15

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.

- Introduction to Logical Storage Structures
  Oracle Database allocates logical space for all data in the database.
- Overview of Data Blocks
  Oracle Database manages the logical storage space in the data files of a database in a unit called a **data block**, also called an *Oracle block* or *page*. A data block is the minimum unit of database I/O.
- Overview of Extents
  An extent is a unit of database storage made up of logically contiguous data blocks. Data blocks can be physically spread out on disk because of RAID striping and file system implementations.
- Overview of Segments
  A segment is a set of extents that contains all the data for a logical storage structure within a tablespace.
- Overview of Tablespaces
  A **tablespace** is a logical storage container for segments.

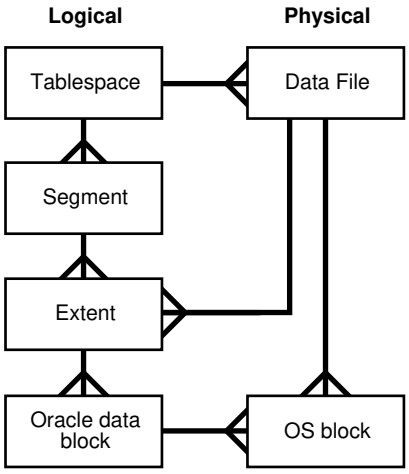
Introduction to Logical Storage Structures

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. The data in the data files is stored in operating system blocks.

The following figure is an entity-relationship diagram for physical and logical storage. The crow's foot notation represents a one-to-many relationship.

Figure 15-1 Logical and Physical Storage



- Logical Storage Hierarchy
  A segment contains one or more extents, each of which contains multiple data blocks.
- Logical Space Management
  Oracle Database must use logical space management to track and allocate the extents in a tablespace.

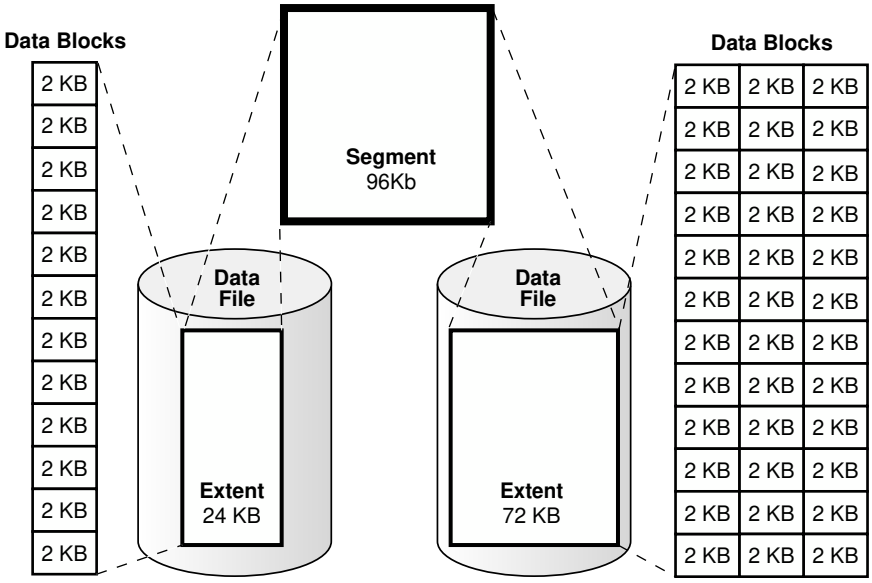
> **See Also:**
>
> "Physical Storage Structures"

Logical Storage Hierarchy

A segment contains one or more extents, each of which contains multiple data blocks.

The following figure 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 15-2 Segments, Extents, and Data Blocks Within a Tablespace



From the lowest level of granularity to the highest, Oracle Database stores data as follows:

- A data block is the smallest logical unit of data storage in Oracle Database.

  One logical data block corresponds to a specific number of bytes of persistent storage, for example, 2 KB. Data blocks are the smallest units of storage that Oracle Database can use or allocate.

  Traditionally, data files have been stored on magnetic disk or Solid State Disk (SSD). Oracle Database also supports Persistent Memory (PMEM) for storage of database files. Regardless of how the underlying data is physically stored, Oracle processes always read and write logical data blocks.

