11

Physical Storage Structures

This chapter describes the primary physical database structures of an Oracle database. Physical structures are viewable at the operating system level.

This chapter contains the following sections:

- Introduction to Physical Storage Structures
- Overview of Data Files
- Overview of Control Files
- Overview of the Online Redo Log

Introduction to Physical Storage Structures

One characteristic of an RDBMS is the independence of logical data structures such as **tables**, **views**, and **indexes** from physical storage structures. Because physical and logical structures are separate, you can manage physical storage of data without affecting access to logical structures. For example, renaming a database file does not rename the tables stored in it.

An **Oracle database** is a set of files that store Oracle data in persistent disk storage. This section discusses the database files generated when you issue a `CREATE DATABASE` statement:

- Data files and temp files

  A **data file** is a physical file on disk that was created by Oracle Database and contains data structures such as tables and indexes. A **temp file** is a data file that belongs to a temporary tablespace. The data is written to these files in an Oracle proprietary format that cannot be read by other programs.

- Control files

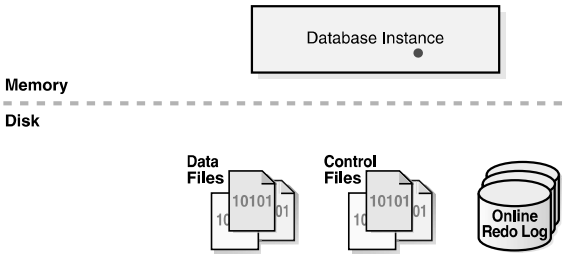
  A **control file** is a root file that tracks the physical components of the database.

- Online redo log files

  The **online redo log** is a set of files containing records of changes made to data.

A database **instance** is a set of memory structures that manage database files. Figure 11–1 shows the relationship between the instance and the files that it manages.

Figure 11–1 Database Instance and Database Files



See Also:

- *Oracle Database Administrator's Guide* to learn how to create a database
- *Oracle Database SQL Language Reference* for `CREATE DATABASE` semantics and syntax

Mechanisms for Storing Database Files

Several mechanisms are available for allocating and managing the storage of these files. The most common mechanisms include:

- Oracle Automatic Storage Management (Oracle ASM)

  Oracle ASM includes a file system designed exclusively for use by Oracle Database. "Oracle Automatic Storage Management (Oracle ASM)" on page 11-3 describes Oracle ASM.

- Operating system file system

  Most Oracle databases store files in a **file system**, which is a data structure built inside a contiguous disk address space. All operating systems have **file managers** that allocate and deallocate disk space into files within a file system.

  A file system enables disk space to be allocated to many files. Each file has a name and is made to appear as a contiguous address space to applications such as Oracle Database. The database can create, read, write, resize, and delete files.

  A file system is commonly built on top of a **logical volume** constructed by a software package called a **logical volume manager (LVM)**. The LVM enables pieces of multiple physical disks to be combined into a single contiguous address space that appears as one disk to higher layers of software.

- Raw device

  Raw devices are disk partitions or logical volumes not formatted with a file system. The primary benefit of raw devices is the ability to perform **direct I/O** and to write larger buffers. In direct I/O, applications write to and read from the storage device directly, bypassing the operating system buffer cache.

  Note: Many file systems now support direct I/O for databases and other applications that manage their own caches. Historically, raw devices were the only means of implementing direct I/O.

- Cluster file system

A **cluster file system** is software that enables multiple computers to share file storage while maintaining consistent space allocation and file content. In an Oracle RAC environment, a cluster file system makes shared storage appears as a file system shared by many computers in a clustered environment. With a cluster file system, the failure of a computer in the cluster does not make the file system unavailable. In an operating system file system, however, if a computer sharing files through NFS or other means fails, then the file system is unavailable.

A database employs a combination of the preceding storage mechanisms. For example, a database could store the control files and online redo log files in a traditional file system, some user data files on raw partitions, the remaining data files in Oracle ASM, and archived the redo log files to a cluster file system.

See Also:

- *Oracle Database 2 Day DBA* to learn how to view database storage structures with Oracle Enterprise Manager (Enterprise Manager)
- *Oracle Database Administrator's Guide* to view database storage structures by querying database views

Oracle Automatic Storage Management (Oracle ASM)

Oracle ASM is a high-performance, ease-of-management storage solution for Oracle Database files. Oracle ASM is a volume manager and provides a file system designed exclusively for use by the database.

Oracle ASM provides several advantages over conventional file systems and storage managers, including the following:

- Simplifies storage-related tasks such as creating and laying out databases and managing disk space
- Distributes data across physical disks to eliminate hot spots and to provide uniform performance across the disks
- Rebalances data automatically after storage configuration changes

To use Oracle ASM, you allocate partitioned disks for Oracle Database with preferences for striping and mirroring. Oracle ASM manages the disk space, distributing the I/O load across all available resources to optimize performance while removing the need for manual I/O tuning. For example, you can increase the size of the disk for the database or move parts of the database to new devices without having to shut down the database.

Oracle ASM Storage Components

Oracle Database can store a data file as an **Oracle ASM file** in an **Oracle ASM disk group**, which is a collection of disks that Oracle ASM manages as a unit. Within a disk group, Oracle ASM exposes a file system interface for database files.

Figure 11–2 shows the relationships between storage components in a database that uses Oracle ASM. The diagram depicts the relationship between an Oracle ASM file and a data file, although Oracle ASM can store other types of files. The crow's foot notation represents a one-to-many relationship.

Figure 11–2 Oracle ASM Components

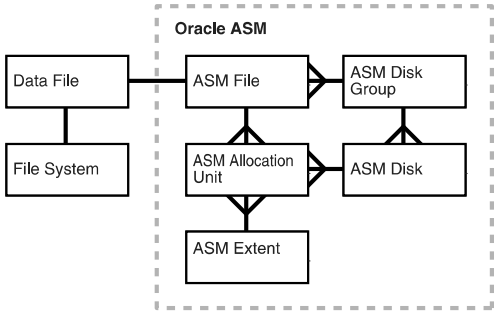


Figure 11–2 illustrates the following Oracle ASM concepts:

- Oracle ASM Disks

  An **Oracle ASM disk** is a storage device that is provisioned to an Oracle ASM disk group. An Oracle ASM disk can be a physical disk or partition, a Logical Unit Number (LUN) from a storage array, a logical volume, or a network-attached file.

  Oracle ASM disks can be added or dropped from a disk group while the database is running. When you add a disk to a disk group, you either assign a disk name or the disk is given an Oracle ASM disk name automatically.

- Oracle ASM Disk Groups

  An **Oracle ASM disk group** is a collection of Oracle ASM disks managed as a logical unit. The data structures in a disk group are self-contained and consume some disk space in a disk group.

  Within a disk group, Oracle ASM exposes a file system interface for Oracle database files. The content of files that are stored in a disk group are evenly distributed, or striped, to eliminate hot spots and to provide uniform performance across the disks. The performance is comparable to the performance of raw devices.

- Oracle ASM Files

  An **Oracle ASM file** is a file stored in an Oracle ASM disk group. Oracle Database communicates with Oracle ASM in terms of files. The database can store data files, control files, online redo log files, and other types of files as Oracle ASM files. When requested by the database, Oracle ASM creates an Oracle ASM file and assigns it a fully qualified name beginning with a plus sign (+) followed by a disk group name, as in `+DISK1`.

  > **Note:** Oracle ASM files can coexist with other storage management options such as raw disks and third-party file systems. This capability simplifies the integration of Oracle ASM into pre-existing environments.

- Oracle ASM Extents

  An **Oracle ASM extent** is the raw storage used to hold the contents of an Oracle ASM file. An Oracle ASM file consists of one or more file extents. Each Oracle ASM extent consists of one or more allocation units on a specific disk.

Note: An Oracle ASM extent is different from the **extent** used to store data in a **segment**.

- Oracle ASM Allocation Units

  An **allocation unit** is the fundamental unit of allocation within a disk group. An allocation unit is the smallest contiguous disk space that Oracle ASM allocates. One or more allocation units form an Oracle ASM extent.

  See Also:

  - *Oracle Database 2 Day DBA* to learn how to administer Oracle ASM disks with Oracle Enterprise Manager (Enterprise Manager)
  - *Oracle Automatic Storage Management Administrator's Guide* to learn more about Oracle ASM

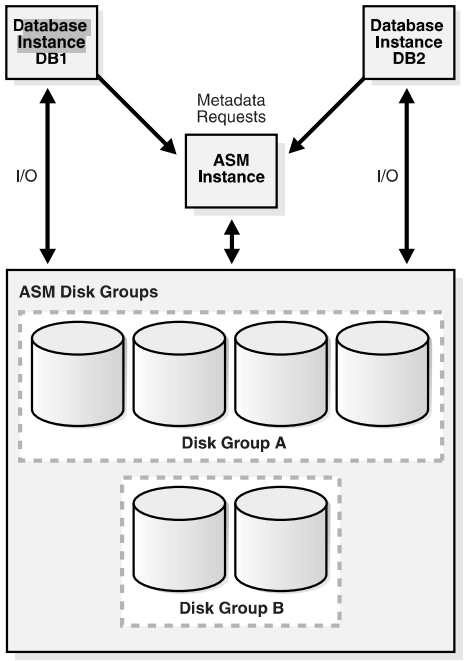
Oracle ASM Instances

An **Oracle ASM instance** is a special Oracle instance that manages Oracle ASM disks. Both the ASM and the database instances require shared access to the disks in an ASM disk group. ASM instances manage the metadata of the disk group and provide file layout information to the database instances. Database instances direct I/O to ASM disks without going through an ASM instance.

An ASM instance is built on the same technology as a database instance. For example, an ASM instance has a **system global area (SGA)** and background processes that are similar to those of a database instance. However, an ASM instance cannot mount a database and performs fewer tasks than a database instance.

Figure 11–3 shows a single-node configuration with one Oracle ASM instance and two database instances, each associated with a different single-instance database. The ASM instance manages the metadata and provides space allocation for the ASM files storing the data for the two databases. One ASM disk group has four ASM disks and the other has two disks. Both database instances can access the disk groups.

Figure 11–3 Oracle ASM Instance and Database Instances



See Also:

- *Oracle Database 2 Day DBA* to learn how to administer Oracle ASM disks with Oracle Enterprise Manager (Enterprise Manager)
- *Oracle Automatic Storage Management Administrator's Guide* to learn more about Oracle ASM

Oracle Managed Files and User-Managed Files

Oracle Managed Files is a file naming strategy that enables you to specify operations in terms of database objects rather than file names. For example, you can create a tablespace without specifying the names of its data files. In this way, Oracle Managed Files eliminates the need for administrators to directly manage the operating system files in a database. Oracle ASM requires Oracle Managed Files.

Note: This feature does not affect the creation or naming of administrative files such as trace files, audit files, and alert logs (see "Overview of Diagnostic Files" on page 13-18).

With **user-managed files**, you directly manage the operating system files in the database. You make the decisions regarding file structure and naming. For example, when you create a tablespace you set the name and path of the tablespace data files.

Through initialization parameters, you specify the file system directory for a specific type of file. The Oracle Managed Files feature ensures that the database creates a unique file and deletes it when no longer needed. The database internally uses standard file system interfaces to create and delete files for data files and temp files, control files, and recovery-related files stored in the **fast recovery area**.

Oracle Managed Files does not eliminate existing functionality. You can create new files while manually administering old files. Thus, a database can have a mixture of Oracle Managed Files and user-managed files.

> **See Also:** *Oracle Database Administrator's Guide* to learn how to use Oracle Managed Files

Overview of Data Files

At the operating system level, Oracle Database stores database data in **data files**. Every database must have at least one data file.

Use of Data Files

Part I, "Oracle Relational Data Structures" explains the logical structures in which users store data, the most important of which are tables. Each nonpartitioned **schema object** and each partition of an object is stored in its own **segment**.

For ease of administration, Oracle Database allocates space for user data in **tablespaces**, which like segments are logical storage structures. Each segment belongs to only one tablespace. For example, the data for a nonpartitioned table is stored in a single segment, which is turn is stored in one tablespace.

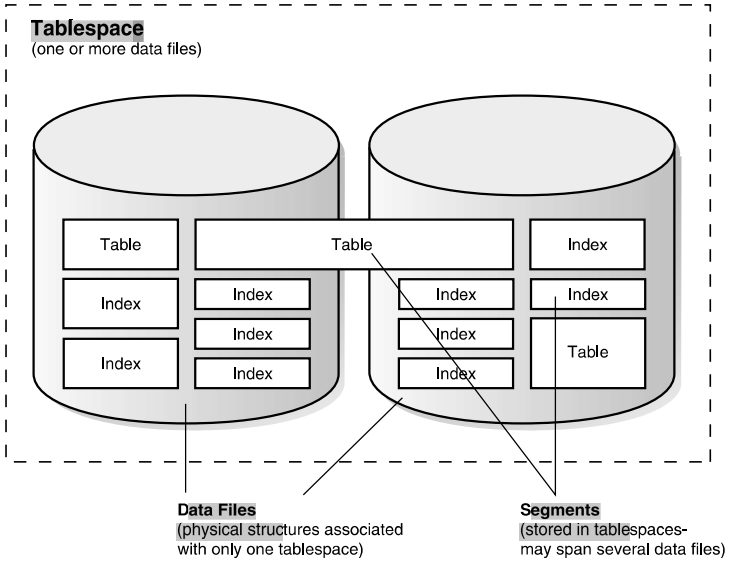
Oracle Database physically stores tablespace data in data files. Tablespaces and data files are closely related, but have important differences:

- Each tablespace consists of one or more data files, which conform to the operating system in which Oracle Database is running.
- The data for a database is collectively stored in the data files located in each tablespace of the database.
- A segment can span one or more data files, but it cannot span multiple tablespaces.
- A database must have the `SYSTEM` and `SYSAUX` tablespaces. Oracle Database automatically allocates the first data files of any database for the `SYSTEM` tablespace during database creation.

