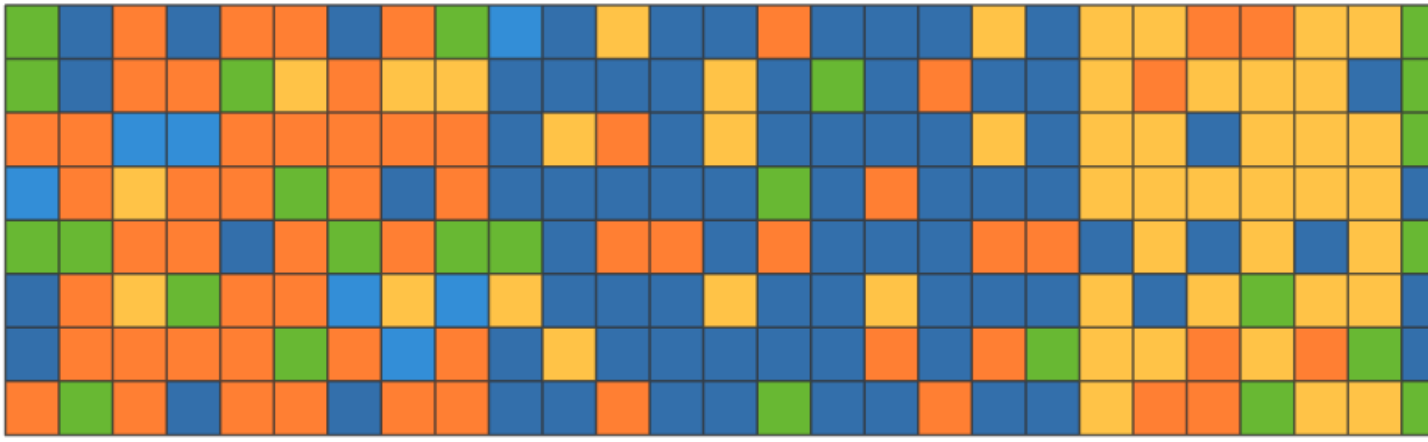
A storage area is a logical set of physical files on disk. It represents the largest storage unit of a database. Each storage area consists of one or more extents. There can be up to 1,024 extents in a storage area.

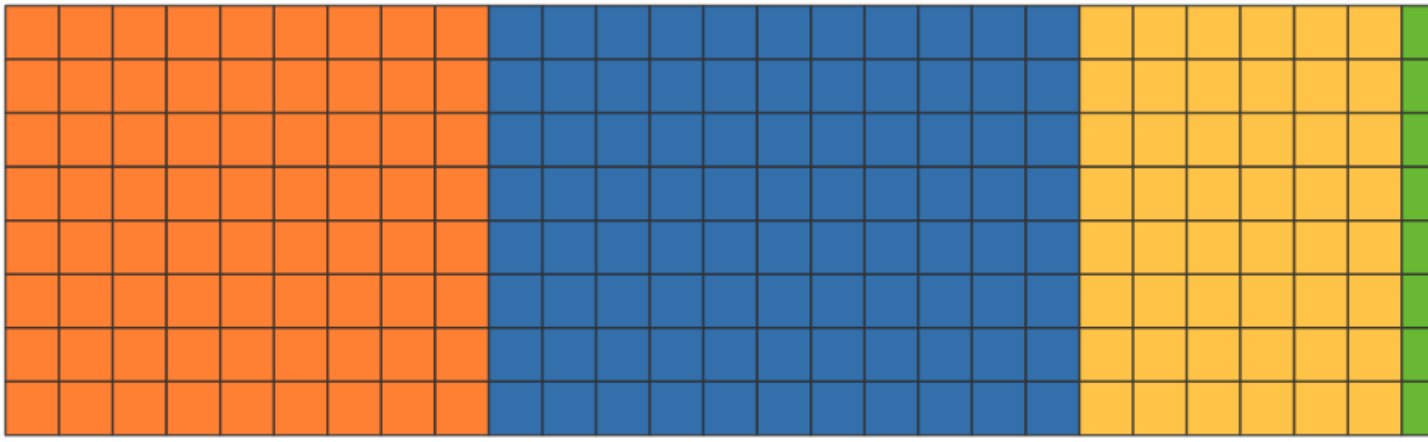
Storage areas enable you to:

- Organize a database into distinct areas, which provides better database performance and greater ease of administration.

- Assign logical groupings of application data to different areas, which allows you to manage different portions of data independently.
- Place specific objects into dedicated areas to keep them in contiguous database blocks or clusters for optimal performance.
- Place different areas on different disks, which results in less disk contention and provides better performance.
- Perform a dump and load by area, which is useful if you cannot dump or load an entire database within a limited time window.
- Keep static (rarely changed) data separate from dynamic (frequently updated) data, thus improving performance.
- Assign each data area the optimal number of records per block and blocks per cluster to fit the data for which it is designed to store.

Types of storage areas





There are two types of data storage areas:

- Type I—Storage space in Type I areas is allocated in units of database blocks. Database blocks in Type I areas store data of different objects. Type I areas cannot have clusters.
- Type II—Storage space in Type II areas is allocated in units of clusters. Clusters are made up of contiguous database blocks. Database blocks in Type II areas store data of the same object.

Type II storage areas provides the following benefits:

- They lessen the need to dump and load data. With Type I areas, a dump and load needs to be performed every 12 to 18 months. With Type II areas, a dump and load needs to be performed every 3 to 5 years.
- They enable data to be stored efficiently in clusters. This results in less block fragmentation and scatter and provides better performance.
- They make it possible to have very large databases.
- They allow you to configure the create and toss limits for each area, enabling data to be stored efficiently.
- They enable object-level maintenance operations (for example, table moves, index fixes) to be done quickly.
- They provide cyclic redundancy checks (CRCs), which is included in every Type II database block and enables early detection of database corruption.

Type II storage areas also enable you to take advantage of database features such as auditing, transparent data encryption (TDE), multi-tenancy, and table partitioning.

So, although you can have both Type I and Type II storage areas in a database, you should use always Type II areas wherever possible.

Using storage areas

OpenEdge RDBMS supports a maximum of 32,000 numbered storage areas, of which the first six are reserved. Currently, OpenEdge RDBMS uses only four of these areas (1, 3, 4, and 6). The areas 2 and 5 are reserved for future use. They cannot be used for application data.

The remaining 31,994 areas (7 to 32,000) are set aside for application data and after-image (AI) transaction notes.

The following table lists the different storage areas with their descriptions:

Area number	Area name	Description
1	Control area	Contains a description of the database structure and acts as the table of contents of a database.
3	Before-Image (BI) area	Contains BI transaction notes used to recover a database from a database crash.
4	Transaction Log (TL) area	Contains transaction log data generated when two-phase commit is in use.

Area number	Area name	Description
6	Schema area	Contains schema tables, indexes, sequences, SQL views and triggers, and Java stored procedures. Note: The schema area is of Type I; it cannot be converted to Type II.
7 to 32,000	• Application data areas • After-Image (AI) areas	These areas contain: • Application tables and indexes • AI transaction notes

Guidelines for creating storage areas

Creating storage areas properly enhances the performance of a database. Follow these guidelines when you create storage areas:

- Create distinct BI, schema, application data, AI, and TL storage areas.
- Use the schema area for storing database schema only—do not store application data in the schema area.
- OpenEdge RDBMS names the BI, AI, and TL storage areas; you do not have to name them.

General guidelines for creating storage areas for application data

When you create storage areas for application data, follow these general guidelines:

Guideline	Reasons
Keep table data separate from index data.	• Rebuilding individual indexes will be faster. • There will be less scatter factor compared to an area that has both tables and indexes stored in the same area. • When reorganizing a database, parallel loads are more practical and efficient. • If an index area becomes corrupted, you can fix the indexes in just that area.
Keep the total number of storage areas to 20 or fewer.	• Fewer file handles are required by the operating system (OS). • There are fewer areas to monitor. • The structure description (.st) file is shorter and simpler. • There is less excess empty space in the database.

In addition, you should also follow these general guidelines:

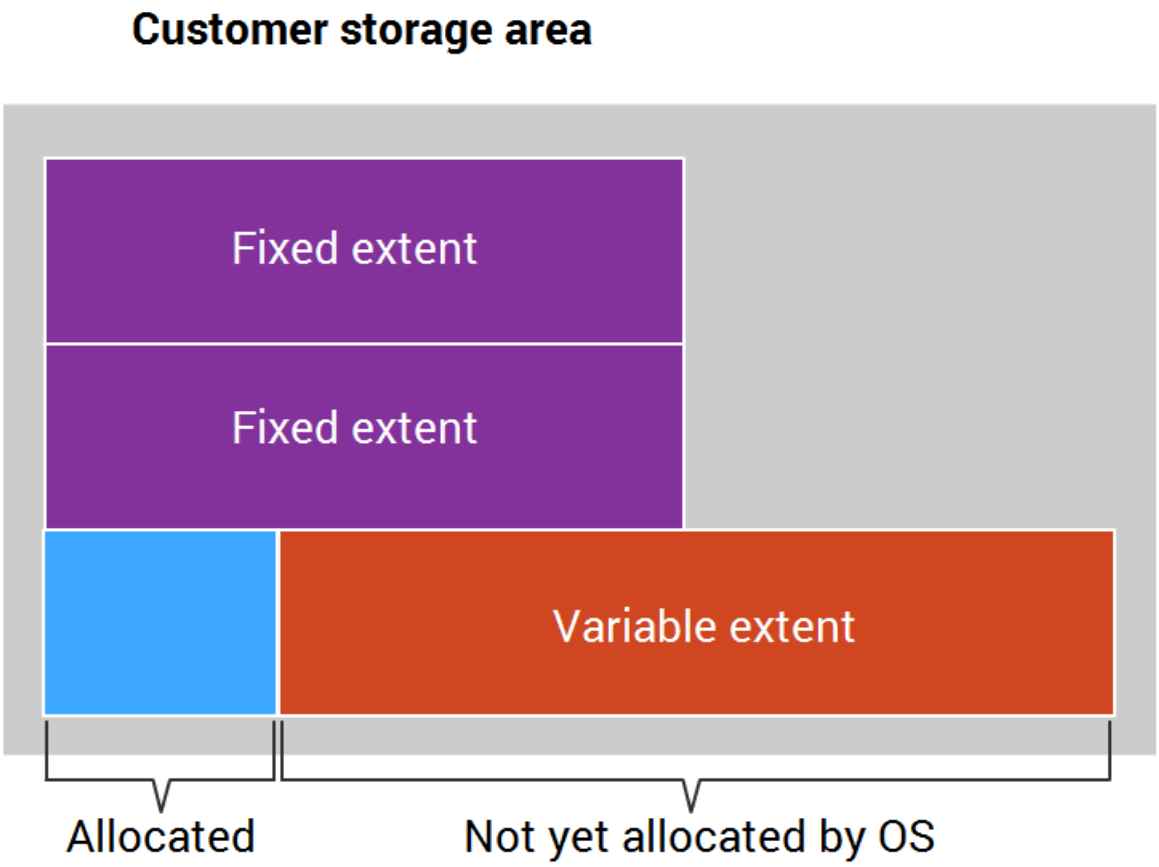
- Keep each storage area to 10GB or less.
- Keep each storage area to 10 extents or fewer.