- An extent is a set of logically contiguous data blocks allocated for storing a specific type of information

  In the preceding graphic, 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.

- A tablespace is a database storage unit that contains one or more segments.

  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 the preceding graphic. 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:**
>
> "Introduction to Physical Storage Structures"

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:

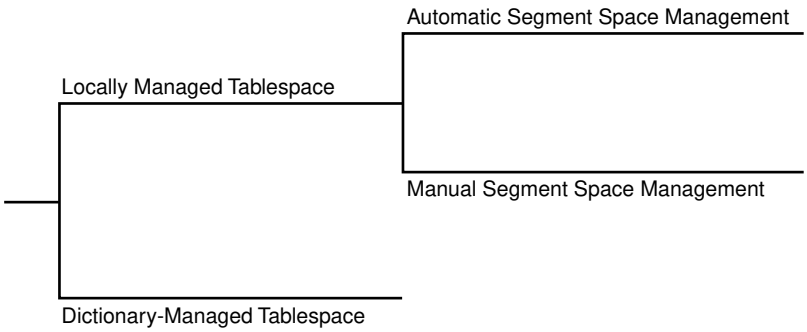
- Locally managed tablespaces (default)

  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.

Figure 15-3 shows the alternatives for logical space management in a tablespace.

Figure 15-3 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.
- Dictionary-Managed Tablespaces
  A dictionary-managed tablespace uses the data dictionary to manage its extents.

> **See Also:**
>
> "Overview of the Data Dictionary"

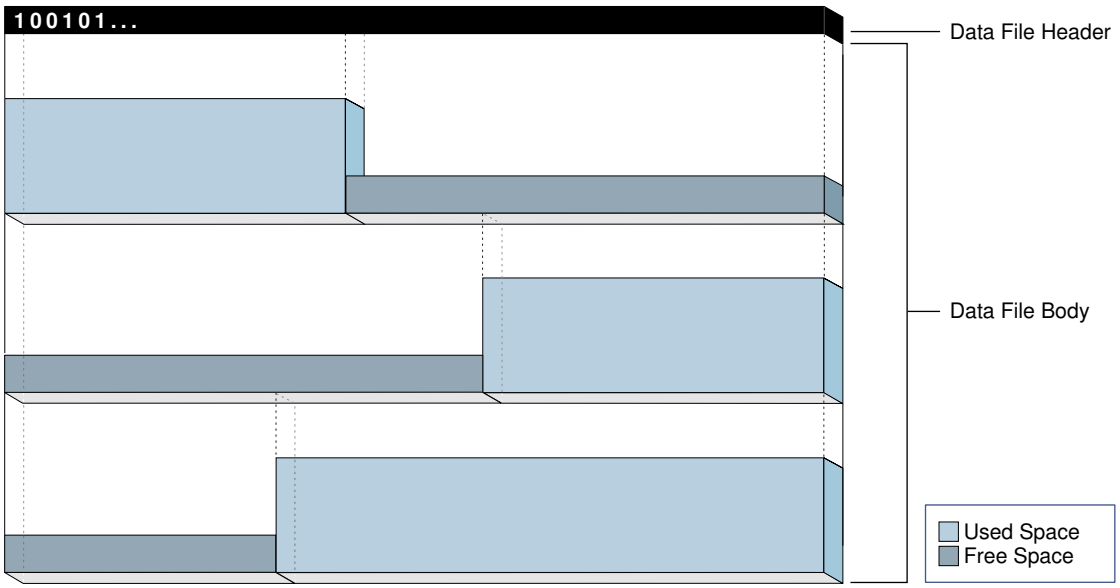
Locally Managed Tablespaces

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.

Figure 15-4 Bitmap-Managed Storage



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 **automatic segment space management (ASSM)** method uses bitmaps to manage space in a tablespace.
- Manual Segment Space Management
  The legacy **manual segment space management (MSSM)** method uses a linked list called a **free list** to manage free space in the segment.

Automatic Segment Space Management

The **automatic segment space management (ASSM)** method uses bitmaps to manage space in a tablespace.

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").
- 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 **manual segment space management (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) (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.