  The `SYSTEM` tablespace contains the **data dictionary**, a set of tables that contains database metadata. Typically, a database also has an **undo tablespace** and a temporary tablespace (usually named `TEMP`).

Figure 11–4 shows the relationship between tablespaces, data files, and segments.

Figure 11–4 Data Files and Tablespaces



See Also:

- "Overview of Tablespaces" on page 12-30
- *Oracle Database Administrator's Guide* and *Oracle Database 2 Day DBA* to learn how to manage data files

Permanent and Temporary Data Files

A **permanent tablespace** contains persistent schema objects. Objects in permanent tablespaces are stored in data files.

A **temporary tablespace** contains schema objects only for the duration of a session. Locally managed temporary tablespaces have temporary files (**temp file**s), which are special files designed to store data in hash, sort, and other operations. Temp files also store result set data when insufficient space exists in memory.

Temp files are similar to permanent data files, with the following exceptions:

- Permanent database objects such as tables are never stored in temp files.
- Temp files are always set to `NOLOGGING` mode, which means that they never have redo generated for them. Media recovery does not recognize temp files.
- You cannot make a temp file read-only.
- You cannot create a temp file with the `ALTER DATABASE` statement.
- When you create or resize temp files, they are not always guaranteed allocation of disk space for the file size specified. On file systems such as Linux and UNIX, temp files are created as **sparse files**. In this case, disk blocks are allocated not at file creation or resizing, but as the blocks are accessed for the first time.

Caution: Sparse files enable fast temp file creation and resizing; however, the disk could run out of space later when the temp files are accessed.

- Temp file information is shown in the data dictionary view `DBA_TEMP_FILES` and the dynamic performance view `V$TEMPFILE`, but not in `DBA_DATA_FILES` or the `V$DATAFILE` view.

See Also:

- "Temporary Tablespaces" on page 12-34
- *Oracle Database Administrator's Guide* to learn how to manage temp files

Online and Offline Data Files

Every data file is either **online** (available) or **offline** (unavailable). You can alter the availability of individual data files or temp files by taking them offline or bringing them online. Offline data files cannot be accessed until they are brought back online.

Administrators may take data files offline for many reasons, including performing offline backups, renaming a data file, or block corruption. The database takes a data file offline automatically if the database cannot write to it.

Like a data file, a tablespace itself is offline or online. When you take a data file offline in an online tablespace, the tablespace itself remains online. You can make all data files of a tablespace temporarily unavailable by taking the tablespace itself offline

See Also:

- "Online and Offline Tablespaces" on page 12-35
- *Oracle Database Administrator's Guide* to learn how to alter data file availability

Data File Structure

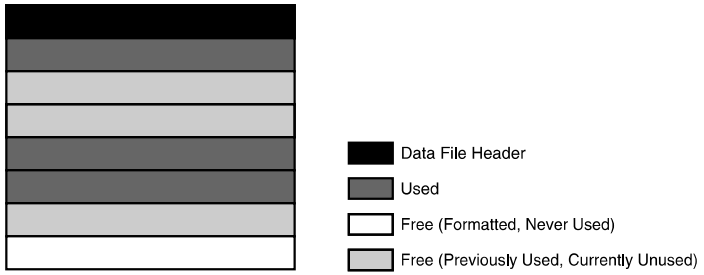
Oracle Database creates a data file for a tablespace by allocating the specified amount of disk space plus the overhead for the **data file header**. The operating system under which Oracle Database runs is responsible for clearing old information and authorizations from a file before allocating it to the database.

The data file header contains metadata about the data file such as its size and **checkpoint SCN**. Each header contains an **absolute file number** and a **relative file number**. The absolute file number uniquely identifies the data file within the database. The relative file number uniquely identifies a data file within a tablespace.

When Oracle Database first creates a data file, the allocated disk space is formatted but contains no user data. However, the database reserves the space to hold the data for future segments of the associated tablespace. As the data grows in a tablespace, Oracle Database uses the free space in the data files to allocate **extents** for the segment.

Figure 11–5 illustrates the different types of space in a data file. Extents are either used, which means they contain segment data, or free, which means they are available for reuse. Over time, updates and deletions of objects within a tablespace can create pockets of empty space that individually are not large enough to be reused for new data. This type of empty space is referred to as **fragmented free space**.

Figure 11–5 Space in a Data File



See Also: *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* to learn how to view data file information

Overview of Control Files

The database **control file** is a small binary file associated with only one database. Each database has one unique control file, although it may maintain identical copies of it.

Use of Control Files

The control file is the root file that Oracle Database uses to find database files and to manage the state of the database generally. A control file contains information such as the following:

- The database name and database unique identifier (DBID)
- The time stamp of database creation
- Information about data files, online redo log files, and **archived redo log files**
- Tablespace information
- RMAN backups

The control file serves the following purposes:

- It contains information about data files, online redo log files, and so on that are required to open the database.

  The control file tracks structural changes to the database. For example, when an administrator adds, renames, or drops a data file or online redo log file, the database updates the control file to reflect this change.

- It contains metadata that must be accessible when the database is not open.

  For example, the control file contains information required to recover the database, including checkpoints. A **checkpoint** indicates the **SCN** in the redo stream where **instance recovery** would be required to begin (see "Overview of Instance Recovery" on page 13-12). Every committed change before a checkpoint SCN is guaranteed to be saved on disk in the data files. At least every three seconds the checkpoint process records information in the control file about the checkpoint position in the online redo log.

Oracle Database reads and writes to the control file continuously during database use and must be available for writing whenever the database is open. For example, recovering a database involves reading from the control file the names of all the data

files contained in the database. Other operations, such as adding a data file, update the information stored in the control file.

See Also:

- "Checkpoint Process (CKPT)" on page 15-10
- *Oracle Database Administrator's Guide* to learn how to manage the control file

Multiple Control Files

Oracle Database enables multiple, identical control files to be open concurrently and written for the same database. By multiplexing a control file on different disks, the database can achieve redundancy and thereby avoid a single point of failure.

Note: Oracle recommends that you maintain multiple control file copies, each on a different disk.

If a control file becomes unusable, then the database instance fails when it attempts to access the damaged control file. When other current control file copies exist, the database can be remounted and opened without **media recovery**. If *all* control files of a database are lost, however, then the instance fails and media recovery is required. Media recovery is not straightforward if an older backup of a control file must be used because a current copy is not available.

See Also:

- *Oracle Database Administrator's Guide* to learn how to maintain multiple control files
- *Oracle Database Backup and Recovery User's Guide* to learn how to back up and restore control files

Control File Structure

Information about the database is stored in different **sections** of the control file. Each section is a set of **records** about an aspect of the database. For example, one section in the control file tracks data files and contains a set of records, one for each data file. Each section is stored in multiple logical **control file blocks**. Records can span blocks within a section.

The control file contains the following types of records:

- Circular reuse records

  These records contain noncritical information that is eligible to be overwritten if needed. When all available record slots are full, the database either expands the control file to make room for a new record or overwrites the oldest record. Examples include records about archived redo log files and RMAN backups.

- Noncircular reuse records

  These records contain critical information that does not change often and cannot be overwritten. Examples of information include tablespaces, data files, online redo log files, and redo threads. Oracle Database never reuses these records unless the corresponding object is dropped from the tablespace.

As explained in "Overview of the Dynamic Performance Views" on page 6-5, you can query the dynamic performance views, also known as `V$` views, to view the

information stored in the control file. For example, you can query V$DATABASE to obtain the database name and DBID. However, only the database can modify the information in the control file.

Reading and writing the control file blocks is different from reading and writing **data blocks**. For the control file, Oracle Database reads and writes directly from the disk to the **program global area (PGA)**. Each process allocates a certain amount of its PGA memory for control file blocks.

See Also:

- *Oracle Database Reference* to learn about the V$CONTROLFILE_RECORD_SECTION view
- *Oracle Database Reference* to learn about the CONTROL_FILE_RECORD_KEEP_TIME initialization parameter

Overview of the Online Redo Log

The most crucial structure for recovery is the online **redo log**, which consists of two or more preallocated files that store changes to the database as they occur. The online redo log records changes to the data files.

Use of the Online Redo Log

The database maintains online redo log files to protect against data loss. Specifically, after an **instance failure** the online redo log files enable Oracle Database to recover committed data not yet written to the data files.

Oracle Database writes every transaction synchronously to the **redo log buffer**, which is then written to the online redo logs. The contents of the log include uncommitted transactions, undo data, and schema and object management statements.

Oracle Database uses the online redo log only for recovery. However, administrators can query online redo log files through a SQL interface in the Oracle LogMiner utility (see "Oracle LogMiner" on page 18-8). Redo log files are a useful source of historical information about database activity.

See Also: "Overview of Instance Recovery" on page 13-12

How Oracle Database Writes to the Online Redo Log

The online redo log for a database instance is called a **redo thread**. In single-instance configurations, only one instance accesses a database, so only one redo thread is present. In an Oracle Real Application Clusters (Oracle RAC) configuration, however, two or more instances concurrently access a database, with each instance having its own redo thread. A separate redo thread for each instance avoids contention for a single set of online redo log files.

An online redo log consists of two or more online redo log files. Oracle Database requires a minimum of two files to guarantee that one is always available for writing while the other is being archived (if the database is in **ARCHIVELOG mode**).

See Also: *Oracle Database 2 Day + Real Application Clusters Guide* and *Oracle Real Application Clusters Administration and Deployment Guide* to learn about online redo log groups in Oracle RAC

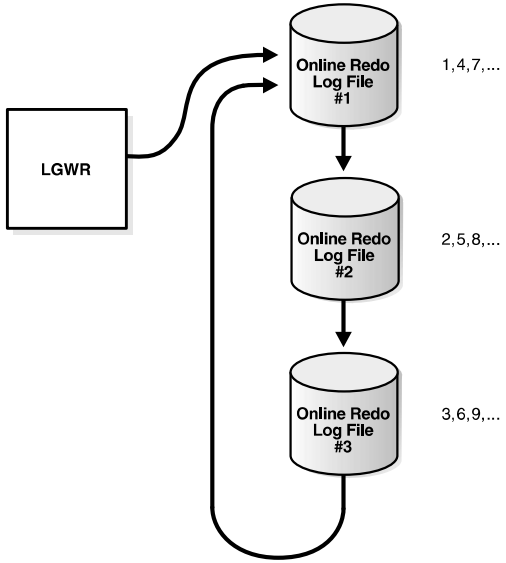
Online Redo Log Switches

Oracle Database uses only one online redo log file at a time to store records written from the redo log buffer. The online redo log file to which the **log writer (LGWR)** process is actively writing is called the **current** online redo log file.

A **log switch** occurs when the database stops writing to one online redo log file and begins writing to another. Normally, a switch occurs when the current online redo log file is full and writing must continue. However, you can configure log switches to occur at regular intervals, regardless of whether the current online redo log file is filled, and force log switches manually.

Log writer writes to online redo log files circularly. When log writer fills the last available online redo log file, the process writes to the first log file, restarting the cycle. Figure 11–6 illustrates the circular writing of the redo log.

Figure 11–6 Reuse of Online Redo Log Files



The numbers in Figure 11–6 shows the sequence in which LGWR writes to each online redo log file. The database assigns each file a new **log sequence number** when a log switches and log writers begins writing to it. When the database reuses an online redo log file, this file receives the next available log sequence number.

Filled online redo log files are available for reuse depending on the archiving mode:

- If archiving is disabled, which means that the database is in `NOARCHIVELOG` mode, then a filled online redo log file is available after the changes recorded in it have been **checkpointed** (written) to disk by **database writer (DBW)**.
- If archiving is enabled, which means that the database is in **ARCHIVELOG mode**, then a filled online redo log file is available to log writer after the changes have been written to the data files *and* the file has been archived.

In some circumstances, log writer may be prevented from reusing an existing online redo log file. For example, an online redo log file may be **active** (required for instance

recovery) rather than **inactive** (not required for instance recovery). Also, an online redo log file may be in the process of being cleared.