Specific guidelines for creating storage areas for application data

When you create storage areas for application data, follow these specific guidelines:

If the database contains …	Then …
Many large or frequently updated tables	Create one area for each table.
Many static tables	Create a separate area for all of the static tables.
Many small tables	Create one area for each category of tables and assign logically related tables to each area. For example, create one area for Customer tables and one area for Sales tables.
Many indexes for large or frequently updated tables	Create one area for each index.
Many small indexes	Create one area for each category of indexes.
Temporary (scratch) tables for storing records that are added and then deleted after the processing is complete	Create a separate area for all of the temporary tables.
Tables that consistently have high fragmentation or scatter factors	Create a separate area for each table.
Tables in which all records are deleted in one operation (for example, history tables that get purged once a year)	Create a separate area for all of the tables.

Extents



An extent is a physical file of a database on disk. In Type II data storage areas, each extent contains multiple clusters.

As a physical file, an extent can grow up to 2GB. If you have an OpenEdge Enterprise RDBMS license, you can enable large files for a database. In such cases, an extent can grow up to the maximum file size permitted by your OS or the maximum extent size of 1TB.

Here are the benefits of extents:

- You can control the size, type, and location of extents using the .st file.
- You can spread the I/O over several disks by strategic placement of extents.
- You can optimize hardware usage by placing the extents of a storage area on a dedicated, fast disk for heavily used tables and indexes.
- You can reduce disk contention and enhance database performance by placing extents of the same storage area on separate disks or file systems.

Types of extents

There are two types of extents—fixed and variable.

The following table lists each type of extent with its description:

Type of extent	Description
Fixed extent	Has a fixed size. Enables you to control how much disk space to allocate to each extent and how many extents to use for each storage area. Is pre-formatted and pre-allocated up to a specified size. This provides better database performance and guarantees that disk space is reserved for your database. OpenEdge RDBMS requires that all extents of a storage area, except the last, be fixed.
Variable extent	Captures the overflowing data of a storage area. Grows until it reaches the maximum file size permitted by your OS or until it uses up an entire disk or file system. As a best practice, you should make the last extent of a storage area a variable extent.

Guidelines for creating extents

When you create extents, follow these guidelines:

- Create only one variable extent for the schema area because a database schema typically does not grow frequently or rapidly.
- When you have multiple extents in a storage area, make the last extent variable.

Making the last extent a variable extent provides a safety valve in case of a sudden burst of activity.

If the last extent is a fixed extent and it fills up, attempts to add data will cause the database to shut down.

When you specify the size of a fixed extent, ensure that it is a multiple of 16 times your database block size. For example, if your database has the 8K block size, you should specify a multiple of 128K as the extent size. If you do not, OpenEdge RDBMS automatically adjusts the extent's size to be a multiple of 16 times your database block size.

- When you specify the size of a variable extent, leave the size blank.

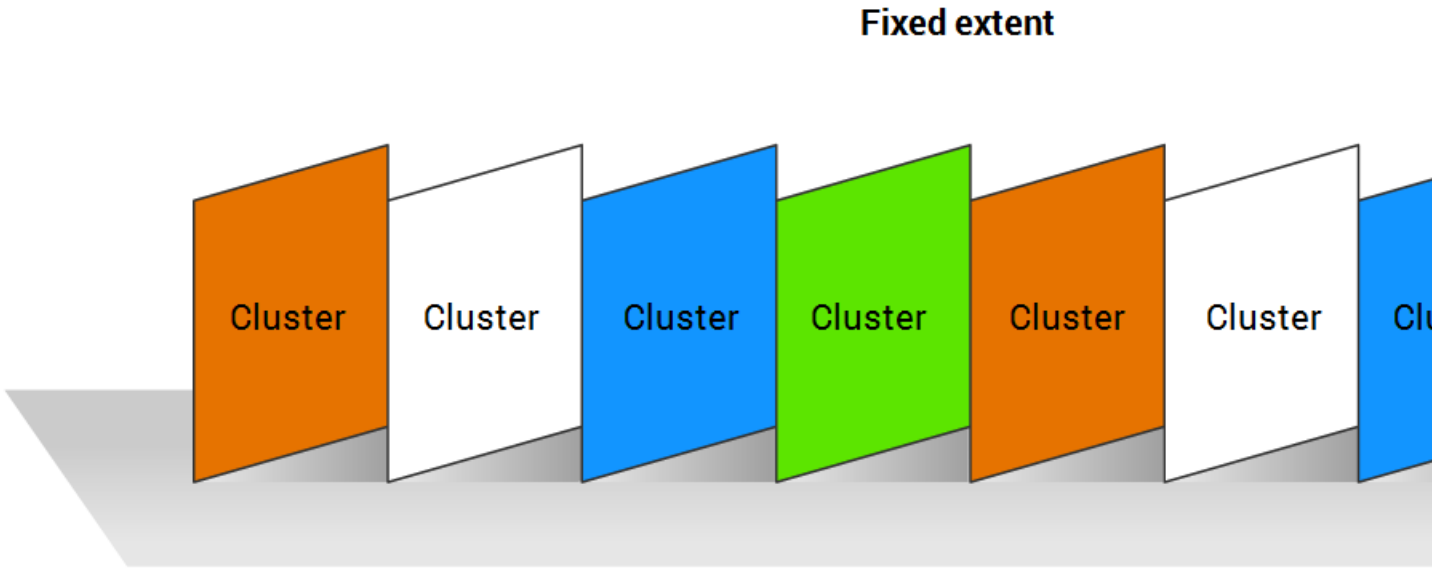
To enhance performance, specify that the BI extents be placed on a disk that is different from the one that contains application data extents. This is to ensure little disk contention when OpenEdge RDBMS writes BI and database buffers to the BI and application data files on disk.

To provide multiple points for recovery in the event of media failure, specify that the AI extents be placed on a disk that is different from the disks that contain the BI and application data extents. If one of the disks fails, you can still use the remaining disks together with your most recent database backup to recover most, or all, of your committed transactions, up to the point of the media failure.

In general, it is better to have a few large extents than many small extents in each storage area. Then OpenEdge RDBMS has fewer files to start up and handle, which makes database administration easier and performance better.

- If your database requires large extents (that is, extents up to 1TB), then you should enable large files. Otherwise, it is recommended that you keep the size of each extent to 1GB or less.

Clusters



To enhance performance, OpenEdge RDBMS uses a database structure called a cluster. Clusters are available for only Type II data storage areas. A cluster is a contiguous range of database blocks that store data of the same database object—table, index, Binary Large OBject (BLOB), or Character Large OBject (CLOB). Clusters can contain 8, 64, or 512 database blocks and are of the same size within the same Type II data storage area.

With clusters, OpenEdge RDBMS can:

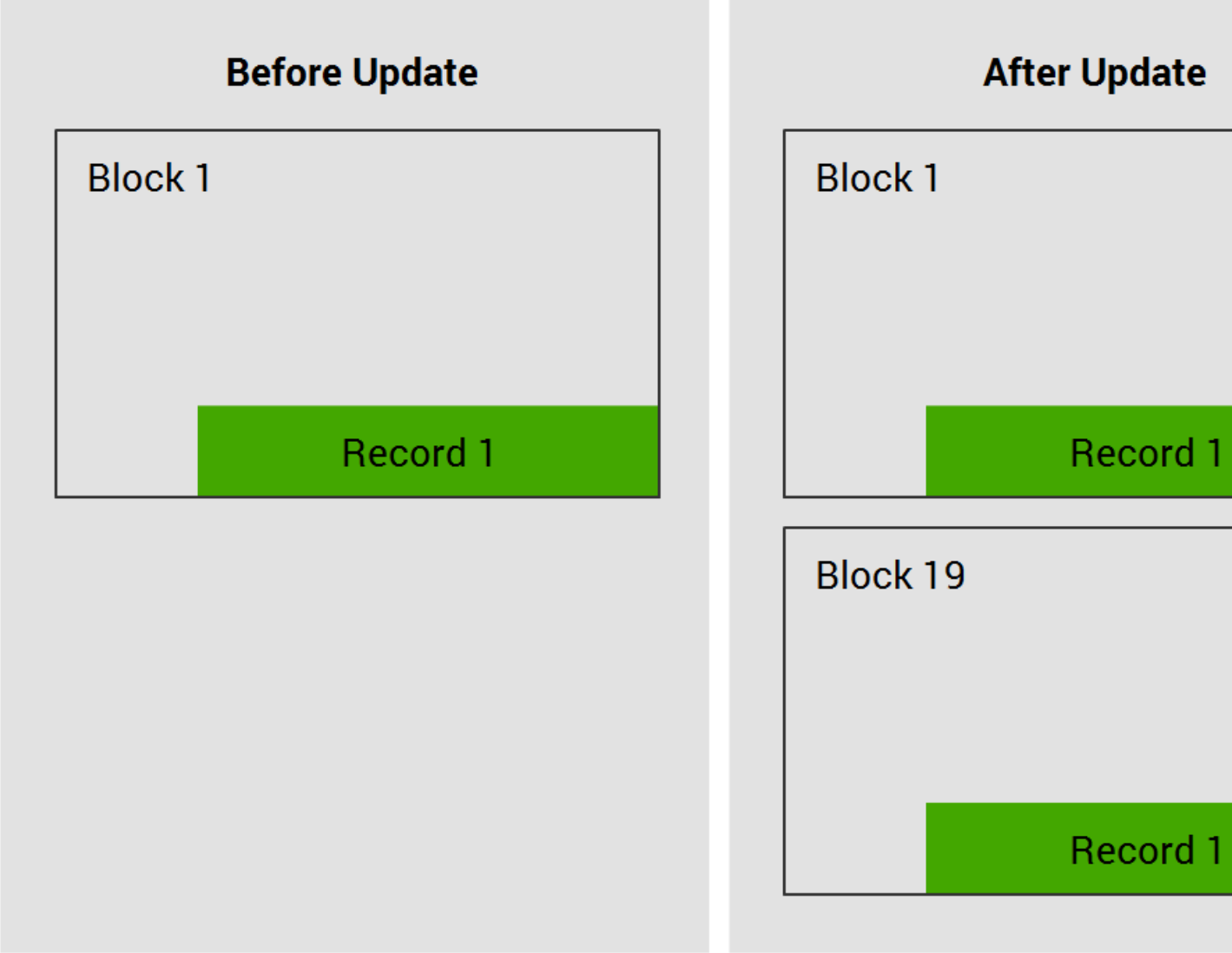
- Reduce record fragmentation and scatter.
- Scan tables quickly without using indexes.
- Execute fast deletion of tables and indexes.
- Extend variable extents quicker using units of clusters.
- Allocate concurrent space to different objects efficiently.