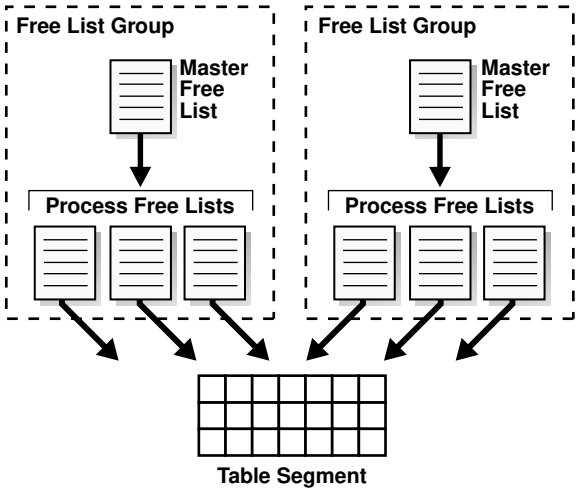
In addition to `PCTFREE`, MSSM requires you to control space allocation with SQL parameters such as `PCTUSED`, `FREELISTS`, and `FREELIST GROUPS`. `PCTUSED` sets the percentage of free space that must exist in a currently used block for the database to put it on the free list. For example, if you set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

For example, 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 does not fit in the block, and if the used space in the block is greater than or equal to `PCTUSED`, then the database removes the block from 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 15-5, 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 list* in the group. Space overhead for free lists, especially for free list groups, can be significant.

Figure 15-5 Free List Groups



Managing segment space manually can be complex. You must adjust `PCTFREE` and `PCTUSED` to reduce row migration 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:

- "Chained and Migrated Rows "
- *Oracle Database Administrator's Guide* to learn about locally managed tablespaces
- *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 a unit called a **data block**, also called an *Oracle block* or *page*. 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.
- 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.
- Data Block Compression
  The database can use **table compression** to eliminate duplicate values in a data block. This section describes the format of data blocks that use compression.
- 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.
- Overview of Index Blocks
  An **index block** is a special type of data block that manages space differently from table blocks. Oracle Database uses index blocks to manage the logical storage space in an index.

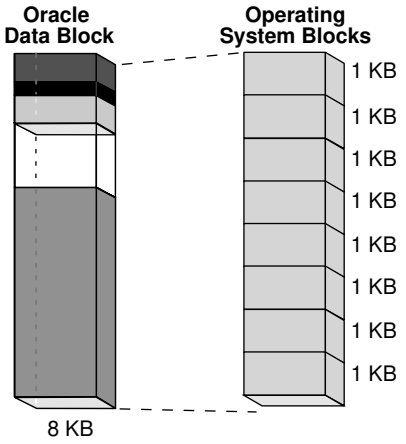
Data Blocks and Operating System Blocks

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.

The following figure 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 15-6 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.
- Tablespace Block Size
  You can create individual tablespaces whose block size differs from the `DB_BLOCK_SIZE` setting.

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.

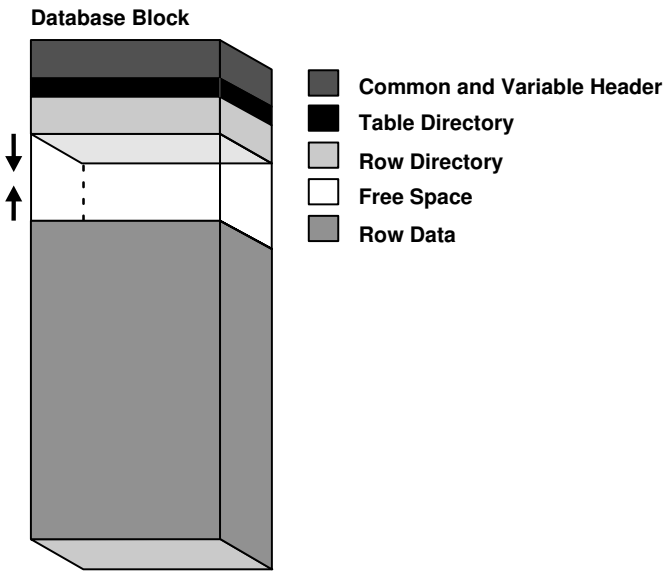
> **See Also:**
>
> *Oracle Database Administrator's Guide* to learn how to specify a nonstandard block size for a tablespace

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.

The following figure shows the format of an uncompressed data block.