See Also:

- "Overview of Background Processes" on page 15-7
- *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* to learn how to manage the online redo log

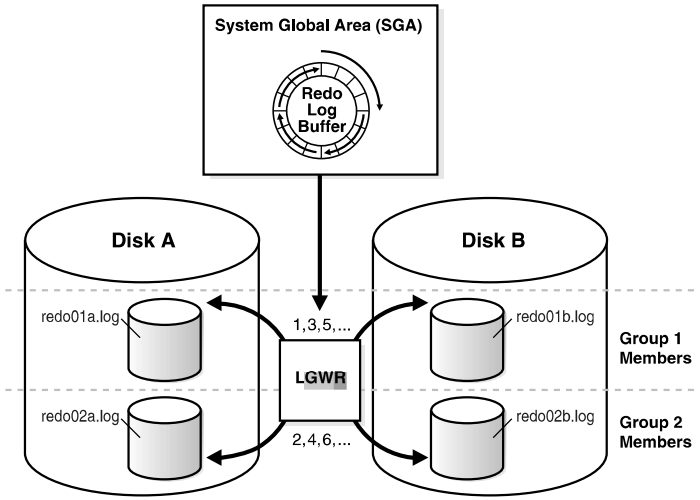
Multiple Copies of Online Redo Log Files

Oracle Database can automatically maintain two or more identical copies of the online redo log in separate locations. An **online redo log group** consists of an online redo log file and its redundant copies. Each identical copy is a **member** of the online redo log group. Each group is defined by a number, such as group 1, group 2, and so on.

Maintaining multiple members of an online redo log group protects against the loss of the redo log. Ideally, the locations of the members should be on separate disks so that the failure of one disk does not cause the loss of the entire online redo log.

In Figure 11–7, `A_LOG1` and `B_LOG1` are identical members of group 1, while `A_LOG2` and `B_LOG2` are identical members of group 2. Each member in a group must be the same size. LGWR writes concurrently to group 1 (members `A_LOG1` and `B_LOG1`), then writes concurrently to group 2 (members `A_LOG2` and `B_LOG2`), then writes to group 1, and so on. LGWR never writes concurrently to members of different groups.

Figure 11–7 Multiple Copies of Online Redo Log Files



Note: Oracle recommends that you multiplex the online redo log. The loss of log files can be catastrophic if recovery is required. When you multiplex the online redo log, the database must increase the amount of I/O it performs. Depending on your system, this additional I/O may impact overall database performance.

See Also: *Oracle Database Administrator's Guide* to learn how to maintain multiple copies of the online redo log files

Archived Redo Log Files

An **archived redo log file** is a copy of a filled member of an online redo log group. This file is not considered part of the database, but is an offline copy of an online redo log file created by the database and written to a user-specified location.

Archived redo log files are a crucial part of a backup and recovery strategy. You can use archived redo log files to:

- Recover a database backup
- Update a **standby database** (see "Computer Failures" on page 17-8)
- Obtain information about the history of a database using the LogMiner utility (see "Oracle LogMiner" on page 18-8)

Archiving is the operation of generating an archived redo log file. Archiving is either automatic or manual and is only possible when the database is in `ARCHIVELOG` mode.

An archived redo log file includes the redo entries and the log sequence number of the identical member of the online redo log group. In Figure 11–7, files `A_LOG1` and `B_LOG1` are identical members of Group 1. If the database is in `ARCHIVELOG` mode, and if automatic archiving is enabled, then the **archiver process (ARCn)** will archive one of these files. If `A_LOG1` is corrupted, then the process can archive `B_LOG1`. The archived redo log contains a copy of every group created since you enabled archiving.

See Also:

- "Data File Recovery" on page 18-14
- *Oracle Database Administrator's Guide* to learn how to manage the archived redo log

Structure of the Online Redo Log

Online redo log files contain **redo records**. A redo record is made up of a group of **change vectors**, each of which describes a change to a **data block**. For example, an update to a salary in the `employees` table generates a redo record that describes changes to the **data segment** block for the table, the undo segment data block, and the **transaction table** of the undo segments.

The redo records have all relevant metadata for the change, including the following:

- **SCN** and time stamp of the change
- Transaction ID of the transaction that generated the change
- SCN and time stamp when the transaction committed (if it committed)
- Type of operation that made the change
- Name and type of the modified data segment

See Also: "Overview of Data Blocks" on page 12-6

12

Logical Storage Structures

This chapter describes the nature of and relationships among logical storage structures. These structures are created and recognized by Oracle Database and are not known to the operating system.

This chapter contains the following sections:

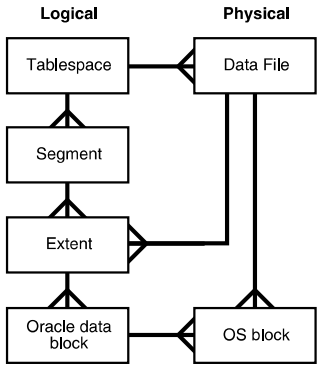
- Introduction to Logical Storage Structures
- Overview of Data Blocks
- Overview of Extents
- Overview of Segments
- Overview of Tablespaces

Introduction to Logical Storage Structures

Oracle Database allocates logical space for all data in the database. The logical units of database space allocation are data blocks, extents, segments, and tablespaces. At a physical level, the data is stored in data files on disk (see Chapter 11, "Physical Storage Structures"). The data in the data files is stored in operating system blocks.

Figure 12–1 is an entity-relationship diagram for physical and logical storage. The crow's foot notation represents a one-to-many relationship.

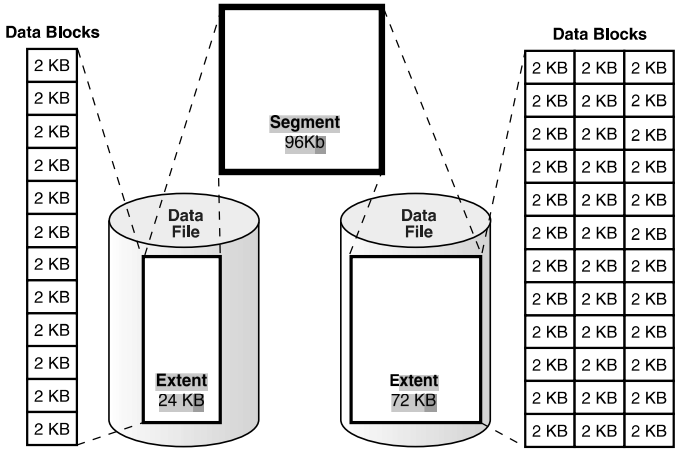
Figure 12–1 Logical and Physical Storage



Logical Storage Hierarchy

Figure 12–2 shows the relationships among data blocks, extents, and segments within a tablespace. In this example, a segment has two extents stored in different data files.

Figure 12–2 Segments, Extents, and Data Blocks Within a Tablespace



At the finest level of granularity, Oracle Database stores data in data blocks. One logical **data block** corresponds to a specific number of bytes of physical disk space, for example, 2 KB. Data blocks are the smallest units of storage that Oracle Database can use or allocate.

An **extent** is a set of logically contiguous data blocks allocated for storing a specific type of information. In Figure 12–2, the 24 KB extent has 12 data blocks, while the 72 KB extent has 36 data blocks.

A **segment** is a set of extents allocated for a specific database object, such as a **table**. For example, the data for the `employees` table is stored in its own **data segment**, whereas each **index** for `employees` is stored in its own **index segment**. Every database object that consumes storage consists of a single segment.

Each segment belongs to one and only one **tablespace**. Thus, all extents for a segment are stored in the same tablespace. Within a tablespace, a segment can include extents from multiple data files, as shown in Figure 12–2. For example, one extent for a segment may be stored in `users01.dbf`, while another is stored in `users02.dbf`. A single extent can never span data files.

See Also: "Overview of Data Files" on page 11-7

Logical Space Management

Oracle Database must use **logical space management** to track and allocate the extents in a tablespace. When a database object requires an extent, the database must have a method of finding and providing it. Similarly, when an object no longer requires an extent, the database must have a method of making the free extent available.

Oracle Database manages space within a tablespace based on the type that you create. You can create either of the following types of tablespaces:

- Locally managed tablespaces (default)

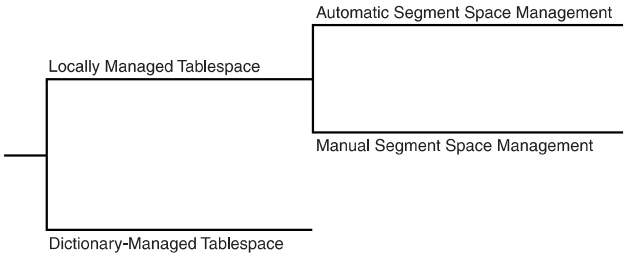
  The database uses bitmaps in the tablespaces themselves to manage extents. Thus, locally managed tablespaces have a part of the tablespace set aside for a bitmap. Within a tablespace, the database can manage segments with **automatic segment space management (ASSM)** or **manual segment space management (MSSM)**.

- Dictionary-managed tablespaces

  The database uses the **data dictionary** to manage extents (see "Overview of the Data Dictionary" on page 6-1).

Figure 12–3 shows the alternatives for logical space management in a tablespace.

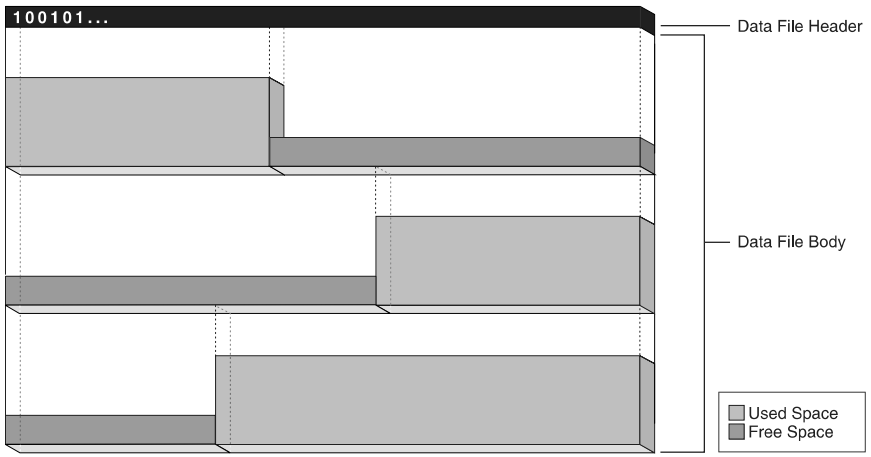
Figure 12–3 Logical Space Management



Locally Managed Tablespaces

A locally managed tablespace maintains a bitmap in the data file header to track free and used space in the data file body. Each bit corresponds to a group of blocks. When space is allocated or freed, Oracle Database changes the bitmap values to reflect the new status of the blocks.

The following graphic is a conceptual representation of bitmap-managed storage. A `1` in the header refers to used space, whereas a `0` refers to free space.



A locally managed tablespace has the following advantages:

- Avoids using the data dictionary to manage extents

  Recursive operations can occur in dictionary-managed tablespaces if consuming or releasing space in an extent results in another operation that consumes or releases space in a data dictionary table or undo segment.

- Tracks adjacent free space automatically

  In this way, the database eliminates the need to coalesce free extents.

- Determines the size of locally managed extents automatically

  Alternatively, all extents can have the same size in a locally managed tablespace and override object storage options.

> **Note:** Oracle strongly recommends the use of locally managed tablespaces with Automatic Segment Space Management.

Segment space management is an attribute inherited from the tablespace that contains the segment. Within a locally managed tablespace, the database can manage segments automatically or manually. For example, segments in tablespace `users` can be managed automatically while segments in tablespace `tools` are managed manually.

Automatic Segment Space Management The ASSM method uses bitmaps to manage space. Bitmaps provide the following advantages:

- Simplified administration

  ASSM avoids the need to manually determine correct settings for many storage parameters. Only one crucial SQL parameter controls space allocation: `PCTFREE`. This parameter specifies the percentage of space to be reserved in a block for future updates (see "Percentage of Free Space in Data Blocks" on page 12-12).

- Increased concurrency

  Multiple **transactions** can search separate lists of free data blocks, thereby reducing contention and waits. For many standard workloads, application performance with ASSM is better than the performance of a well-tuned application that uses MSSM.

- Dynamic affinity of space to instances in an Oracle Real Application Clusters (Oracle RAC) environment

ASSM is more efficient and is the default for permanent, locally managed tablespaces.

> **Note:** This chapter assumes the use of ASSM in all of its discussions of logical storage space.

Manual Segment Space Management The legacy MSSM method uses a linked list called a **free list** to manage free space in the segment. For a database object that has free space, a free list keeps track of blocks under the **high water mark** (HWM), which is the dividing line between segment space that is used and not yet used. As blocks are used, the database puts blocks on or removes blocks from the free list as needed.

In addition to `PCTFREE`, MSSM requires you to control space allocation with SQL parameters such as `PCTUSED`, `FREELISTS`, and `FREELIST GROUPS`. `PCTUSED` sets the percentage of free space that must exist in a currently used block for the database to put it on the free list. For example, if you set `PCTUSED` to `40` in a `CREATE TABLE`

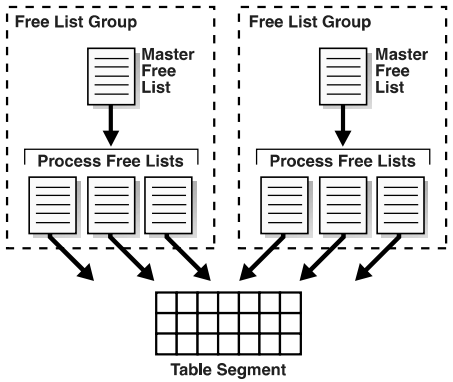
statement, then you cannot insert rows into a block in the segment until less than 40% of the block space is used.