Clusters are especially suitable for storing large objects (BLOBs and CLOBs), which can span across hundreds of database blocks.

How clusters improve performance

Clusters improve performance of OpenEdge RDBMS by reducing record fragmentation and record scatter.

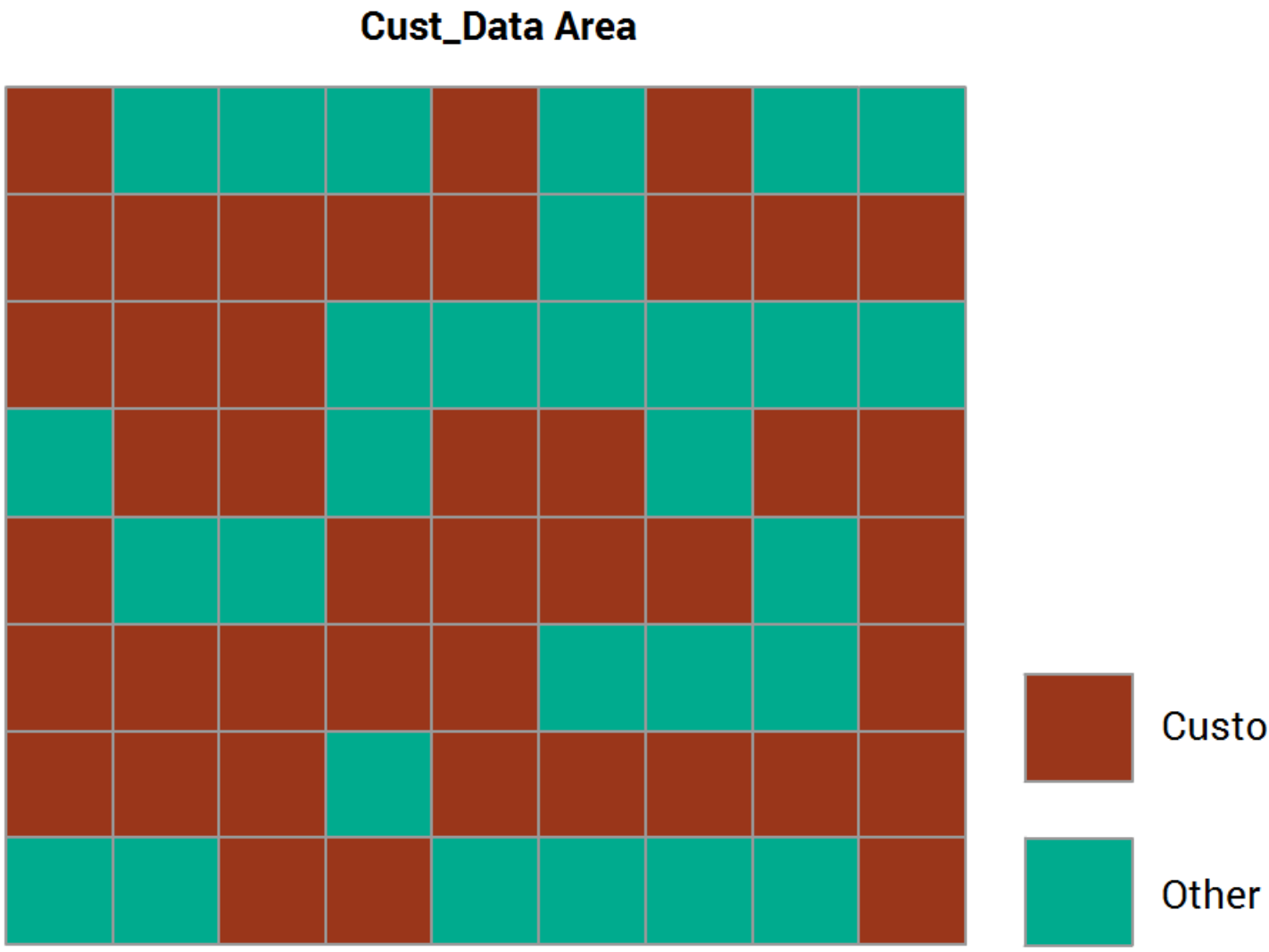
Record fragmentation



Record fragmentation occurs when the same record is split across multiple record blocks. This degrades database performance because OpenEdge RDBMS must traverse multiple record blocks to fetch one record. Since the largest record size is 32K and the largest database block size is 8K, fragmentation of large records cannot be avoided. However, a point to consider is the fragmentation of even small records when they are updated.

When you use clusters, data of the same table resides on contiguous database blocks or clusters. In addition, with clusters you can reduce record fragmentation by adjusting the space set aside for record expansion or creation in each database block. As a result, when a record is updated, there is enough space in the block for the record to expand without having to split across multiple record blocks.

Record scatter



Record scatter occurs when consecutive records of the same table are stored in multiple record blocks spread across a storage area. This degrades database performance because OpenEdge RDBMS must traverse the entire area to fetch consecutive records of the same table.

Since clusters enable records of the same table to reside next to each other, OpenEdge RDBMS must traverse only a few clusters to fetch records of the same table. This reduces record scatter.

Guidelines for specifying cluster sizes

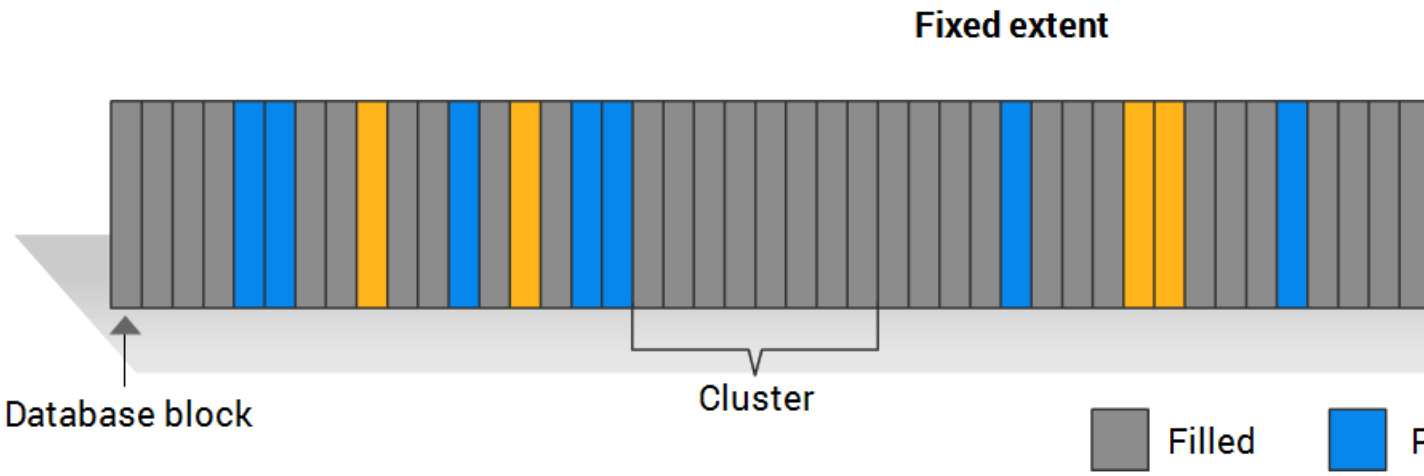
When you create a database, you can specify the number of database blocks per cluster for each storage area. In general, you should consider the nature and usage of your application data to decide which cluster size to use for a specific area.

When you specify cluster sizes for storage areas, follow these guidelines:

- Use 64 for index storage areas.

- Use 8 for data storage areas that contain small tables to minimize wasted space.
- Use 64 for data storage areas that contain medium to large tables.
- Use 512 for data storage areas that are likely to see high growth.

Database blocks



A database block is the smallest storage unit of a database. At its most basic, a database is a collection of database blocks. All database blocks in the same cluster are contiguous, and store data of the same object, be it a table or an index.

Database blocks can be set in four sizes: 1K, 2K, 4K, and 8K. If you have an OpenEdge Enterprise RDBMS license, you can specify the database block size when you create a database. When you specify the right block size, it enhances database performance because OpenEdge RDBMS:

- Stores records in your file system efficiently.
- Uses fewer disk I/Os to fetch records from disk and write data to disk.

Guideline for specifying database block size

When you specify the database block size for a database, for best performance always use the 8K block size.

2

Database files

Before you learn how to create a database, let's examine the files associated with an OpenEdge database. From the perspective of OpenEdge RDBMS, a database is organized into storage areas, extents, clusters, and database blocks. However, from the perspective of the OS, a database is a collection of physical files on disk. Every database consists of different types of files that belong to different storage areas.

The following table lists each type of file with its description:

Type of file	Description
Database structure (.db) file	A binary file that holds the table of contents of a database.
BI (.bn) files	Store BI transaction notes.The n in the .bn extension denotes different BI files such as .b1, .b2, .b3.
Transaction log (.tn) files	Store transaction numbers of all committed distributed transactions for two-phase commit.The n in the .tn extension denotes different TL files such as .t1, .t2, .t3.
Schema (.dn) files	Store schema tables and indexes.The n in the .dn extension denotes different schema files such as .d1, .d2, .d3.

Type of file	Description
Data (.dn) files	Store application data.The n in the .dn extension denotes different data files such as .d1, .d2, .d3.
AI (.an) files	Store AI transactions notes.The n in the .an extension denotes different AI files such as .a1, .a2, .a3.

For details, see the following topics:

- Additional files
- Matching database files with storage areas

Additional files

A database contains additional files that exist on disk but do not belong to any storage area.

The following table lists each additional type of file with its description:

Type of file	Description
Structure description (.st) file	A text file that describes the physical structure of a database. It includes such information as types of storage areas, records per block, blocks per cluster, types of extents, and extent sizes.
Database log (.lg) file	A text file that contains the history of important database events, such as database startups and shutdowns, login and logout information, operation of utilities, and system error messages.
Lock (.lk) file	A file whose presence indicates that the database is in use and prevents incompatible modes of operation (for example, concurrent single-user and multi-user modes).
License usage (.lic) file	A file that shows the date and time and the number of users who access the database.
Key store (.ks) file	A file that contains encryption key information for Transparent Data Encryption (TDE).