Figure 15-7 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.
- 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.

> **See Also:**
>
> "Data Block Compression" to learn about compressed blocks

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 "Data Block Format", 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. In a table cluster, 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. The database can place a row anywhere in the bottom of the block. The row address is recorded in one of the slots of the row directory vector.

  A rowid points to a specific file, block, and row number. For example, in the rowid `AAAPecAAFAAAABSAAA`, the final `AAA` represents the row number. The row number is an index into an entry in the row directory. The row directory entry contains a pointer to the location of the row on the data block. If the database moves a row within a block, then the database updates the row directory entry to modify the pointer. The rowid stays constant.

  After the database allocates space in the row directory, the database does not reclaim this space after deleting rows. 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 a session inserts new rows 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.

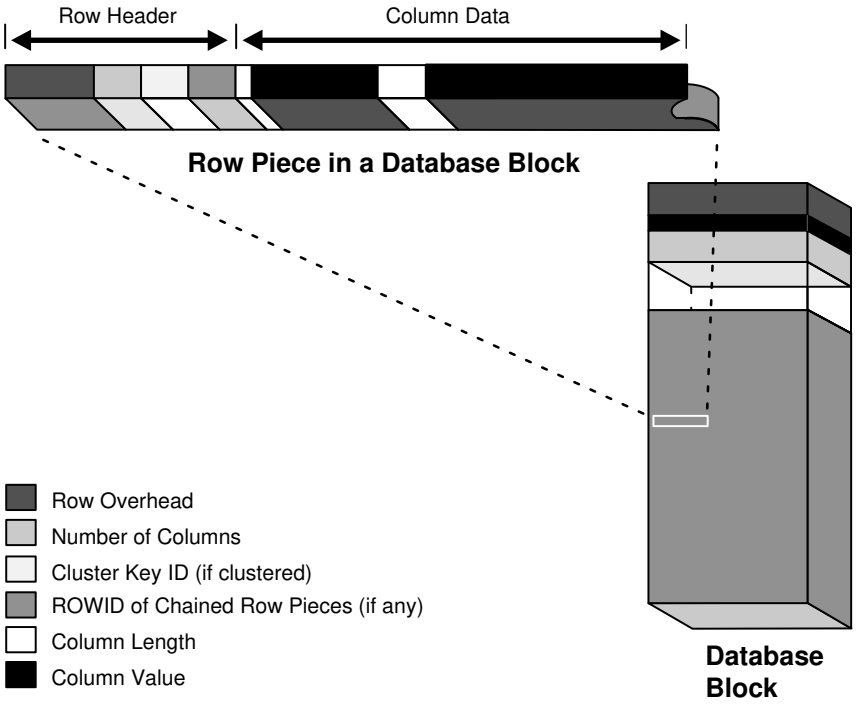
Row Format

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 sections. Each section is called a row piece. Each row piece has a row header and column data.

The following figure shows the format of a row.

Figure 15-8 The Format of a Row Piece



- Row Header
  Oracle Database uses the row header to manage the row piece stored in the block.
- 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.
- 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.

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. A data block usually contains only one row piece per row.
- Cluster keys for table clusters

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

> **See Also:**
>
> - "Chained and Migrated Rows "
> - "Overview of Table Clusters"

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 the figure in "Row Format", 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" and "Index Storage"

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 15-1 ROWID Pseudocolumn

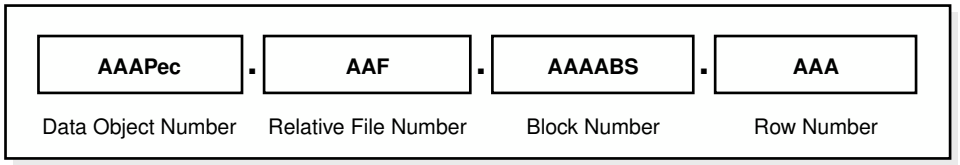
The following example queries the `ROWID` pseudocolumn to show the extended rowid of the row in the `employees` table for employee 100:

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

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

The following figure illustrates the format of an extended rowid.

Figure 15-9 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 the sample query). 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 the sample query).

- `BBBBBB`

  The data block number identifies the block that contains the row (block `AAAABS` in the sample query). 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 the sample query).

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"
> - *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. This section describes the format of data blocks that use compression.