As an illustration, suppose you insert a row into a table. The database checks a free list of the table for the first available block. If the row cannot fit in the block, and if the used space in the block is greater than or equal to `PCTUSED`, then the database takes the block off the list and searches for another block. If you delete rows from the block, then the database checks whether used space in the block is now less than `PCTUSED`. If so, then the database places the block at the beginning of the free list.

An object may have multiple free lists. In this way, multiple sessions performing DML on a table can use different lists, which can reduce contention. Each database session uses only one free list for the duration of its session.

As shown in Figure 12–4, you can also create an object with one or more **free list groups**, which are collections of free lists. Each group has a **master free list** that manages the individual **process free lists** in the group. Space overhead for free lists, especially for free list groups, can be significant.

Figure 12–4 Free List Groups



Managing segment space manually can be complex. You must adjust `PCTFREE` and `PCTUSED` to reduce row migration (see "Chained and Migrated Rows" on page 12-16) and avoid wasting space. For example, if every used block in a segment is half full, and if `PCTUSED` is 40, then the database does not permit inserts into any of these blocks. Because of the difficulty of fine-tuning space allocation parameters, Oracle strongly recommends ASSM. In ASSM, `PCTFREE` determines whether a new row can be inserted into a block, but it does not use free lists and ignores `PCTUSED`.

See Also:

- *Oracle Database Administrator's Guide* to learn about locally managed tablespaces
- *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* to learn more about automatic segment space management
- *Oracle Database SQL Language Reference* to learn about storage parameters such as `PCTFREE` and `PCTUSED`

Dictionary-Managed Tablespaces

A dictionary-managed tablespace uses the data dictionary to manage its extents. Oracle Database updates tables in the data dictionary whenever an extent is allocated or freed for reuse. For example, when a table needs an extent, the database queries the data dictionary tables, and searches for free extents. If the database finds space, then it modifies one data dictionary table and inserts a row into another. In this way, the database manages space by modifying and moving data.

The SQL that the database executes in the background to obtain space for database objects is **recursive SQL**. Frequent use of recursive SQL can have a negative impact on performance because updates to the data dictionary must be serialized. Locally managed tablespaces, which are the default, avoid this performance problem.

> **See Also:** *Oracle Database Administrator's Guide* to learn how to migrate tablespaces from dictionary-managed to locally managed

Overview of Data Blocks

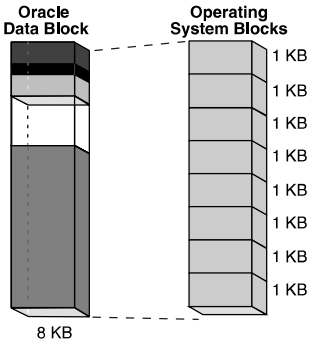
Oracle Database manages the logical storage space in the data files of a database in units called **data blocks**, also called **Oracle blocks** or **pages**. A data block is the minimum unit of database I/O.

Data Blocks and Operating System Blocks

At the physical level, database data is stored in disk files made up of operating system blocks. An **operating system block** is the minimum unit of data that the operating system can read or write. In contrast, an Oracle block is a logical storage structure whose size and structure are not known to the operating system.

Figure 12–5 shows that operating system blocks may differ in size from data blocks. The database requests data in multiples of data blocks, not operating system blocks.

Figure 12–5 Data Blocks and Operating System Blocks



When the database requests a data block, the operating system translates this operation into a requests for data in permanent storage. The logical separation of data blocks from operating system blocks has the following implications:

- Applications do not need to determine the physical addresses of data on disk.
- Database data can be striped or mirrored on multiple physical disks.

Database Block Size

Every database has a **database block size**. The DB_BLOCK_SIZE initialization parameter sets the data block size for a database when it is created. The size is set for the SYSTEM and SYSAUX tablespaces and is the default for all other tablespaces. The database block size cannot be changed except by re-creating the database.

If DB_BLOCK_SIZE is not set, then the default data block size is operating system-specific. The standard data block size for a database is 4 KB or 8 KB. If the size differs for data blocks and operating system blocks, then the data block size must be a multiple of the operating system block size.

See Also:

- *Oracle Database Reference* to learn about the DB_BLOCK_SIZE initialization parameter
- *Oracle Database Administrator's Guide* and *Oracle Database Performance Tuning Guide* to learn how to choose block sizes

Tablespace Block Size

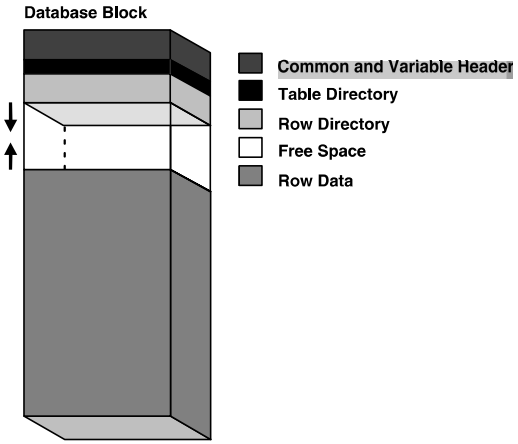
You can create individual tablespaces whose block size differs from the DB_BLOCK_SIZE setting. A nonstandard block size can be useful when moving a **transportable tablespace** to a different platform.

See Also: *Oracle Database Administrator's Guide* to learn how to specify a nonstandard block size for a tablespace

Data Block Format

Every data block has a **format** or internal structure that enables the database to track the data and free space in the block. This format is similar whether the data block contains table, index, or **table cluster** data. Figure 12–6 shows the format of an uncompressed data block (see "Data Block Compression" on page 12-11 to learn about compressed blocks).

Figure 12–6 Data Block Format



Data Block Overhead

Oracle Database uses the **block overhead** to manage the block itself. The block overhead is not available to store user data. As shown in Figure 12–6, the block overhead includes the following parts:

- Block header

  This part contains general information about the block, including disk address and segment type. For blocks that are transaction-managed, the **block header** contains active and historical transaction information.

  A **transaction entry** is required for every transaction that updates the block. Oracle Database initially reserves space in the block header for transaction entries. In data blocks allocated to segments that support transactional changes, free space can also hold transaction entries when the header space is depleted. The space required for transaction entries is operating system dependent. However, transaction entries in most operating systems require approximately 23 bytes.

- Table directory

  For a **heap-organized table**, this directory contains metadata about tables whose rows are stored in this block. Multiple tables can store rows in the same block.

- Row directory

  For a heap-organized table, this directory describes the location of rows in the data portion of the block.

  After space has been allocated in the row directory, the database does not reclaim this space after row deletion. Thus, a block that is currently empty but formerly had up to 50 rows continues to have 100 bytes allocated for the row directory. The database reuses this space only when new rows are inserted in the block.

Some parts of the block overhead are fixed in size, but the total size is variable. On average, the block overhead totals 84 to 107 bytes.

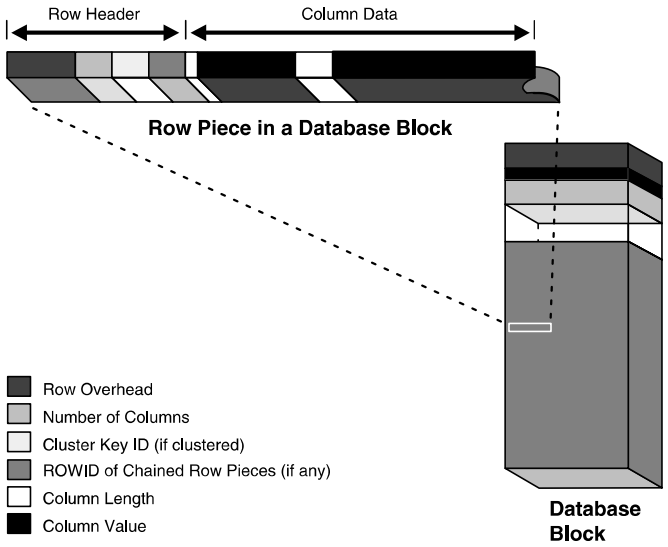
Row Format

The row data part of the block contains the actual data, such as table rows or index key entries. Just as every data block has an internal format, every row has a **row format** that enables the database to track the data in the row.

Oracle Database stores rows as variable-length records. A row is contained in one or more **row pieces**. Each row piece has a **row header** and **column data**.

Figure 12–7 shows the format of a row.

Figure 12–7 The Format of a Row Piece



Row Header Oracle Database uses the row header to manage the row piece stored in the block. The row header contains information such as the following:

- Columns in the row piece
- Pieces of the row located in other data blocks

  If an entire row can be inserted into a single data block, then Oracle Database stores the row as one row piece. However, if all of the row data cannot be inserted into a single block or an update causes an existing row to outgrow its block, then the database stores the row in multiple row pieces (see "Chained and Migrated Rows" on page 12-16). A data block usually contains only one row piece per row.
- Cluster keys for **table clusters** (see "Overview of Table Clusters" on page 2-22)

A row fully contained in one block has at least 3 bytes of row header.

Column Data After the row header, the column data section stores the actual data in the row. The row piece usually stores columns in the order listed in the `CREATE TABLE` statement, but this order is not guaranteed. For example, columns of type `LONG` are created last.

As shown in Figure 12–7, for each column in a row piece, Oracle Database stores the column length and data separately. The space required depends on the data type. If the data type of a column is variable length, then the space required to hold a value can grow and shrink with updates to the data.

Each row has a slot in the row directory of the data block header. The slot points to the beginning of the row.

> **See Also:** "Table Storage" on page 2-18 and "Index Storage" on page 3-20

- **Rowid Format** Oracle Database uses a **rowid** to uniquely identify a row. Internally, the rowid is a structure that holds information that the database needs to access a row. A rowid is not physically stored in the database, but is inferred from the file and block on which the data is stored.

  An **extended rowid** includes a data object number. This rowid type uses a base 64 encoding of the physical address for each row. The encoding characters are `A-Z`, `a-z`, `0-9`, `+`, and `/`.

  Example 12–1 queries the `ROWID` **pseudocolumn** to show the extended rowid of the row in the `employees` table for employee 100.

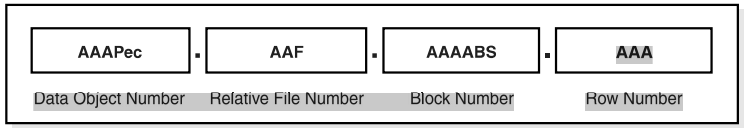
  Example 12–1 ROWID Pseudocolumn

  ```
  SQL> SELECT ROWID FROM employees WHERE employee_id = 100;

  ROWID
  ------------------
  AAAPecAAFAAAABSAAA
  ```

  Figure 12–8 illustrates the format of an extended rowid.

  Figure 12–8 ROWID Format

  An extended rowid is displayed in a four-piece format, `OOOOOOFFFBBBBBBRRR`, with the format divided into the following components:

  - `OOOOOO`

    The **data object number** identifies the segment (data object `AAAPec` in Example 12–1). A data object number is assigned to every database segment. Schema objects in the same segment, such as a **table cluster**, have the same data object number.

  - `FFF`

    The tablespace-relative **data file number** identifies the data file that contains the row (file `AAF` in Example 12–1).

  - `BBBBBB`

    The **data block number** identifies the block that contains the row (block `AAAABS` in Example 12–1). Block numbers are relative to their data file, not their tablespace. Thus, two rows with identical block numbers could reside in different data files of the same tablespace.

  - `RRR`

    The **row number** identifies the row in the block (row `AAA` in Example 12–1).

  After a rowid is assigned to a row piece, the rowid can change in special circumstances. For example, if **row movement** is enabled, then the rowid can change because of partition key updates, Flashback Table operations, shrink table operations, and so on. If row movement is disabled, then a rowid can change if the row is exported and imported using Oracle Database utilities.

Note: Internally, the database performs row movement as if the row were physically deleted and reinserted. However, row movement is considered an update, which has implications for triggers.

See Also:

- "Rowid Data Types" on page 2-13
- *Oracle Database SQL Language Reference* to learn about rowids

Data Block Compression

The database can use **table compression** to eliminate duplicate values in a data block (see "Table Compression" on page 2-19). This section describes the format of data blocks that use compression.

The format of a data block that uses basic and advanced row compression is essentially the same as an uncompressed block. The difference is that a **symbol table** at the beginning of the block stores duplicate values for the rows and columns. The database replaces occurrences of these values with a short reference to the symbol table.

Assume that the rows in Example 12–2 are stored in a data block for the seven-column `sales` table.

Example 12–2 Rows in sales Table

```
2190,13770,25-NOV-00,S,9999,23,161
2225,15720,28-NOV-00,S,9999,25,1450
34005,120760,29-NOV-00,P,9999,44,2376
9425,4750,29-NOV-00,I,9999,11,979
1675,46750,29-NOV-00,S,9999,19,1121
```

When basic or advanced row compression is applied to this table, the database replaces duplicate values with a symbol reference. Example 12–3 is a conceptual representation of the compression in which the symbol `*` replaces `29-NOV-00` and `%` replaces `9999`.

Example 12–3 OLTP Compressed Rows in sales Table

```
2190,13770,25-NOV-00,S,%,23,161
2225,15720,28-NOV-00,S,%,25,1450
34005,120760,*,P,%,44,2376
9425,4750,*,I,%,11,979
1675,46750,*,S,%,19,1121
```

Table 12–1 conceptually represents the symbol table that maps symbols to values.