Matching database files with storage areas

Now that you know about different database files, let's examine the working directory of a sports database.

The following table shows how the database files of the sports database correspond to its storage areas. It also shows the other files that are related to the sports database but do not belong to any storage area:

File	Type of file	Storage area
When the database is not in use, you have these files:		
sports.db	Database structure (.db) file	Control area (Area 1)
sports.b1, sports.b2, sports.b3	BI (.bn) files	BI area (Area 3)
sports.d1	Schema (.dn) file	Schema area (Area 6)
sports_7.d1, sports_7.d2 sports_8.d1, sports_8.d2 sports_9.d1, sports_9.d2 sports_10.d1, sports_10.d2 sports_11.d1, sports_11.d2 sports_12.d1, sports_12.d2	Data (.dn) files	Application data areas: 7, 8, 9, 10, 11, and 12
sports.a1, sports.a2, sports.a3	AI (.an) files	After-Image (AI) areas
sports.st	Structure description (.st) file	Not applicable
sports.lg	Database log (.lg) file	Not applicable
sports.lic	License (.lic) file	Not applicable
When the database is in use, you have this additional file:		
sports.lk	Lock (.lk) file	Not applicable

3

Methods of creating a database

An OpenEdge database typically starts out as a copy of some other database. The source database can be an existing database or an empty database. There are four ways to create an OpenEdge database:

- PRODB—This is the easiest and fastest way of creating a database. It is usually used by application developers to create test or development databases to test their applications.
- PROCOPY—This is the most commonly used method of creating a database. It is the easiest way of creating a new database based on a source database. For example, customers of an application partner can use PROCOPY to create their database based on a source database provided by the partner.
- PROSTRCT CREATE and PROCOPY—This is the method you should use if you want to completely reorganize an existing database. It gives you complete control over the creation of a database to meet specific technical requirements. But it is the most time-consuming method.
- PROREST—This is the method you should use if you need to restore a database from a backup copy.

In this section, you will learn about the first three ways. All methods require a source database. Next, you will learn about different source database options.

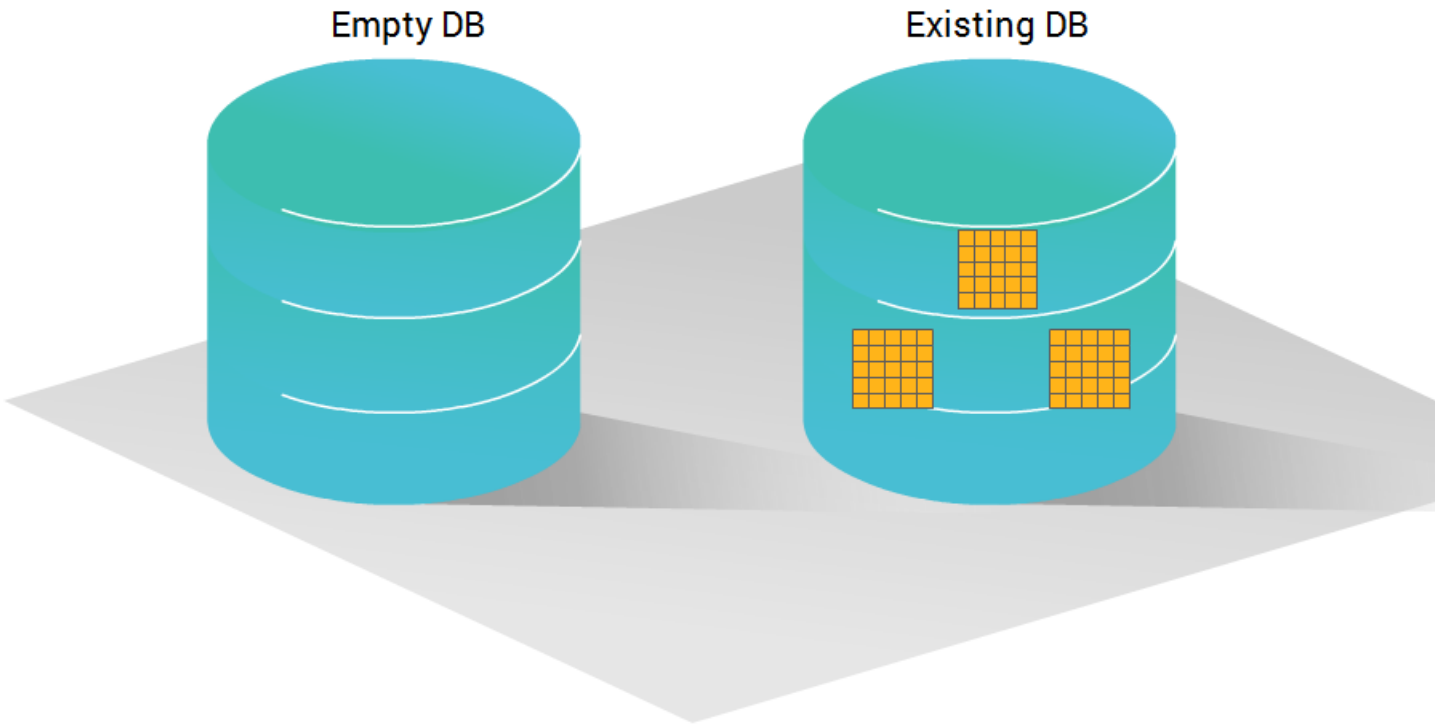
Note:

- Do not create a database in the OpenEdge install (DLC) directory (by default C:\Progress\OpenEdge in Windows or /usr/dlc in UNIX).
- OpenEdge RDBMS provides features to create, authenticate, and manage users and control their access to data in the underlying database. In applications with an embedded OpenEdge database, user authentication is often handled in the application logic. In this section, you will create and access a database without creating or authenticating users.

For details, see the following topics:

- Source databases
- Method 1-Creating a database using PRODB
- Method 2-Creating a database using PROCOPY
- Method 3-Creating a database using PROSTRCT CREATE and PROCOPY

Source databases



Source databases can be of two kinds:

- Existing databases
- Empty databases

Existing databases

An existing database contains meta-schema (default system tables), schema (database objects), and application data. You should create a new database using an existing database if you want the new database to contain the meta-schema, schema, and data of the existing database.

When you create a database using an existing database, OpenEdge RDBMS:

1. Creates a database.
2. Creates database files.
3. Populates the database files with meta-schema and schema information from the existing database.
4. Populates the database files with application data from the existing database.

When OpenEdge RDBMS is installed on a system, four sample databases are also installed. These databases are placed into the OpenEdge install directory (by default, C:\Progress\OpenEdge in Windows or /usr/dlc in UNIX). You can use these databases to quickly create a test or development database.

The following table lists each sample database with its description:

Sample database	Description
sports	The sporting goods application database.
sports2000	Similar to the sports database but contains more storage areas and application data.
isports	The international version of the sports database.
demo	The smallest of the sample databases.

Note: All the sample databases have a 4K block size.

Empty databases

An empty database contains only meta-schema information but no schema or application data. You should create a new database using an empty database if you want the new database to contain only meta-schema information, and if you want it to be based on a database block size of your choice.

When you create a database using an empty database, OpenEdge RDBMS:

1. Creates a database.
2. Creates new database files.
3. Populates the database files with meta-schema information from the empty database.

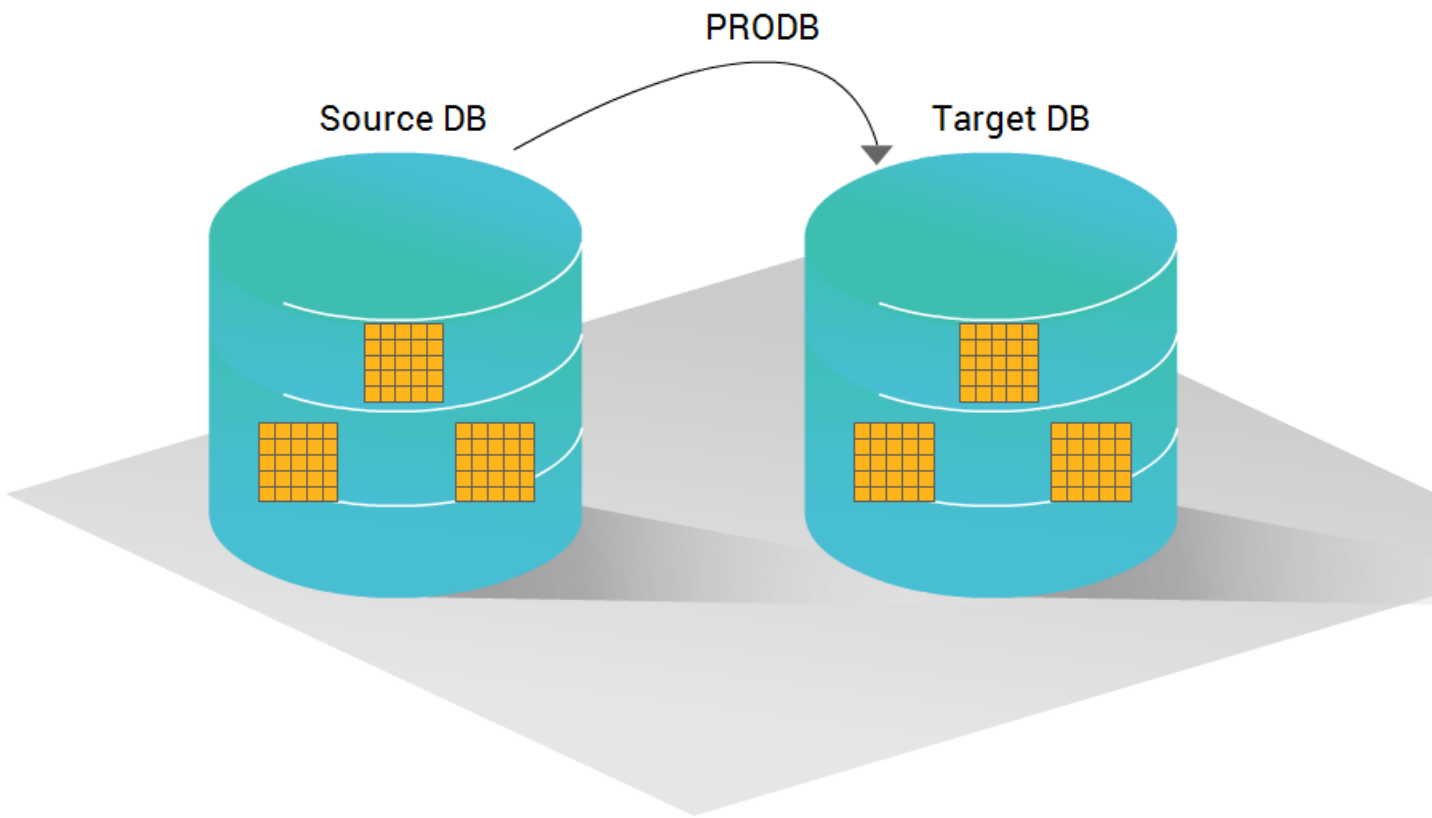
When OpenEdge RDBMS is installed on a system, five empty databases of different database block sizes are also installed in the OpenEdge install directory.