The format of a data block that uses basic table 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.

Example 15-2 Format of Compressed Data Blocks

Assume that the following rows are stored in a data block for the seven-column `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 table or advanced row compression is applied to this table, the database replaces duplicate values with a symbol reference. The following conceptual representation of the compression shows the symbol `*` replacing `29-NOV-00` and `%` replacing `9999`:

```
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 15-1 conceptually represents the symbol table that maps symbols to values.

Table 15-1 Symbol Table

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

> **See Also:**
>
> "Table Compression"

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.

Free space in a data block can also shrink during updates, as when changing a trailing null value to a non-null 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` SQL parameter sets the minimum percentage of a data block reserved as free space for updates to existing rows. `PCTFREE` is important for preventing row migration and avoiding wasted space.
- 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, setting `PCTFREE` to 20% prevents the total amount of free space from dropping to 5% of the block, but allows 50% of the block to be free.
- Chained and Migrated Rows
  Oracle Database uses chaining and migration to manage rows that are too large to fit into a single block.

Percentage of Free Space in Data Blocks

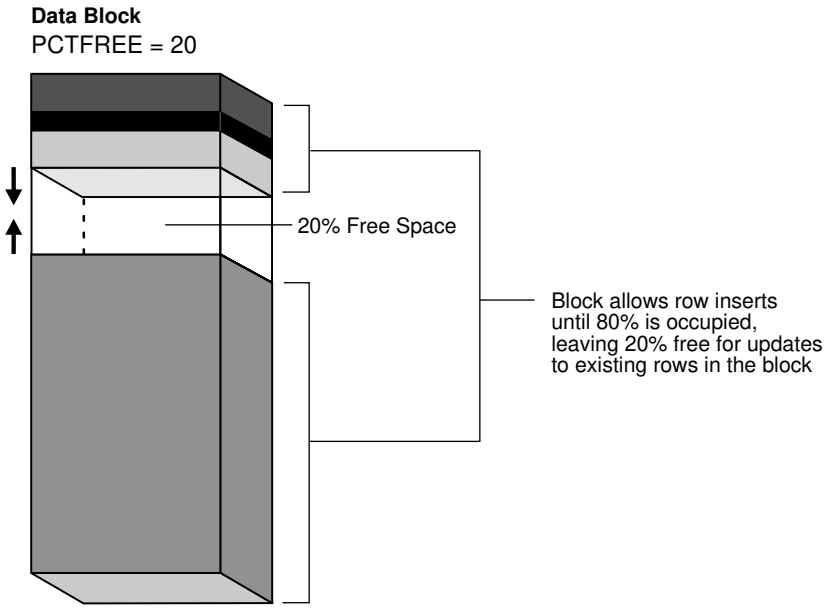
The `PCTFREE` SQL parameter sets the minimum percentage of a data block reserved as free space for updates to existing rows. `PCTFREE` is important for preventing row migration and avoiding wasted space.

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 15-10 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 15-10 PCTFREE



Note:

This discussion does not apply to LOB data types, which do not use the `PCTFREE` storage parameter or free lists.

See Also:

- *Oracle Database SecureFiles and Large Objects Developer's Guide*
- *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, setting `PCTFREE` to 20% prevents the total amount of free space from dropping to 5% of the block, but allows 50% of the block to be free.