Table 12–1 Symbol Table

Symbol	Value	Column	Rows
*	29-NOV-00	3	958-960
%	9999	5	956-960

Space Management in Data Blocks

As the database fills a data block from the bottom up, the amount of **free space** between the row data and the block header decreases. This free space can also shrink during updates, as when changing a trailing null to a nonnull value. The database manages free space in the data block to optimize performance and avoid wasted space.

> **Note:** This section assumes the use of automatic segment space management.

Percentage of Free Space in Data Blocks

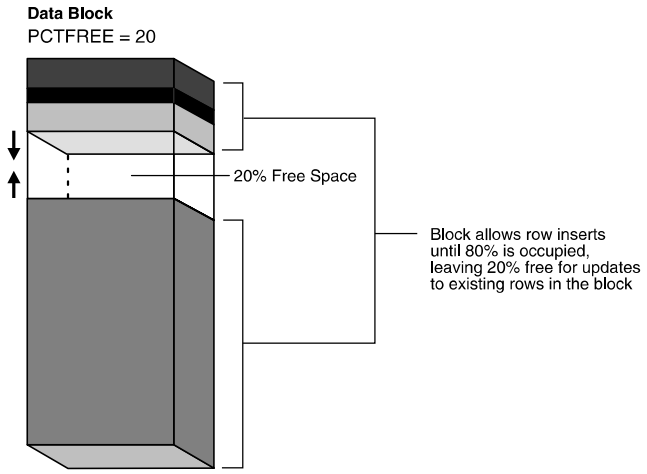
The `PCTFREE` storage parameter is essential to how the database manages free space. This SQL parameter sets the minimum percentage of a data block reserved as free space for updates to existing rows. Thus, `PCTFREE` is important for preventing row migration and avoiding wasted space.

For example, assume that you create a table that will require only occasional updates, most of which will not increase the size of the existing data. You specify the `PCTFREE` parameter within a `CREATE TABLE` statement as follows:

```
CREATE TABLE test_table (n NUMBER) PCTFREE 20;
```

Figure 12–9 shows how a `PCTFREE` setting of `20` affects space management. The database adds rows to the block over time, causing the row data to grow upwards toward the block header, which is itself expanding downward toward the row data. The `PCTFREE` setting ensures that *at least* 20% of the data block is free. For example, the database prevents an `INSERT` statement from filling the block so that the row data and header occupy a combined 90% of the total block space, leaving only 10% free.

Figure 12–9 PCTFREE



> **Note:** This discussion does not apply to **LOB** data types, which do not use the `PCTFREE` storage parameter or free lists. See "Overview of LOBs" on page 19-12.

See Also: *Oracle Database SQL Language Reference* for the syntax and semantics of the PCTFREE parameter

Optimization of Free Space in Data Blocks

While the percentage of free space cannot be *less* than PCTFREE, the amount of free space can be *greater*. For example, a PCTFREE setting of 20% prevents the total amount of free space from dropping to 5% of the block, but permits 50% of the block to be free space. The following SQL statements can increase free space:

- DELETE statements
- UPDATE statements that either update existing values to smaller values or increase existing values and force a row to migrate
- INSERT statements on a table that uses OLTP compression

  If inserts fill a block with data, then the database invokes block compression, which may result in the block having more free space.

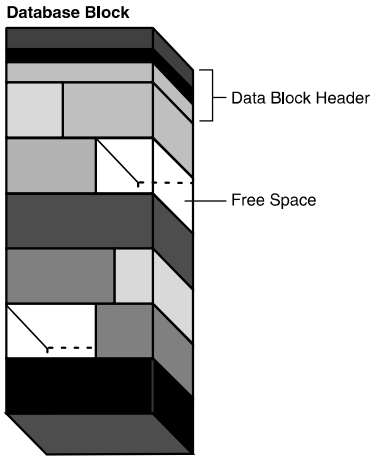
The space released is available for INSERT statements under the following conditions:

- If the INSERT statement is in the same transaction and after the statement that frees space, then the statement can use the space.
- If the INSERT statement is in a separate transaction from the statement that frees space (perhaps run by another user), then the statement can use the space made available only after the other transaction commits and only if the space is needed.

See Also: *Oracle Database Administrator's Guide* to learn about OLTP compression

Coalescing Fragmented Space Released space may or may not be contiguous with the main area of free space in a data block, as shown in Figure 12–10. Noncontiguous free space is called **fragmented space**.

Figure 12–10 Data Block with Fragmented Space

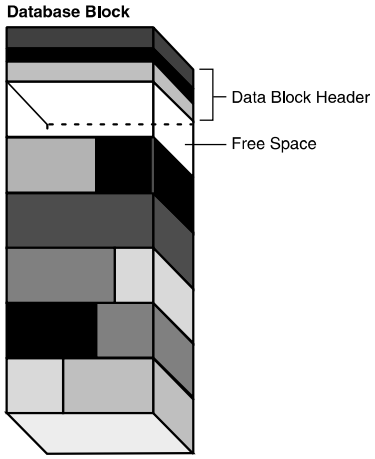


Oracle Database automatically and transparently coalesces the free space of a data block *only* when the following conditions are true:

- An `INSERT` or `UPDATE` statement attempts to use a block that contains sufficient free space to contain a new row piece.
- The free space is fragmented so that the row piece cannot be inserted in a contiguous section of the block.

After coalescing, the amount of free space is identical to the amount before the operation, but the space is now contiguous. Figure 12–11 shows a data block after space has been coalesced.

Figure 12–11 Data Block After Coalescing Free Space



Oracle Database performs coalescing only in the preceding situations because otherwise performance would decrease because of the continuous coalescing of the free space in data blocks.

Reuse of Index Space The database can reuse space within an index block. For example, if you insert a value into a column and delete it, and if an index exists on this column, then the database can reuse the index slot when a row requires it.

The database can reuse an index block itself. Unlike a table block, an index block only becomes free when it is empty. The database places the empty block on the free list of the index structure and makes it eligible for reuse. However, Oracle Database does not automatically compact the index: an `ALTER INDEX REBUILD` or `COALESCE` statement is required.

Figure 12–12 represents an index of the `employees.department_id` column before the index is coalesced. The first three leaf blocks are only partially full, as indicated by the gray fill lines.

Figure 12–12 Index Before Coalescing

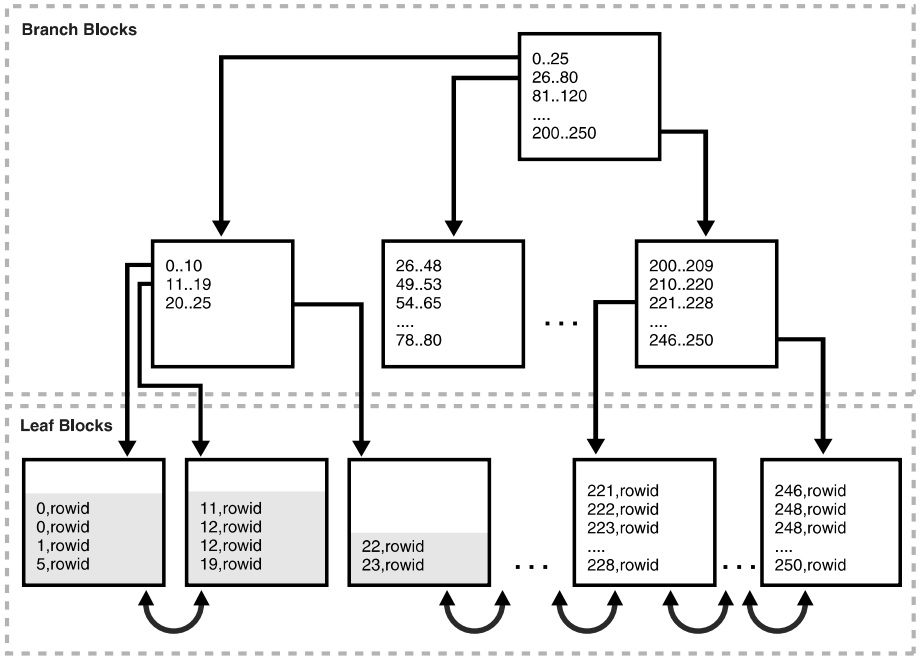
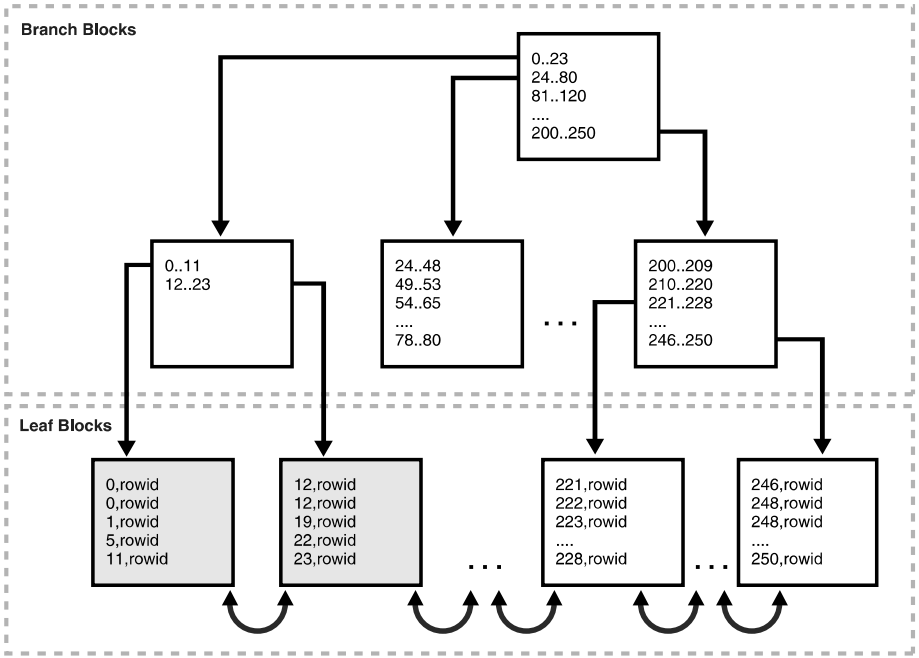


Figure 12–13 shows the index in Figure 12–12 after the index has been coalesced. The first two leaf blocks are now full, as indicated by the gray fill lines, and the third leaf block has been freed.

Figure 12–13 Index After Coalescing



See Also:

- *Oracle Database Administrator's Guide* to learn how to coalesce and rebuild indexes
- *Oracle Database SQL Language Reference* to learn about the `COALESCE` statement

Chained and Migrated Rows

Oracle Database must manage rows that are too large to fit into a single block. The following situations are possible:

- The row is too large to fit into one data block when it is first inserted.

  In **row chaining**, Oracle Database stores the data for the row in a **chain** of one or more data blocks reserved for the segment. Row chaining most often occurs with large rows. Examples include rows that contain a column of data type `LONG` or `LONG RAW`, a `VARCHAR2(4000)` column in a 2 KB block, or a row with a huge number of columns. Row chaining in these cases is unavoidable.

- A row that originally fit into one data block is updated so that the overall row length increases, but insufficient free space exists to hold the updated row.

  In **row migration**, Oracle Database moves the entire row to a new data block, assuming the row can fit in a new block. The original row piece of a migrated row contains a pointer or "forwarding address" to the new block containing the migrated row. The rowid of a migrated row does not change.

- A row has more than 255 columns.

Oracle Database can only store 255 columns in a row piece. Thus, if you insert a row into a table that has 1000 columns, then the database creates 4 row pieces, typically chained over multiple blocks.

Figure 12–14 depicts shows the insertion of a large row in a data block. The row is too large for the left block, so the database chains the row by placing the first row piece in the left block and the second row piece in the right block.

Figure 12–14 Row Chaining

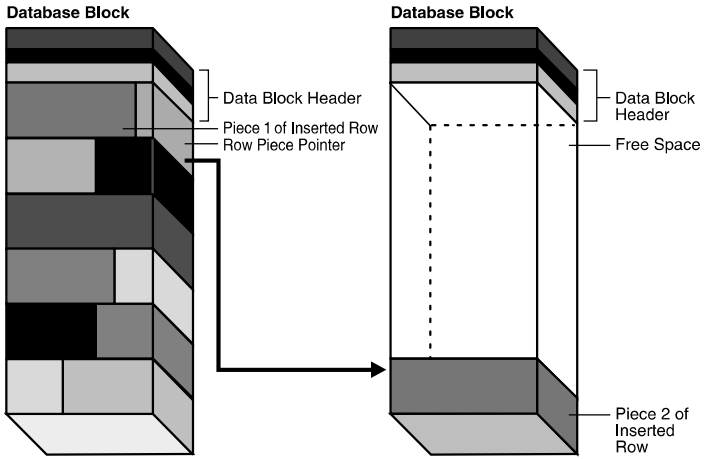
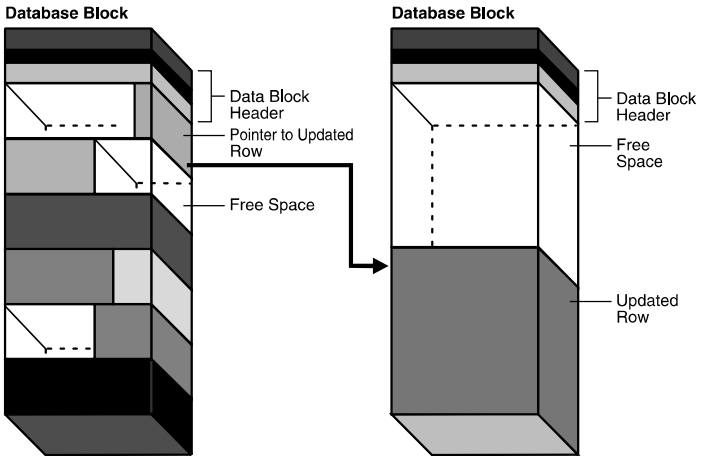


Figure 12–15, the left block contains a row that is updated so that the row is now too large for the block. The database moves the entire row to the right block and leaves a pointer to the migrated row in the left block.