The following table lists each empty database with its description:

Empty database	Description
empty	Equal to your OS page size
empty1	1K
empty2	2K
empty4	4K
empty8	8K

Note: For best database performance, always use the 8K block size.

Method 1-Creating a database using PRODB



PRODB is usually used by application developers to create test or development databases to test their applications.

In this method, you specify a source database and a target database. PRODB copies the whole source database to the target database using the structure of the source database, and places all the extents in the current working directory.

Note: Do not use OS file commands to copy an OpenEdge database. Instead, use PRODB or PROCOPY.

Before you use the PRODB utility, ensure that the source database is not in use. The command to create a database using PRODB is:

```
prodb target-db [DLC/]source-db
```

Where:

`target-db`	The name of the database that you want to create. You can use any combination of letters and numbers, starting with a letter. Do not use OpenEdge ABL keywords or special characters in a database name.
`[DLC]`	The OpenEdge install directory. • This parameter is optional if the source database is not a default existing or empty database. • It is required if the source database is a default existing or empty database. In such a case, use %DLC% in Windows and $DLC in UNIX.
`source-db`	The name of the database that you want to use to create a new database.

Suppose you want to use PRODB to create a new database, mydb, based on the empty database, empty8, which uses the 8K database block size. In Windows, at the Proenv command line prompt enter:

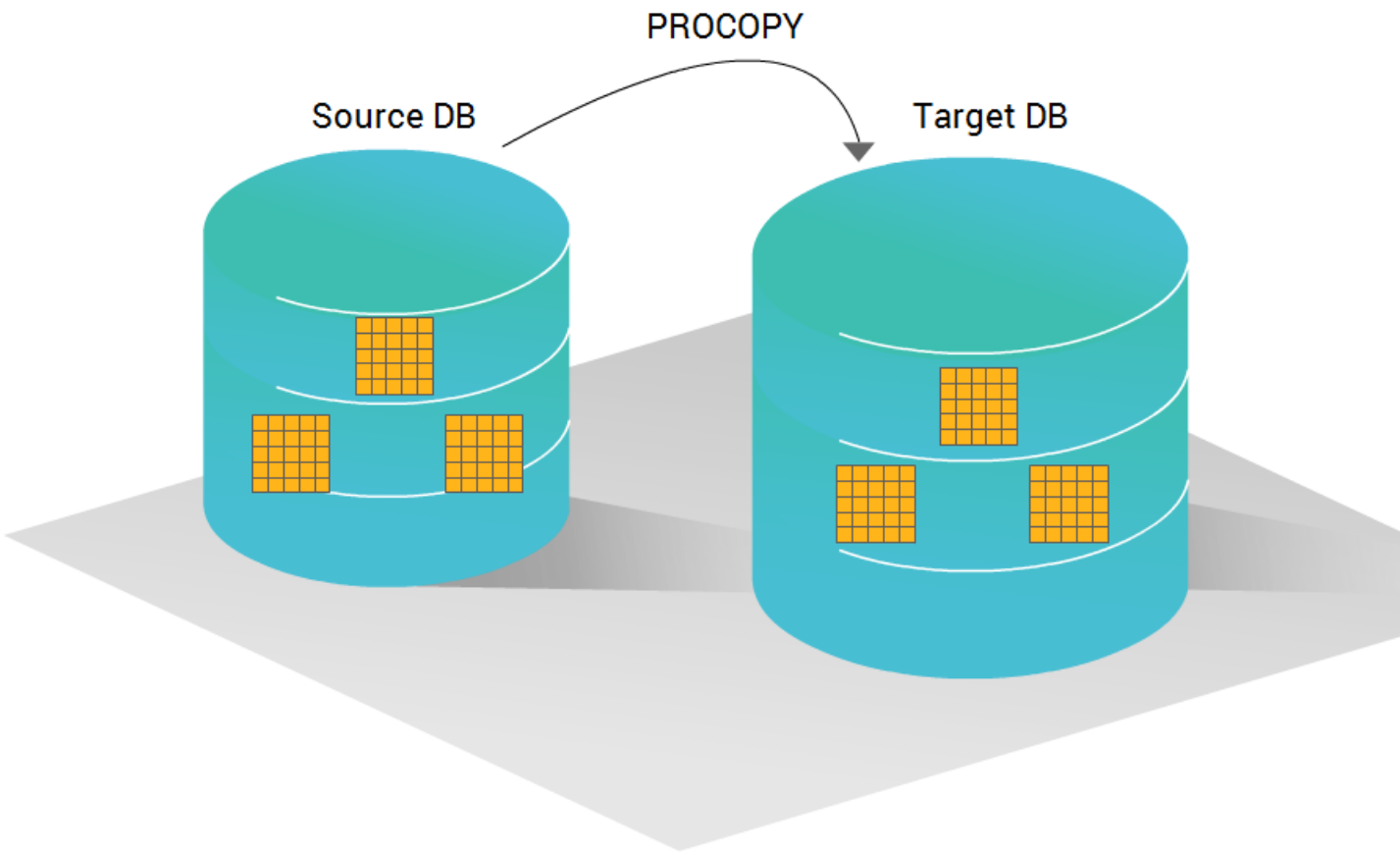
```
prodb mydb %DLC%/empty8
```

In UNIX, at the Proenv command line prompt enter:

```
prodb mydb $DLC/empty8
```

The **mydb** database is created in your working directory.

Method 2-Creating a database using PROCOPY



PROCOPY is the most commonly used method of creating a database. It is the easiest way of creating a new production database based on a source database. PROCOPY copies the meta-schema, schema, and application data from a source database to a target database.

If you want to add more extents or storage areas in your target database, then you can make a copy of the .st file of the source database and modify it. PROCOPY will use your new .st file when it creates the target database. Otherwise, PROCOPY will use the .st file of the source database.

Before you use the PROCOPY utility, ensure that the source database is not in use. The command to create a database using PROCOPY is:

```
procopy [DLC/]source-db target-db
```

Where:

[DLC]	The OpenEdge install directory. • This parameter is optional if the source database is not a default existing or empty database. • It is required if the source database is a default existing or empty database. In such a case, use %DLC% in Windows and $DLC in UNIX.
source-db	The name of the database that you want to use to create a new database.
target-db	The name of the database that you want to create.

Note: The source and target database parameters are in the reverse order of PRODB.

Suppose you want to use PROCOPY to create a database, mysports, based on the default existing database, sports. In Windows, enter:

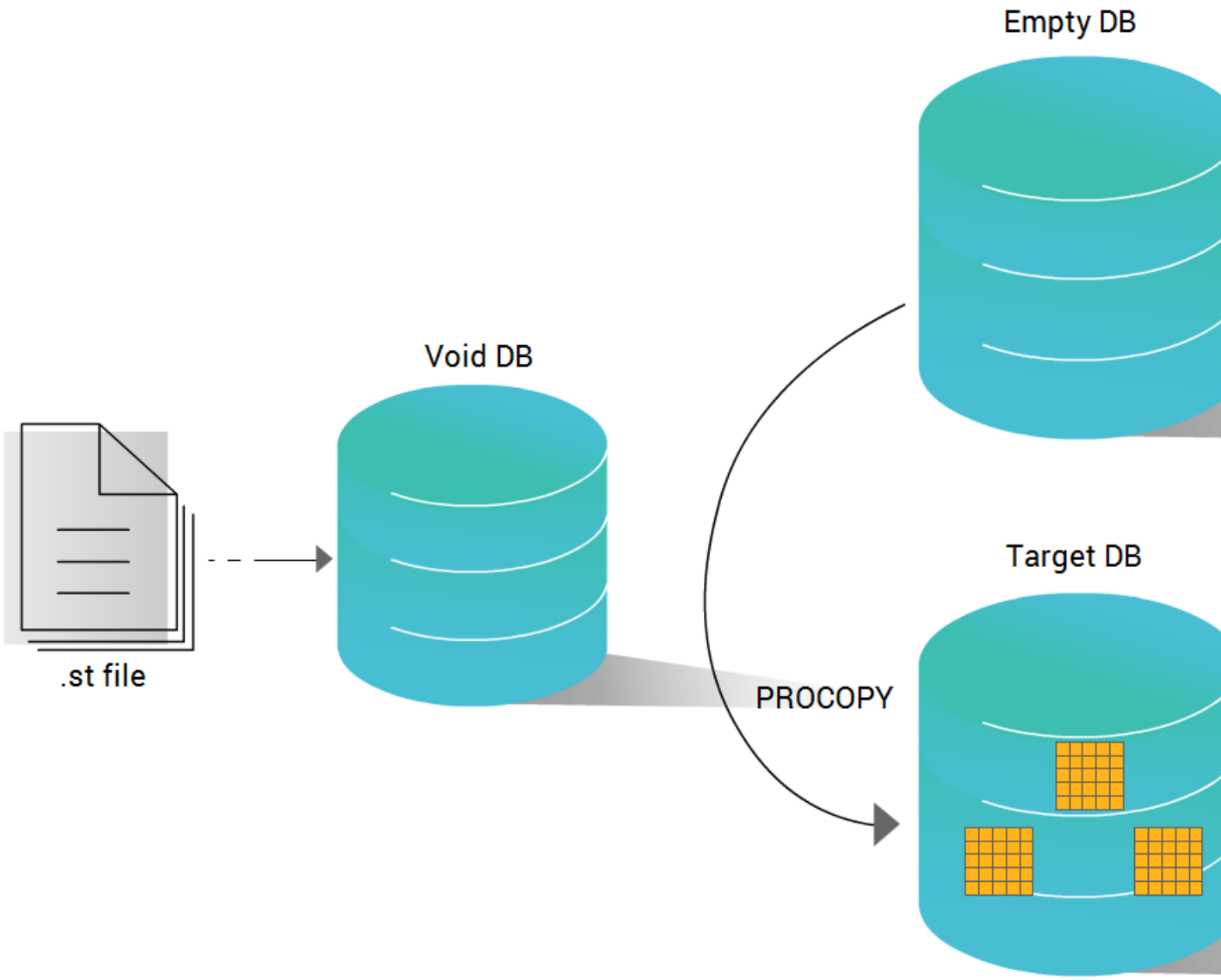
```
procopy %DLC%/sports mysports
```

In UNIX, enter:

```
procopy $DLC/sports mysports
```

The **mysports** database is created in your working directory.

Method 3-Creating a database using PROSTRCT CREATE and PROCOPY



If you want to completely reorganize an existing database, then use PROSTRCT CREATE and PROCOPY. This gives you the most control and flexibility over the creation of a new database, but it is time-consuming.

This method allows you to reorganize an existing database by fixing the limitations and performance problems. It allows you to leverage the latest OpenEdge RDBMS features and most up-to-date best practices.

With this method, you can define exactly what you want for each physical component in your database, including:

- Types, locations, sizes, and number of extents for each storage area
- Records per block and blocks per cluster for each area
- Database block size

To create a database using PROSTRCT CREATE and PROCOPY, you must perform the following tasks:

1. Create a .st file.
2. Validate the .st file.
3. Create a void database using PROSTRCT CREATE.
4. Populate the void database using PROCOPY.
5. Load definitions and data into the new database.

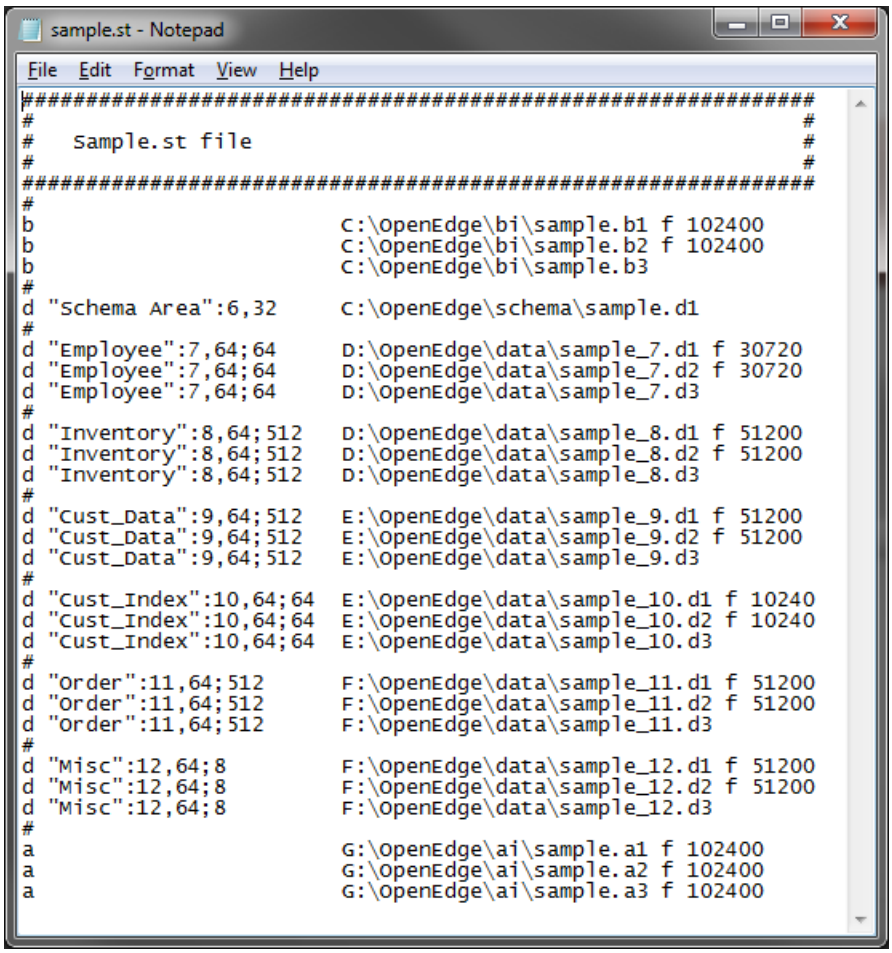
Before create a database using PROSTRCT CREATE and PROCOPY, there are a number of decisions you need to make. You will learn about them next.

Decisions to make before using PROSTRCT CREATE and PROCOPY

Before you create a new database using PROSTRCT CREATE and PROCOPY, you must make the following decisions:

- What is the name of the new database?
- What is the name of the source database?
- How many storage areas does the new database need to have?
- Where do you want to put storage areas for BI, application data, and AI?
- For each storage area for application data, you have to determine:
  - Area type, name, and number
  - Number of records per block
  - Number of blocks per cluster
  - Number of fixed and variable extents
  - Location and size of each fixed extent
  - Location of each variable extent
  - Area contents (tables and indexes)
- How many AI extents do you need to have?
- What is the database block size?

Task 1-Creating a .st file



The .st file is like the blueprint of a database. It contains the information required to create the physical files of a database.

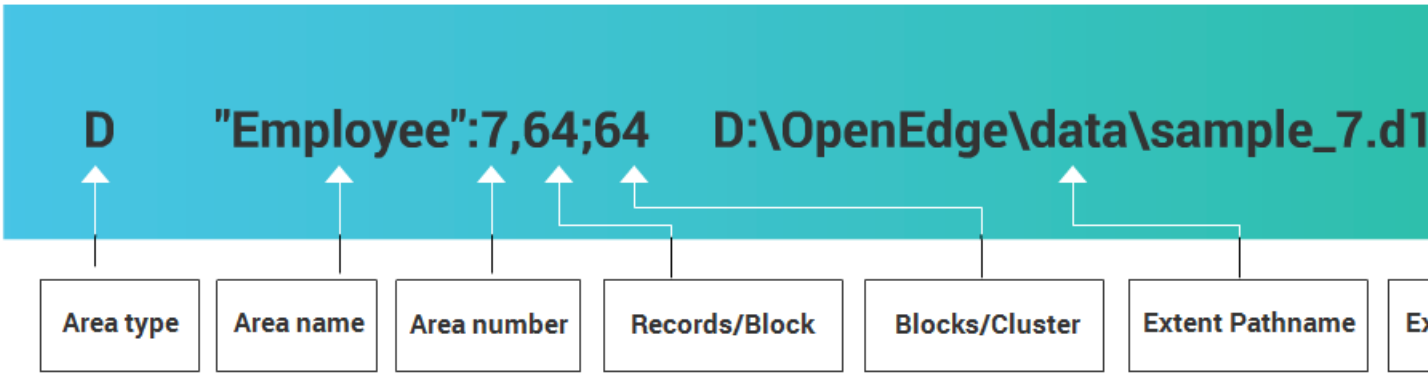
In a .st file, you can specify all the characteristics of the storage areas that make up a database. A .st file consists of the following three elements:

- Blocks—One or more lines used to describe an area
- Lines—Lines of text used to describe an extent in an area
- Tokens—Elements within a line used to describe characteristics of an extent

You can create a .st file with any text editor. Remember to save it with a .st extension.

Note: A recommended practice for creating a .st file is to copy the .st file of an existing database and edit the file to fit your storage area requirements. If you do not have an existing database, you should create a new database using one of the default existing databases, and then modify the resultant .st file to suit your needs.

Tokens



Tokens are important building blocks of a .st file. There are eight tokens, but not all of them must be used in every line:

- Area type
- Area name
- Area number
- Records per block
- Blocks per cluster
- Extent pathname
- Extent type
- Extent size

The following table lists each token with its description:

Token	Description
Area type	The type of a storage area. Valid values are: b - BI area t - Two-phase commit transaction log area d - Schema and application data areas a - AI area e - Encryption Policy Area (for databases that are enabled with Transparent Data Encryption)
Area name	The name of a storage area, in double quotes.
Area number	The number of a storage area. If you do not specify area numbers for storage areas, OpenEdge RDBMS automatically numbers the areas sequentially starting at number 7.

Token	Description
Records per block	The number of records in each database block. • This token applies only to storage areas 7 through 32,000. Valid values are 1, 2, 4, 8, 16, 32, 64, 128, and 256. • For application data areas, if you do not specify the number of records per block, the default value is 32 for the 1K, 2K, and 4K database block sizes, and 64 for the 8K database block size. • For the schema area (area 6), the number of records per block is fixed at 32 for the 1K, 2K, and 4K database block sizes, and 64 for the 8K database block size.
Blocks per cluster	The number of database blocks per cluster in a Type II data storage area. • This token applies only to areas 7 through 32,000. Valid values are 8, 64, and 512. • If you leave the token blank or specify 1, it defaults to a Type I area. Note: The schema area (area 6) does not use clusters; it is always of Type I.
Extent pathname	The absolute or relative pathname of each extent. • You do not have to specify the pathname of an extent. Specify the extent's directory; OpenEdge RDBMS automatically names it. • Use a dot (.) to indicate a relative pathname that begins in your current working directory.
Extent type	The type of the extent. Valid values are: f – Fixed extent [blank] - Variable extent with a size up to the OS file size limit
Extent size	The size of a fixed extent in kilobytes (KB). This should be a multiple of 16 times your database block size. **Note:** If you have an OpenEdge Enterprise RDBMS license, you can enable OpenEdge to support large files (up to 1TB) for your extents. If you specify fixed extents that are greater than 2GB in your .st file, when you create the database using PROSTRCT CREATE, the database will support large files. Once a database is enabled for large files, your variable extents can grow up to 1TB or up to the OS file size limit.

Rules for creating a .st file

When you create a .st file, follow these rules:

- You do not have to specify an area number for the schema area. But if you do, you must use number 6.
- For application data areas, you do not have to specify an area number for each area. But if you do, you must start at number 7.
- For area names that include spaces, use double quotes (""). The name can be up to 32 characters long.
- Group the area name, area number, records per block, and blocks per cluster, and do not leave any spaces between them. For example, "Employee":7,32;64.
- Use a colon (:) to separate the area name from the area number.
- Use a comma (,) to separate the area number from the records per block.
- Use a semicolon (;) to separate the records per block from the blocks per cluster.
- Specify the same blocks per cluster on all extents of an area or only on the first extent. By default, if the cluster size is specified only on the first extent, subsequent extents of the same storage area inherit the same value.
- If you want to place an extent in the current directory, use a dot (.) for the relative pathname. OpenEdge RDBMS places the extent in the directory. In Windows, you can use either forward slashes (/) or backward slashes (\) in the pathnames. In UNIX, you must use only forward slashes (/).