Figure 12–15 Row Migration



When a row is chained or migrated, the I/O needed to retrieve the data increases. This situation results because Oracle Database must scan multiple blocks to retrieve the information for the row. For example, if the database performs one I/O to read an index and one I/O to read a nonmigrated table row, then an additional I/O is required to obtain the data for a migrated row.

The Segment Advisor, which can be run both manually and automatically, is an Oracle Database component that identifies segments that have space available for reclamation. The advisor can offer advice about objects that have significant free space or too many chained rows.

See Also:

- "Row Storage" on page 2-19 and "Rowids of Row Pieces" on page 2-19
- *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* to learn how to reclaim wasted space
- *Oracle Database Performance Tuning Guide* to learn about reducing chained and migrated rows

Overview of Extents

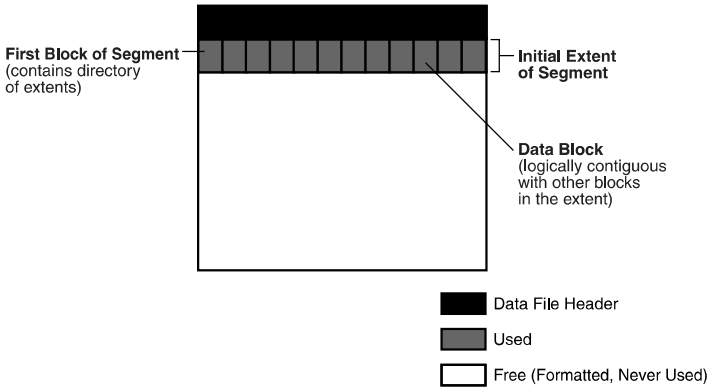
An **extent** is a logical unit of database storage space allocation made up of contiguous data blocks. Data blocks in an extent are logically contiguous but can be physically spread out on disk because of RAID striping and file system implementations.

Allocation of Extents

By default, the database allocates an **initial extent** for a data segment when the segment is created. An extent is always contained in one data file.

Although no data has been added to the segment, the data blocks in the initial extent are reserved for this segment exclusively. The first data block of every segment contains a directory of the extents in the segment. Figure 12–16 shows the initial extent in a segment in a data file that previously contained no data.

Figure 12–16 Initial Extent of a Segment

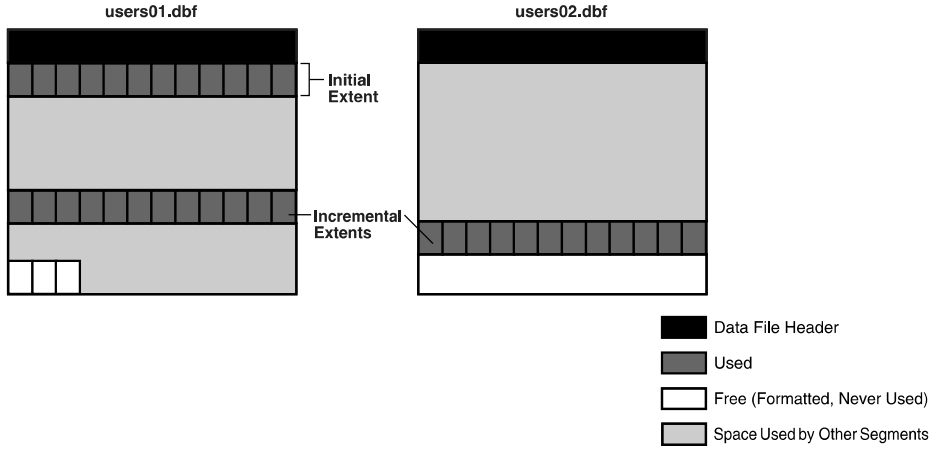


If the initial extent become full, and if more space is required, then the database automatically allocates an **incremental extent** for this segment. An incremental extent is a subsequent extent created for the segment.

The allocation algorithm depends on whether the tablespace is locally managed or dictionary-managed. In the locally managed case, the database searches the bitmap of a data file for adjacent free blocks. If the data file has insufficient space, then the database looks in another data file. Extents for a segment are always in the same tablespace but may be in different data files.

Figure 12–17 shows that the database can allocate extents for a segment in any data file in the tablespace. For example, the segment can allocate the initial extent in `users01.dbf`, allocate the first incremental extent in `users02.dbf`, and allocate the next extent in `users01.dbf`.

Figure 12–17 Incremental Extent of a Segment



The blocks of a newly allocated extent, although they were free, may not be empty of old data. In ASSM, Oracle Database formats the blocks of a newly allocated extent when it starts using the extent, but only as needed (see "Segment Space and the High Water Mark" on page 12-27).

Note: This section applies to serial operations, in which one **server process** parses and runs a statement. Extents are allocated differently in parallel SQL statements, which entail multiple server processes.

See Also: *Oracle Database Administrator's Guide* to learn how to manually allocate extents

Deallocation of Extents

In general, the extents of a user segment do not return to the tablespace unless you drop the object using a `DROP` command. In Oracle Database 11*g* Release 2 (11.2.0.2), you can also drop the segment using the `DBMS_SPACE_ADMIN` package. For example, if you delete all rows in a table, then the database does not reclaim the data blocks for use by other objects in the tablespace.

Note: In an undo segment, Oracle Database periodically deallocates one or more extents if it has the `OPTIMAL` size specified or if the database is in **automatic undo management mode** (see "Undo Tablespaces" on page 12-33).

In some circumstances, you can manually deallocate space. The Oracle Segment Advisor helps determine whether an object has space available for reclamation based on the level of fragmentation in the object. The following techniques can free extents:

- You can use an **online segment shrink** to reclaim fragmented space in a segment. Segment shrink is an online, in-place operation. In general, data compaction leads to better cache utilization and requires fewer blocks to be read in a **full table scan**.
- You can move the data of a nonpartitioned table or table partition into a new segment, and optionally into a different tablespace for which you have quota.
- You can rebuild or coalesce the index (see "Reuse of Index Space" on page 12-14).
- You can **truncate** a table or table cluster, which removes all rows. By default, Oracle Database deallocates all space used by the removed rows except that specified by the `MINEXTENTS` storage parameter. In Oracle Database 11*g* Release 2 (11.2.0.2), you can also use `TRUNCATE` with the `DROP ALL STORAGE` option to drop entire segments.
- You can deallocate unused space, which frees the unused space at the high water mark end of the database segment and makes the space available for other segments in the tablespace (see "Segment Space and the High Water Mark" on page 12-27).

When extents are freed, Oracle Database modifies the bitmap in the data file for locally managed tablespaces to reflect the regained extents as available space. Any data in the blocks of freed extents becomes inaccessible.

See Also: *Oracle Database Administrator's Guide* to learn how to reclaim segment space

Storage Parameters for Extents

Every segment is defined by **storage parameters** expressed in terms of extents. These parameters control how Oracle Database allocates free space for a segment.

The storage settings are determined in the following order of precedence, with setting higher on the list overriding settings lower on the list:

1. Segment storage clause
2. Tablespace storage clause
3. Oracle Database default

A locally managed tablespace can have either uniform extent sizes or variable extent sizes determined automatically by the system:

- For **uniform extents**, you can specify an extent size or use the default size of 1 MB. All extents in the tablespace are of this size. Locally managed temporary tablespaces can only use this type of allocation.
- For **automatically allocated extents**, Oracle Database determines the optimal size of additional extents.

For locally managed tablespaces, some storage parameters cannot be specified at the tablespace level. However, you can specify these parameters at the segment level. In this case, the databases uses all parameters together to compute the initial size of the segment. Internal algorithms determine the subsequent size of each extent.

See Also:

- *Oracle Database Administrator's Guide* to learn about extent management considerations when creating a locally managed tablespace
- *Oracle Database SQL Language Reference* to learn about options in the storage clause

Overview of Segments

A segment is a set of extents that contains all the data for a logical storage structure within a tablespace. For example, Oracle Database allocates one or more extents to form the data segment for a table. The database also allocates one or more extents to form the index segment for a table.

As explained in "Logical Space Management", Oracle Database manages segment space automatically or manually. This section assumes the use of ASSM.

User Segments

A single data segment in a database stores the data for one user object. There are different types of segments. Examples of **user segments** include:

- Table, table partition, or table cluster
- LOB or LOB partition
- Index or index partition

Each nonpartitioned object and object partition is stored in its own segment. For example, if an index has five partitions, then five segments contain the index data.

User Segment Creation

By default, the database uses **deferred segment creation** to update only database metadata when creating tables and indexes. Starting in Oracle Database 11*g* Release 2 (11.2.0.2), the database also defers segment creation when creating partitions. When a user inserts the first row into a table or partition, the database creates segments for the table or partition, its LOB columns, and its indexes.

Deferred segment creation enables you to avoid using database resources unnecessarily. For example, installation of an application can create thousands of objects, consuming significant disk space. Many of these objects may never be used.

You can use the `DBMS_SPACE_ADMIN` package to manage segments for empty objects. Starting with Oracle Database 11*g* Release 2 (11.2.0.2), you can use this PL/SQL package to do the following:

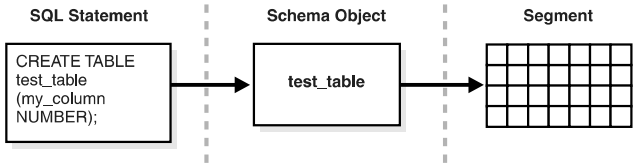
- Manually materialize segments for empty tables or partitions that do not have segments created
- Remove segments from empty tables or partitions that currently have an empty segment allocated

To best illustrate the relationship between object creation and segment creation, assume that deferred segment creation is disabled. You create a table as follows:

```
CREATE TABLE test_table (my_column NUMBER);
```

As shown in Figure 12–18, the database creates one segment for the table.

Figure 12–18 Creation of a User Segment

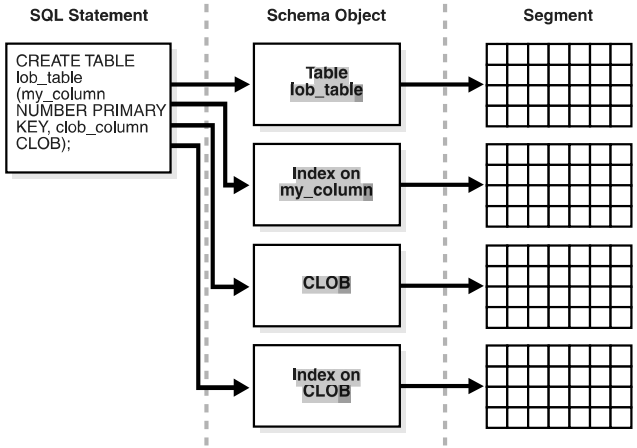


- When you create a table with a **primary key** or unique key, Oracle Database automatically creates an index for this key. Again assume that deferred segment creation is disabled. You create a table as follows:

```
CREATE TABLE lob_table (my_column NUMBER PRIMARY KEY, clob_column CLOB);
```

Figure 12–19 shows that the data for `lob_table` is stored in one segment, while the implicitly created index is in a different segment. Also, the CLOB data is stored in its own segment, as is its associated CLOB index (see "Internal LOBs" on page 19-12). Thus, the `CREATE TABLE` statement results in the creation of *four* different segments.

Figure 12–19 Multiple Segments



Note: The segments of a table and the index for this table do not have to occupy the same tablespace.

The database allocates one or more extents when a segment is created. Storage parameters for the object determine how the extents for each segment are allocated (see "Storage Parameters for Extents" on page 12-20). The parameters affect the efficiency of data retrieval and storage for the data segment associated with the object.

See Also:

- *Oracle Database Administrator's Guide* to learn how to manage deferred segment creation
- *Oracle Database Advanced Replication* for information on materialized views and materialized view logs
- *Oracle Database SQL Language Reference* for `CREATE TABLE` syntax

Temporary Segments

When processing a **query**, Oracle Database often requires temporary workspace for intermediate stages of SQL statement execution. Typical operations that may require a **temporary segment** include sorting, **hashing**, and merging bitmaps. While creating an index, Oracle Database also places index segments into temporary segments and then converts them into permanent segments when the index is complete.

Oracle Database does not create a temporary segment if an operation can be performed in memory. However, if memory use is not possible, then the database automatically allocates a temporary segment on disk.