Example of a .st file

Here is an example of a .st file, `sample.st`:

```
########### Sample.st file ## #########
b C:\OpenEdge\bi\sample.b1 f 102400
b C:\OpenEdge\bi\sample.b2 f 102400
b C:\OpenEdge\bi\sample.b3
#
d "Schema Area":6,32 C:\OpenEdge\schema\sample.d1
#
d "Employee":7,64;64 D:\OpenEdge\data\sample_7.d1 f 30720
d "Employee":7,64;64 D:\OpenEdge\data\sample_7.d2 f 30720
d "Employee":7,64;64 D:\OpenEdge\data\sample_7.d3
#
d "Inventory":8,64;512 D:\OpenEdge\data\sample_8.d1 f 51200
d "Inventory":8,64;512 D:\OpenEdge\data\sample_8.d2 f 51200
d "Inventory":8,64;512 D:\OpenEdge\data\sample_8.d3
#
d "Cust_Data":9,64;512 E:\OpenEdge\data\sample_9.d1 f 51200
d "Cust_Data":9,64;512 E:\OpenEdge\data\sample_9.d2 f 51200
d "Cust_Data":9,64;512 E:\OpenEdge\data\sample_9.d3
#
d "Cust_Index":10,64;64 E:\OpenEdge\data\sample_10.d1 f 10240
d "Cust_Index":10,64;64 E:\OpenEdge\data\sample_10.d2 f 10240
d "Cust_Index":10,64;64 E:\OpenEdge\data\sample_10.d3
#
d "Order":11,64;512 F:\OpenEdge\data\sample_11.d1 f 51200
d "Order":11,64;512 F:\OpenEdge\data\sample_11.d2 f 51200
d "Order":11,64;512 F:\OpenEdge\data\sample_11.d3
#
d "Misc":12,64;8 F:\OpenEdge\data\sample_12.d1 f 51200
d "Misc":12,64;8 F:\OpenEdge\data\sample_12.d2 f 51200
d "Misc":12,64;8 F:\OpenEdge\data\sample_12.d3
#
a G:\OpenEdge\ai\sample.a1 f 102400
a G:\OpenEdge\ai\sample.a2 f 102400
a G:\OpenEdge\ai\sample.a3 f 102400
```

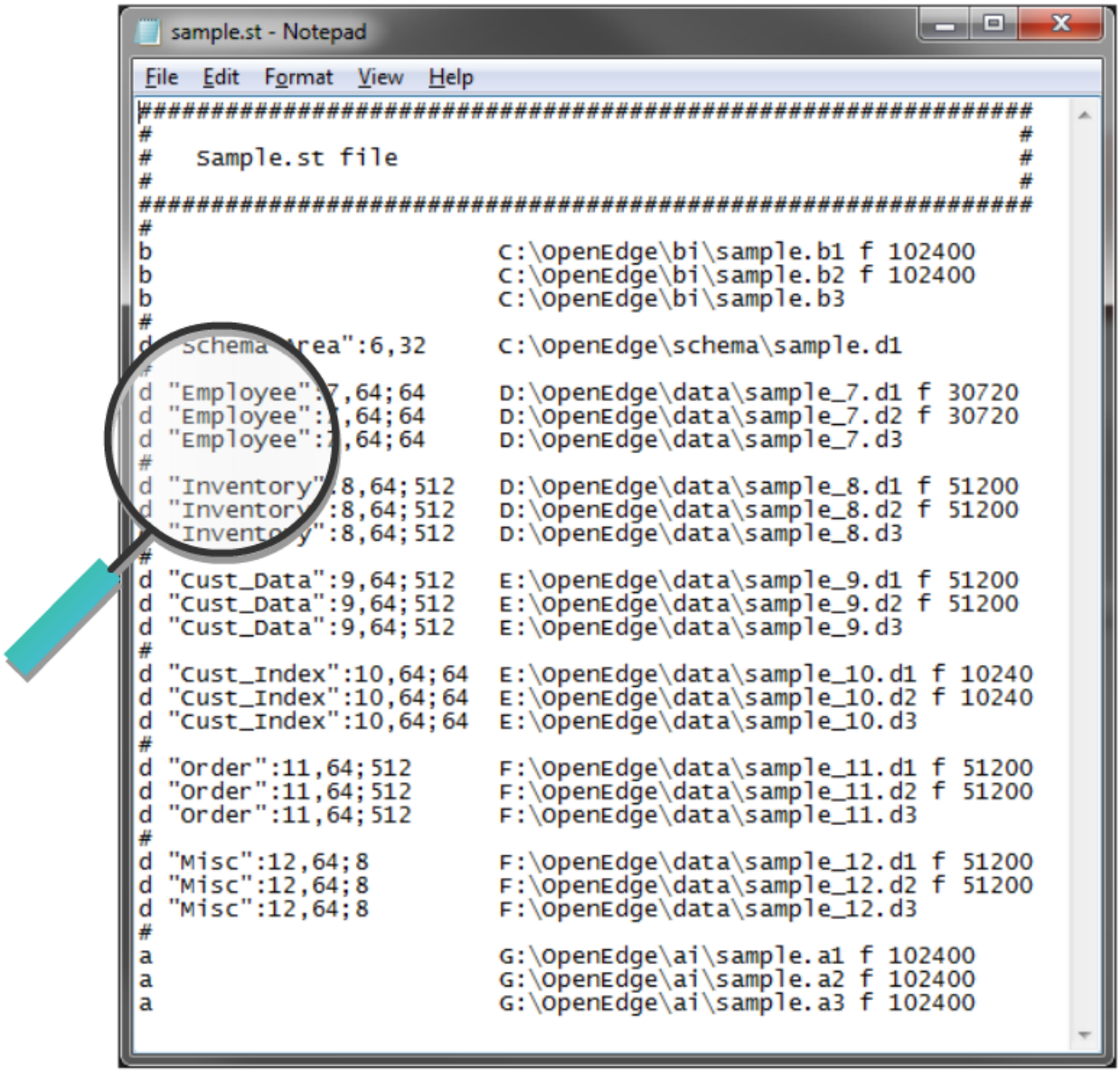
From the sample.st example, you can observe the following information:

- The BI area spans three extents. The first two BI extents are fixed at 102400K. The last BI extent is variable.
- The schema area is of Type I. It has one variable extent with 32 records per block.
- There are six application data areas, Employee (Area 7) through Misc (Area 12). Each data area has three extents, two fixed and one variable.
- All data extents are of Type II. They all have 64 records per block, but different blocks per cluster. The different cluster sizes reflect whether an area is expected to have high data growth (512), moderate data growth (64) or low data growth (8).
- The file names of the data extents follow this convention:

```
database name + "_" + storage-area-number + "." + storage-area-type + n
```

- There are three AI areas, each containing one extent. All three AI areas are fixed.
- The extents of the BI, schema, application data, and AI areas are spread across different disks (C:, D:, E:, F:, G:), thus lowering disk contention and enhancing database performance.

Task 2-Validating a .st file



After you create the .st file of a database, it is a good practice to validate it for accurate syntax and valid content. If the validation process reports any errors, you can correct the .st file. This reduces the chances of errors occurring when you create a database using the .st file.

To validate a .st file, you use the PROSTRCT CREATE -VALIDATE utility.

What PROSTRCT CREATE -VALIDATE does

The PROSTRCT CREATE -VALIDATE utility does the following:

- Examines the .st file for accurate syntax and valid content.
- Verifies that size of each extent is a multiple of 16 times the block size and larger than 32.

- Verifies that the extent sizes of Type II areas are large enough to contain at least one cluster.
- Verifies that the directories for the extents exist.
- Verifies that a database with the same name does not already exist.
- Determines if there is sufficient disk space to create new extents.
- Reports any errors that the .st file might contain.

Note: PROSTRCT CREATE -VALIDATE does not create a database.

Using PROSTRCT CREATE -VALIDATE

The command to validate a .st file is:

```
prostrct create db-name .st-file -validate
```

Where:

`db-name`	The name of the database whose .st file you want to validate.
`.st-file`	The name of the .st file that you want to validate. **Note:** If no .st file is specified, OpenEdge RDBMS looks for a .st file in your current directory that is named after the database to perform the validation.

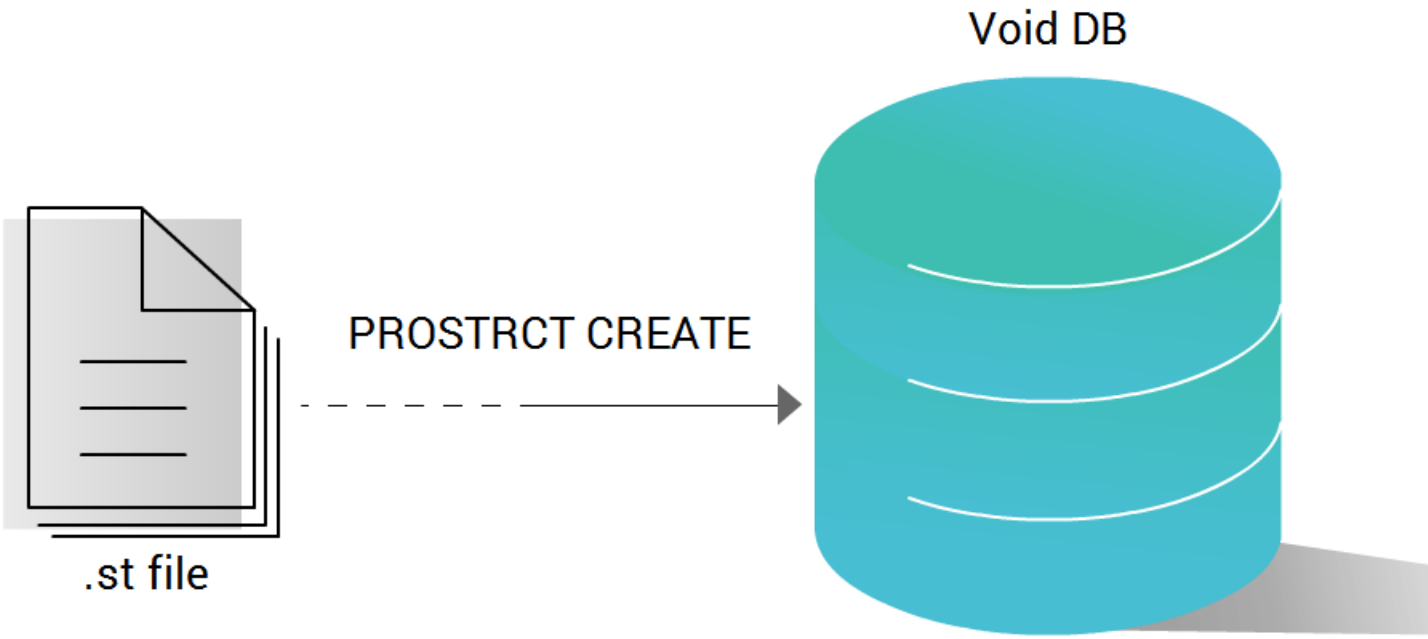
If PROSTRCT CREATE -VALIDATE issues any warnings about your extent sizes (for example, the extent sizes are not multiples of 16 times your database block size), modify the .st file to reflect the extent sizes recommended by the utility.