Allocation of Temporary Segments for Queries

Oracle Database allocates temporary segments for queries as needed during a user session and drops them when the query completes. Changes to temporary segments are not recorded in the **online redo log**, except for space management operations on the temporary segment (see "Overview of the Online Redo Log" on page 11-12).

The database creates temporary segments in the temporary tablespace assigned to the user. The default storage characteristics of the tablespace determine the characteristics of the extents in the temporary segment. Because allocation and deallocation of temporary segments occurs frequently, the best practice is to create at least one special tablespace for temporary segments. The database distributes I/O across disks and avoids fragmenting `SYSTEM` and other tablespaces with temporary segments.

Note: When `SYSTEM` is locally managed, you must define a default temporary tablespace at database creation. A locally managed `SYSTEM` tablespace cannot be used for default temporary storage.

See Also:

- *Oracle Database Administrator's Guide* to learn how to create temporary tablespaces
- *Oracle Database SQL Language Reference* for `CREATE TEMPORARY TABLESPACE` syntax and semantics

Allocation of Temporary Segments for Temporary Tables and Indexes

Oracle Database can also allocate temporary segments for **temporary tables** and their indexes. Temporary tables hold data that exists only for the duration of a transaction or session. Each session accesses only the extents allocated for the session and cannot access extents allocated for other sessions.

Oracle Database allocates segments for a temporary table when the first `INSERT` into that table occurs. The insertion can occur explicitly or because of `CREATE TABLE AS SELECT`. The first `INSERT` into a temporary table allocates the segments for the table and its indexes, creates the root page for the indexes, and allocates any `LOB` segments.

Segments for a temporary table are allocated in a temporary tablespace of the current user. Assume that the temporary tablespace assigned to `user1` is `temp1` and the temporary tablespace assigned to `user2` is `temp2`. In this case, `user1` stores temporary data in the `temp1` segments, while `user2` stores temporary data in the `temp2` segments.

See Also:

- "Temporary Tables" on page 2-15
- *Oracle Database Administrator's Guide* to learn how to create temporary tables

Undo Segments

Oracle Database maintains records of the actions of transactions, collectively known as **undo data**. Oracle Database uses undo to do the following:

- Roll back an **active transaction**
- Recover a terminated transaction
- Provide **read consistency**
- Perform some logical flashback operations

Oracle Database stores undo data inside the database rather than in external logs. Undo data is stored in blocks that are updated just like data blocks, with changes to these blocks generating redo. In this way, Oracle Database can efficiently access undo data without needing to read external logs.

Undo data is stored in an **undo tablespace**. Oracle Database provides a fully automated mechanism, known as **automatic undo management mode**, for managing undo segments and space in an undo tablespace.

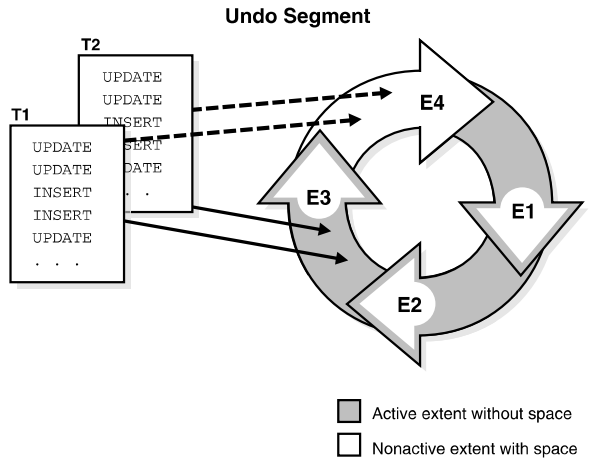
Undo Segments and Transactions

When a transaction starts, the database binds (assigns) the transaction to an undo segment, and therefore to a **transaction table**, in the current undo tablespace. In rare circumstances, if the database instance does not have a designated undo tablespace, then the transaction binds to the system undo segment.

Multiple active transactions can write concurrently to the same undo segment or to different segments. For example, transactions T1 and T2 can both write to undo segment U1, or T1 can write to U1 while T2 writes to undo segment U2.

Conceptually, the extents in an undo segment form a ring. Transactions write to one undo extent, and then to the next extent in the ring, and so on in cyclical fashion. Figure 12–20 shows two transactions, T1 and T2, which begin writing in the third extent (E3) of an undo segment and continue writing to the fourth extent (E4).

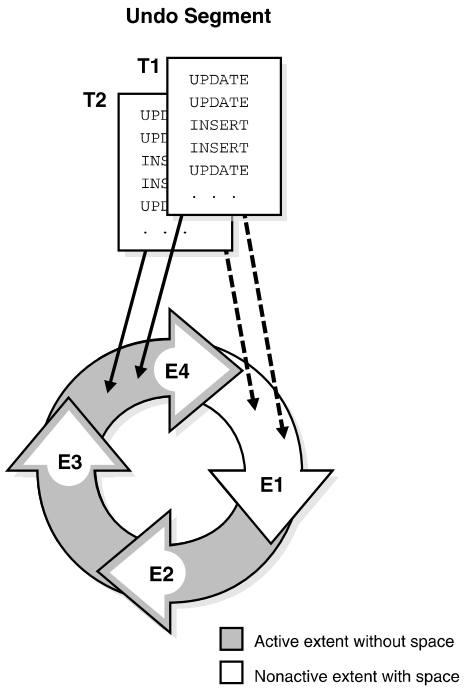
Figure 12–20 Ring of Allocated Extents in an Undo Segment



At any given time, a transaction writes sequentially to only one extent in an undo segment, known as the **current extent** for the transaction. Multiple active transactions can write simultaneously to the same current extent or to different current extents. Figure 12–20 shows transactions T1 and T2 writing simultaneously to extent E3. Within an undo extent, a data block contains data for only one transaction.

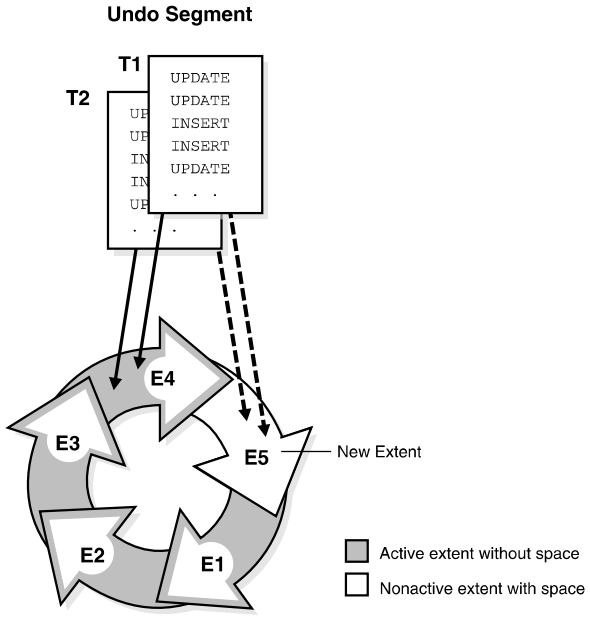
As the current undo extent fills, the first transaction needing space checks the availability of the next allocated extent in the ring. If the next extent does *not* contain data from an active transaction, then this extent becomes the current extent. Now all transactions that need space can write to the new current extent. In Figure 12–21, when E4 is full, T1 and T2 continue writing to E1, overwriting the nonactive undo data in E1.

Figure 12–21 Cyclical Use of Allocated Extents in an Undo Segment



If the next extent *does* contain data from an active transaction, then the database must allocate a new extent. Figure 12–22 shows a scenario in which T1 and T2 are writing to E4. When E4 fills up, the transactions cannot continue writing to E1 because E1 contains active undo entries. Therefore, the database allocates a new extent (E5) for this undo segment. The transactions continue writing to E5.

Figure 12–22 Allocation of a New Extent for an Undo Segment



See Also: *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* to learn how to manage undo segments

Transaction Rollback

When a `ROLLBACK` statement is issued, the database uses undo records to roll back changes made to the database by the uncommitted transaction. During recovery, the database rolls back any uncommitted changes applied from the online redo log to the data files. Undo records provide **read consistency** by maintaining the before image of the data for users accessing data at the same time that another user is changing it.

Segment Space and the High Water Mark

To manage space, Oracle Database tracks the state of blocks in the segment. The **high water mark (HWM)** is the point in a segment beyond which data blocks are unformatted and have never been used.

MSSM uses free lists to manage segment space. At table creation, no blocks in the segment are formatted. When a session first inserts rows into the table, the database searches the free list for usable blocks. If the database finds no usable blocks, then it preformats a group of blocks, places them on the free list, and begins inserting data into the blocks. In MSSM, a full table scan reads *all* blocks below the HWM.

ASSM does not use free lists and so must manage space differently. When a session first inserts data into a table, the database formats a single bitmap block instead of preformatting a group of blocks as in MSSM. The bitmap tracks the state of blocks in the segment, taking the place of the free list. The database uses the bitmap to find free blocks and then formats each block before filling it with data. ASSM spread out inserts among blocks to avoid concurrency issues.

Every data block in an ASSM segment is in one of the following states:

- Above the HWM

  These blocks are unformatted and have never been used.

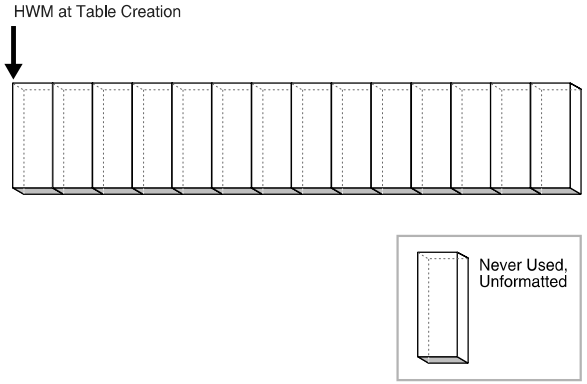
- Below the HWM

  These blocks are in one of the following states:

  - Allocated, but currently unformatted and unused
  - Formatted and contain data
  - Formatted and empty because the data was deleted

Figure 12–23 depicts an ASSM segment as a horizontal series of blocks. At table creation, the HWM is at the beginning of the segment on the left. Because no data has been inserted yet, all blocks in the segment are unformatted and never used.

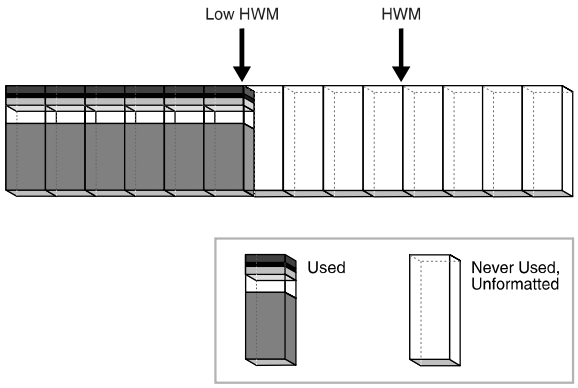
Figure 12–23 HWM at Table Creation



Suppose that a transaction inserts rows into the segment. The database must allocate a group of blocks to hold the rows. The allocated blocks fall below the HWM. The database formats a bitmap block in this group to hold the metadata, but does not preformat the remaining blocks in the group.

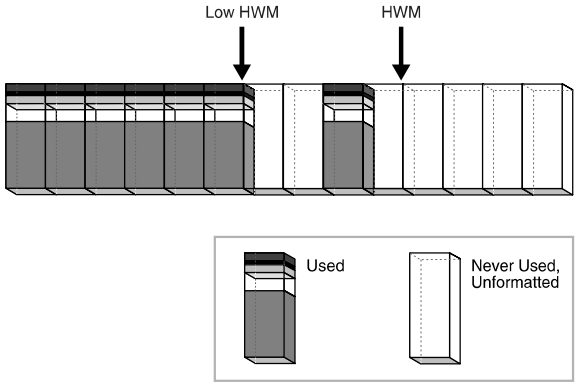
In Figure 12–24, the blocks below the HWM are allocated, whereas blocks above the HWM are neither allocated or formatted. As inserts occur, the database can write to any block with available space. The **low high water mark (low HWM)** marks the point below which all blocks are known to be formatted because they either contain data or formerly contained data.

Figure 12–24 HWM and Low HWM



In Figure 12–25, the database chooses a block between the HWM and low HWM and writes to it. The database could have just as easily chosen any other block between the HWM and low HWM, or any block below the low HWM that had available space. In Figure 12–25, the blocks to either side of the newly filled block are unformatted.

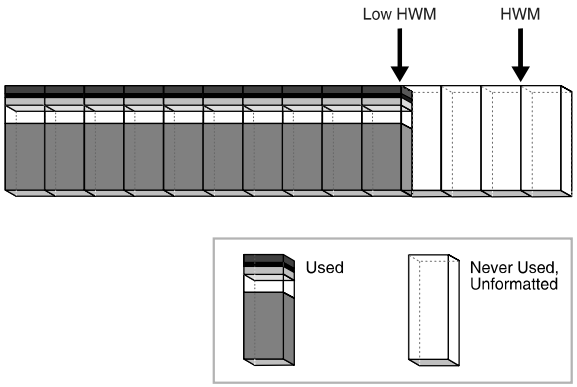
Figure 12–25 HWM and Low HWM



The low HWM is important in a **full table scan**. Because blocks below the HWM are formatted only when used, some blocks could be unformatted, as in Figure 12–25. For this reason, the database reads the bitmap block to obtain the location of the low HWM. The database reads all blocks up to the low HWM because they are known to be formatted, and then carefully reads only the formatted blocks between the low HWM and the HWM.