When you create a database using the PROSTRCT CREATE utility and a .st file that contains sizing irregularities, the utility automatically adjusts the extent sizes to be multiples of 16 times your database block size and creates the database. It does not, however, update the .st file to reflect its adjustment. Therefore, you should modify the .st file before creating the database. This ensures that the .st file is consistent with the .db file of the database.

Suppose you have an appdb.st file that you want to use to create an appdb database. To validate appdb.st before creating the database, enter:

```
prostrct create appdb appdb.st -validate
```

Task 3-Creating a void database using PROSTRCT CREATE



After you validate the .st file of a database, you use the PROSTRCT CREATE utility to create a void database. PROSTRCT CREATE reads the extent information from the .st file and creates database files that fit the descriptions of those extents.

The command to create a void database using PROSTRCT CREATE is:

```
prostrct create db–name .st-file -blocksize size-in-bytes
```

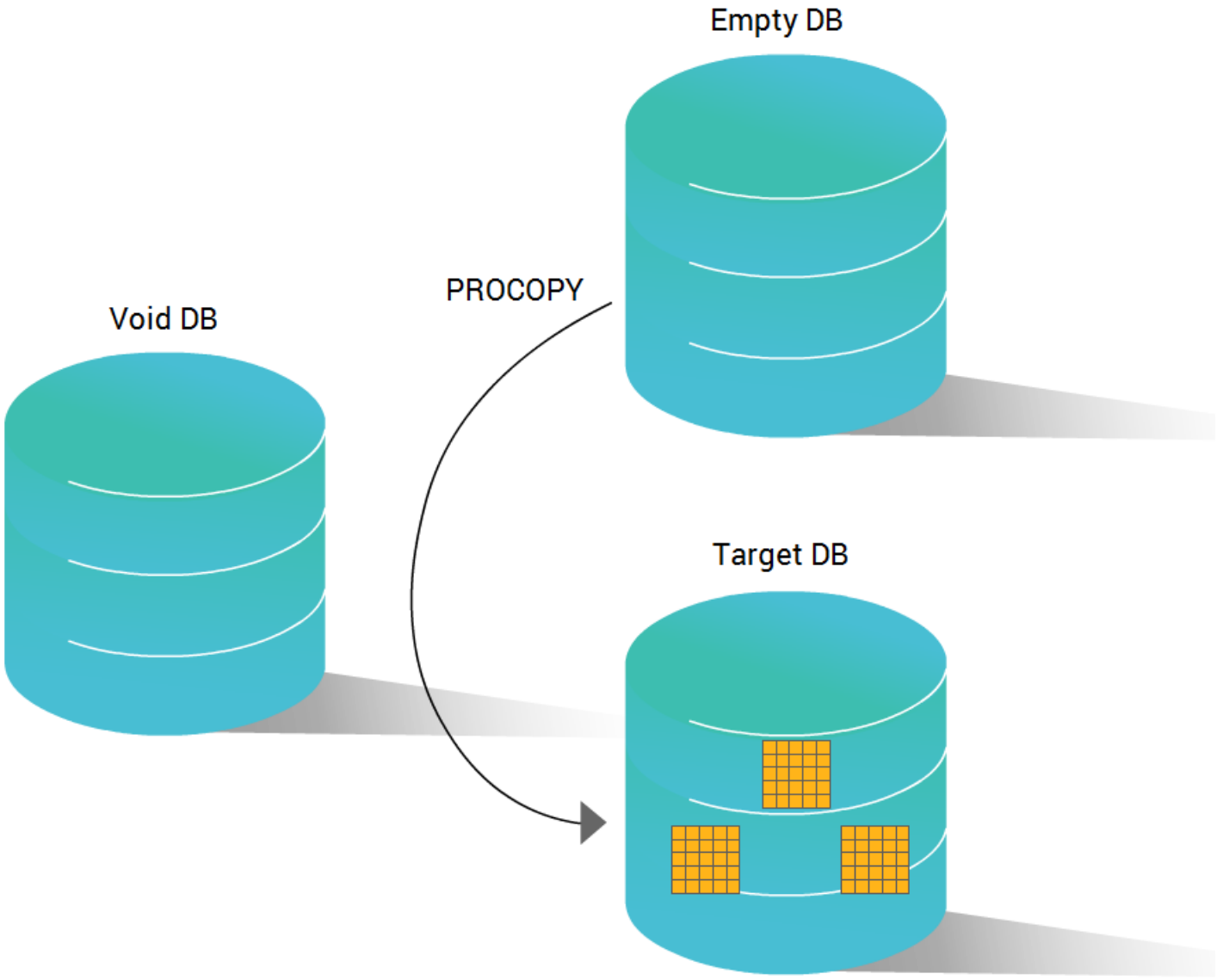
Where:

`db-name`	The name of the database that you want to create.
`.st-file`	The name of the .st file.
`size-in-bytes`	The database block size in bytes.Valid values are 1024, 2048, 4096, and 8192. It is recommended that you always use 8192. **Note:** If no block size is specified, OpenEdge RDBMS uses the OS page size.

To create a void appdb database using the appdb.st file and the 8K database block size, enter:

```
prostrct create appdb appdb.st -blocksize 8192
```

Task 4-Populating the void database using PROCOPY



A void database consists of all the extents defined in the .st file but does not contain any meta-schema information. Consequently, a void database cannot be started and is practically useless.

To populate a void database with meta-schema, you use the PROCOPY utility with a default empty database that has the same database block size as the new database.

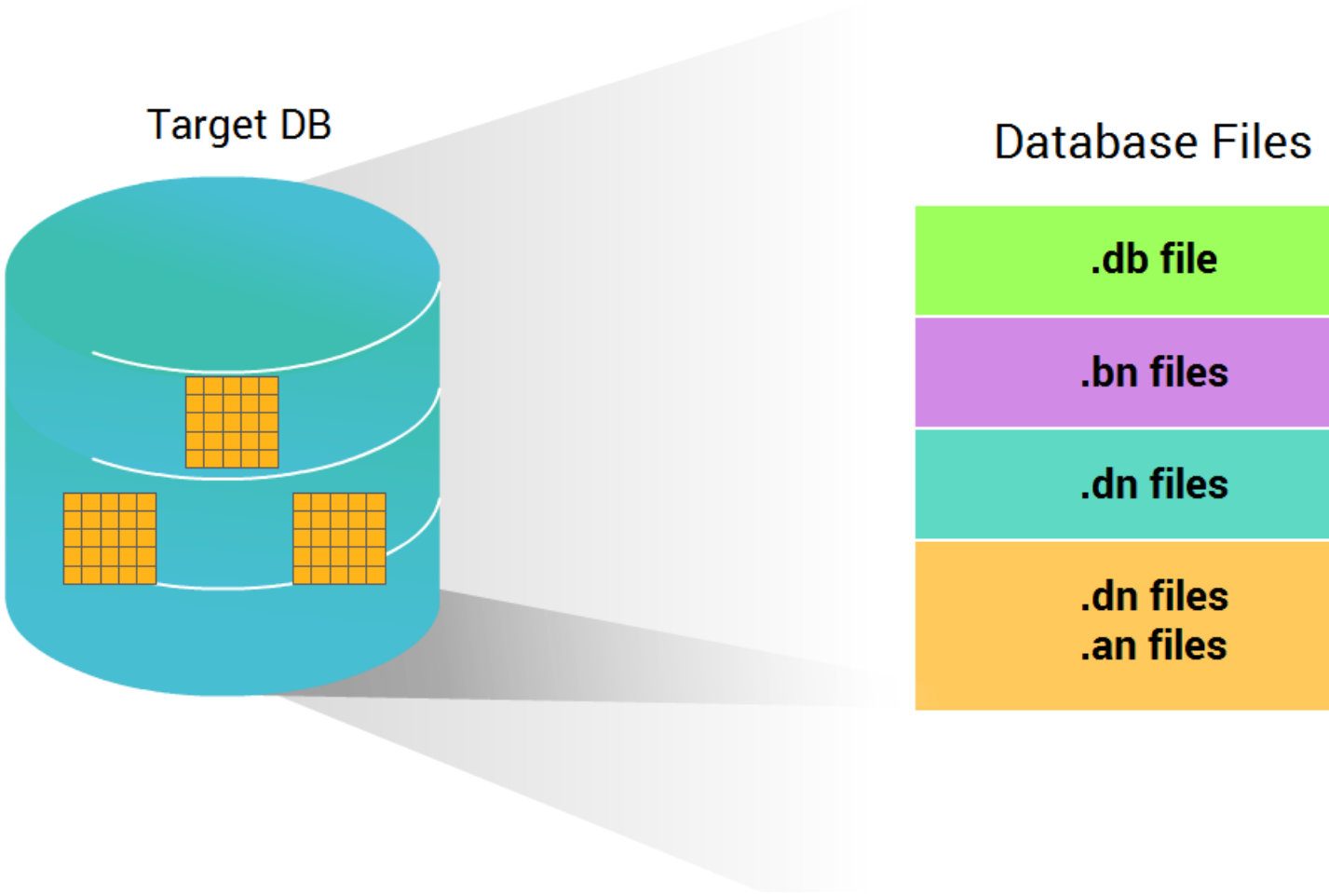
The void appdb database uses the 8K database block size. So does the empty8 database. In Windows, to copy the meta-schema from empty8 to appdb, enter:

```
procopy %DLC%/empty8 appdb
```

In UNIX, to copy the meta-schema from empty8 to appdb, enter:

```
procopy $DLC/empty8 appdb
```

Task 5-Loading definitions and data into the new database



After you verify extents of the new database, you next need to load definitions and data of the existing database from external files into the new database. This assumes that definitions and data from the existing database have been dumped to external files.

4

Deleting a database



You may have to delete a database when it becomes obsolete. To do this, you use the PRODEL utility.

When you delete a database using PRODEL, OpenEdge RDBMS displays a message that it will delete the BI, data, and AI files as well as the .db, .lg, and .lic files for that database. If you respond in the affirmative, OpenEdge RDBMS deletes those files.

Note: PRODEL does not delete the .st file of the database. To delete it, you must use an OS command or your file system manager.

Before you delete a database, ensure that it is offline. The command to delete a database is:

prodel db-name

Suppose the mysports database has become obsolete and you want to delete it. Enter:

```
prodel mysports
```

All mysports database files are deleted except the mysports.st file.