Assume that a new transaction inserts rows into the table, but the bitmap indicates that insufficient free space exists under the HWM. In Figure 12–26, the database advances the HWM to the right, allocating a new group of unformatted blocks.

Figure 12–26 Advancing HWM and Low HWM



When the blocks between the HWM and low HWM are full, the HWM advances to the right and the low HWM advances to the location of the old HWM. As the database inserts data over time, the HWM continues to advance to the right, with the low HWM always trailing behind it. Unless you manually rebuild, truncate, or shrink the object, the HWM never retreats.

See Also:

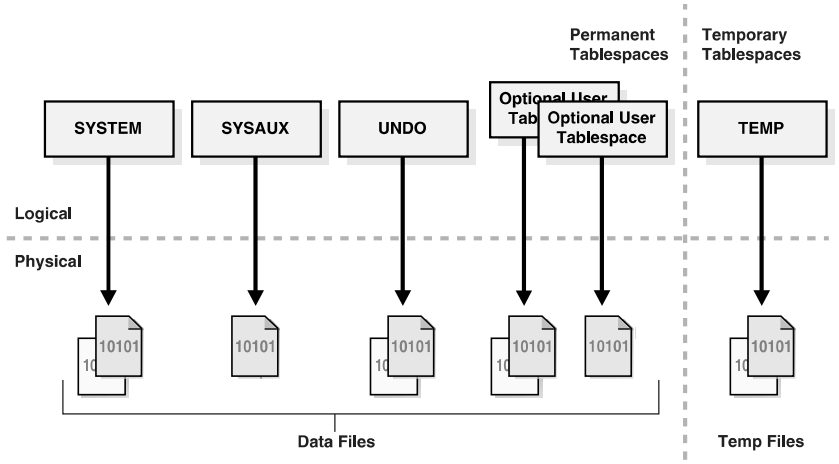
- *Oracle Database Administrator's Guide* to learn how to shrink segments online
- *Oracle Database SQL Language Reference* for `TRUNCATE TABLE` syntax and semantics

Overview of Tablespaces

A **tablespace** is a logical storage container for segments. Segments are database objects, such as tables and indexes, that consume storage space. At the physical level, a tablespace stores data in one or more data files or temp files.

A database must have the `SYSTEM` and `SYSAUX` tablespaces. Figure 12–27 shows the tablespaces in a typical database. The following sections describe the tablespace types.

Figure 12–27 Tablespaces



Permanent Tablespaces

A **permanent tablespace** groups persistent schema objects. The segments for objects in the tablespace are stored physically in data files.

Each database user is assigned a default permanent tablespace. A very small database may need only the default `SYSTEM` and `SYSAUX` tablespaces. However, Oracle recommends that you create at least one tablespace to store user and application data. You can use tablespaces to achieve the following goals:

- Control disk space allocation for database data
- Assign a **quota** (space allowance or limit) to a database user
- Take individual tablespaces online or offline without affecting the availability of the whole database
- Perform backup and recovery of individual tablespaces
- Import or export application data by using the Oracle Data Pump utility (see "Oracle Data Pump Export and Import" on page 18-7)
- Create a **transportable tablespace** that you can copy or move from one database to another, even across platforms

  Moving data by transporting tablespaces can be orders of magnitude faster than either export/import or unload/load of the same data, because transporting a tablespace involves only copying data files and integrating the tablespace metadata. When you transport tablespaces you can also move index data.

  See Also:

  - *Oracle Database Administrator's Guide* to learn how to transport tablespaces
  - *Oracle Database Utilities* to learn about Oracle Data Pump
  - *Oracle Streams Concepts and Administration* for more information on ways to copy or transport files

The SYSTEM Tablespace

The SYSTEM tablespace is a necessary administrative tablespace included with the database when it is created. Oracle Database uses SYSTEM to manage the database.

The SYSTEM tablespace includes the following information, all owned by the SYS user:

- The data dictionary
- Tables and views that contain administrative information about the database
- Compiled stored objects such as triggers, procedures, and packages

The SYSTEM tablespace is managed as any other tablespace, but requires a higher level of privilege and is restricted in some ways. For example, you cannot rename or drop the SYSTEM tablespace.

By default, Oracle Database sets all newly created user tablespaces to be locally managed. In a database with a locally managed SYSTEM tablespace, you cannot create dictionary-managed tablespaces (which are deprecated). However, if you execute the CREATE DATABASE statement manually and accept the defaults, then the SYSTEM tablespace is dictionary managed. You can migrate an existing dictionary-managed SYSTEM tablespace to a locally managed format.

Note: Oracle strongly recommends that you use Database Configuration Assistant (DBCA) to create new databases so that all tablespaces, including SYSTEM, are locally managed by default.

See Also:

- "Online and Offline Tablespaces" on page 12-35 for information about the permanent online condition of the SYSTEM tablespace
- "Tools for Database Installation and Configuration" on page 18-4 to learn about DBCA
- *Oracle Database Administrator's Guide* to learn how to create or migrate to a locally managed SYSTEM tablespace
- *Oracle Database SQL Language Reference* for CREATE DATABASE syntax and semantics

The SYSAUX Tablespace

The SYSAUX tablespace is an auxiliary tablespace to the SYSTEM tablespace. The SYSAUX tablespace provides a centralized location for database metadata that does not reside in the SYSTEM tablespace. It reduces the number of tablespaces created by default, both in the seed database and in user-defined databases.

Several database components, including Oracle Enterprise Manager and Oracle Streams, use the SYSAUX tablespace as their default storage location. Therefore, the SYSAUX tablespace is created automatically during database creation or upgrade.

During normal database operation, the database does not allow the SYSAUX tablespace to be dropped or renamed. If the SYSAUX tablespace becomes unavailable, then core database functionality remains operational. The database features that use the SYSAUX tablespace could fail, or function with limited capability.

See Also: *Oracle Database Administrator's Guide* to learn about the SYSAUX tablespace

Undo Tablespaces

An **undo tablespace** is a locally managed tablespace reserved for system-managed undo data (see "Undo Segments" on page 12-24). Like other permanent tablespaces, undo tablespaces contain data files. Undo blocks in these files are grouped in extents.

Automatic Undo Management Mode Undo tablespaces require the database to be in the default **automatic undo management mode**. This mode eliminates the complexities of manually administering undo segments. The database automatically tunes itself to provide the best possible retention of undo data to satisfy long-running queries that may require this data.

An undo tablespace is automatically created with a new installation of Oracle Database. Earlier versions of Oracle Database may not include an undo tablespace and use legacy rollback segments instead, known as **manual undo management mode**. When upgrading to Oracle Database 11*g*, you can enable automatic undo management mode and create an undo tablespace. Oracle Database contains an Undo Advisor that provides advice on and helps automate your undo environment.

A database can contain multiple undo tablespaces, but only one can be in use at a time. When an instance attempts to open a database, Oracle Database automatically selects the first available undo tablespace. If no undo tablespace is available, then the instance starts without an undo tablespace and stores undo data in the `SYSTEM` tablespace. Storing undo data in `SYSTEM` is not recommended.

See Also:

- *Oracle Database Administrator's Guide* to learn about automatic undo management
- *Oracle Database Upgrade Guide* to learn how to migrate to automatic undo management mode
- *Oracle Database 2 Day DBA* for information on the Undo Advisor and on how to use advisors

Automatic Undo Retention The **undo retention period** is the minimum amount of time that Oracle Database attempts to retain old undo data before overwriting it. Undo retention is important because long-running queries may require older block images to supply **read consistency**. Also, some Oracle Flashback features can depend on undo availability.

In general, it is desirable to retain old undo data as long as possible. After a transaction commits, undo data is no longer needed for rollback or transaction recovery. The database can retain old undo data if the undo tablespace has space for new transactions. When available space is low, the database begins to overwrite old undo data for committed transactions.

Oracle Database automatically provides the best possible undo retention for the current undo tablespace. The database collects usage statistics and tunes the retention period based on these statistics and the undo tablespace size. If the undo tablespace is configured with the `AUTOEXTEND` option, and if the maximum size is not specified, then undo retention tuning is different. In this case, the database tunes the undo retention period to be slightly longer than the longest-running query, if space allows.

See Also: *Oracle Database Administrator's Guide* for more details on automatic tuning of undo retention

Temporary Tablespaces

A **temporary tablespace** contains transient data that persists only for the duration of a session. No permanent schema objects can reside in a temporary tablespace. The database stores temporary tablespace data in **temp files**.

Temporary tablespaces can improve the concurrency of multiple sort operations that do not fit in memory. These tablespaces also improve the efficiency of space management operations during sorts.

When the `SYSTEM` tablespace is locally managed, a default temporary tablespace is included in the database by default during database creation. A locally managed `SYSTEM` tablespace cannot serve as default temporary storage.

> **Note:** You cannot make a default temporary tablespace permanent.

You can specify a user-named default temporary tablespace when you create a database by using the `DEFAULT TEMPORARY TABLESPACE` extension to the `CREATE DATABASE` statement. If `SYSTEM` is dictionary managed, and if a default temporary tablespace is not defined at database creation, then `SYSTEM` is the default temporary storage. However, the database writes a warning in the **alert log** saying that a default temporary tablespace is recommended.

See Also:

- "Permanent and Temporary Data Files" on page 11-8
- *Oracle Database Administrator's Guide* to learn how to create a default temporary tablespace
- *Oracle Database SQL Language Reference* for the syntax of the `DEFAULT TEMPORARY TABLESPACE` clause of `CREATE DATABASE` and `ALTER DATABASE`

Tablespace Modes

The **tablespace mode** determines the accessibility of the tablespace.

Read/Write and Read-Only Tablespaces

Every tablespace is in a **write mode** that specifies whether it can be written to. The mutually exclusive modes are as follows:

- Read/write mode

  Users can read and write to the tablespace. All tablespaces are initially created as read/write. The `SYSTEM` and `SYSAUX` tablespaces and temporary tablespaces are permanently read/write, which means that they cannot be made read-only.

- Read-only mode

  Write operations to the data files in the tablespace are prevented. A read-only tablespace can reside on read-only media such as DVDs or WORM drives.

  Read-only tablespaces eliminate the need to perform backup and recovery of large, static portions of a database. Read-only tablespaces do not change and thus do not require repeated backup. If you recover a database after a media failure, then you do not need to recover read-only tablespaces.

See Also:

- *Oracle Database Administrator's Guide* to learn how to change a tablespace to read only or read/write mode
- *Oracle Database SQL Language Reference* for `ALTER TABLESPACE` syntax and semantics
- *Oracle Database Backup and Recovery User's Guide* for more information about recovery

Online and Offline Tablespaces

A tablespace can be **online** (accessible) or **offline** (not accessible) whenever the database is open. A tablespace is usually online so that its data is available to users. The `SYSTEM` tablespace and temporary tablespaces cannot be taken offline.

A tablespace can go offline automatically or manually. For example, you can take a tablespace offline for maintenance or backup and recovery. The database automatically takes a tablespace offline when certain errors are encountered, as when the **database writer (DBW)** process fails in several attempts to write to a data file. Users trying to access tables in an offline tablespace receive an error.

When a tablespace goes offline, the database does the following:

- The database does not permit subsequent DML statements to reference objects in the offline tablespace. An offline tablespace cannot be read or edited by any utility other than Oracle Database.
- Active transactions with completed statements that refer to data in that tablespace are not affected at the transaction level.
- The database saves undo data corresponding to those completed statements in a deferred undo segment in the `SYSTEM` tablespace. When the tablespace is brought online, the database applies the undo data to the tablespace, if needed.

See Also:

- "Online and Offline Data Files" on page 11-9
- "Database Writer Process (DBWn)" on page 15-8
- *Oracle Database Administrator's Guide* to learn how to alter tablespace availability

Tablespace File Size

A tablespace is either a **bigfile tablespace** or a **smallfile tablespace**. These tablespaces are indistinguishable in terms of execution of SQL statements that do not explicitly refer to data files or temp files. The difference is as follows:

- A smallfile tablespace can contain multiple data files or temp files, but the files cannot be as large as in a bigfile tablespace. This is the default tablespace type.
- A bigfile tablespace contains one very large data file or temp file. This type of tablespaces can do the following:
  - Increase the storage capacity of a database

    The maximum number of data files in a database is limited (usually to 64 KB files), so increasing the size of each data file increases the overall storage.
  - Reduce the burden of managing many data files and temp files

Bigfile tablespaces simplify file management with Oracle Managed Files and Automatic Storage Management (Oracle ASM) by eliminating the need for adding new files and dealing with multiple files.

– Perform operations on tablespaces rather than individual files

  Bigfile tablespaces make the tablespace the main unit of the disk space administration, backup and recovery, and so on.

Bigfile tablespaces are supported only for locally managed tablespaces with ASSM. However, locally managed undo and temporary tablespaces can be bigfile tablespaces even when segments are manually managed.

See Also:

- "Backup and Recovery" on page 18-9
- *Oracle Database Administrator's Guide* to learn how to manage bigfile tablespaces