- Optimization by Increasing Free Space
  Some DML statements can increase free space in data blocks.
- Optimization by Coalescing Fragmented Space
  Released space may or may not be contiguous with the main area of free space in a data block. Noncontiguous free space is called *fragmented space*.

Optimization by Increasing Free Space

Some DML statements can increase free space in data blocks.

The following statements can increase 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 advanced row compression

  If `INSERT` statements 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 if it is 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), and if space is needed, then the statement can use the space made available, but only after the other transaction commits.

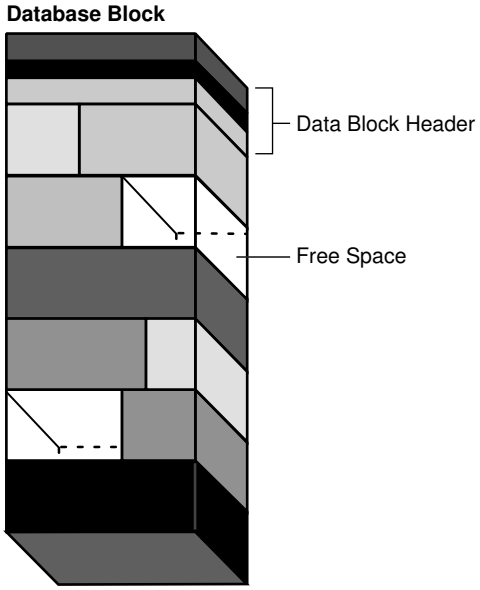
> **See Also:**
>
> *Oracle Database Administrator's Guide* to learn about advanced row compression

Optimization by Coalescing Fragmented Space

Released space may or may not be contiguous with the main area of free space in a data block. Noncontiguous free space is called *fragmented space*.

The following figure shows a data block with noncontiguous free space.

Figure 15-11 Data Block with Fragmented Space

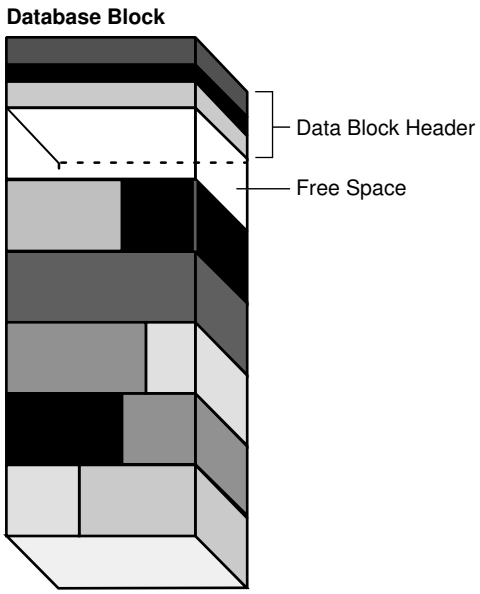


Oracle Database automatically and transparently coalesces the free space of a data block *only* when the following conditions are true:

- 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 15-12 shows a data block after space has been coalesced.

Figure 15-12 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.

Chained and Migrated Rows

Oracle Database uses chaining and migration to manage rows that are too large to fit into a single block.

> **Note:**
>
> A chained row in a standard table is different from a row chain in a blockchain table. Oracle Database uses different technology to manage rows in a blockchain table.

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` , 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 15-13 depicts 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 15-13 Row Chaining

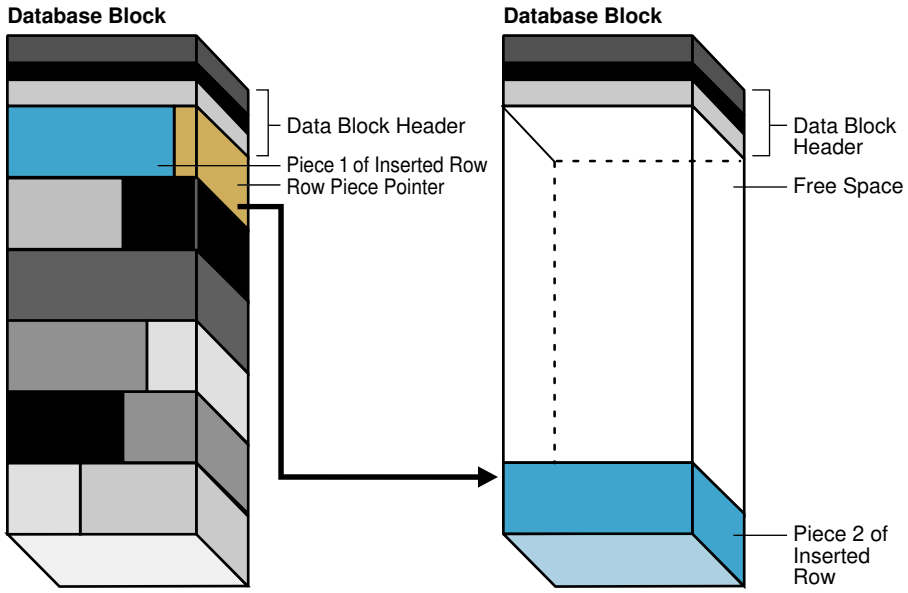
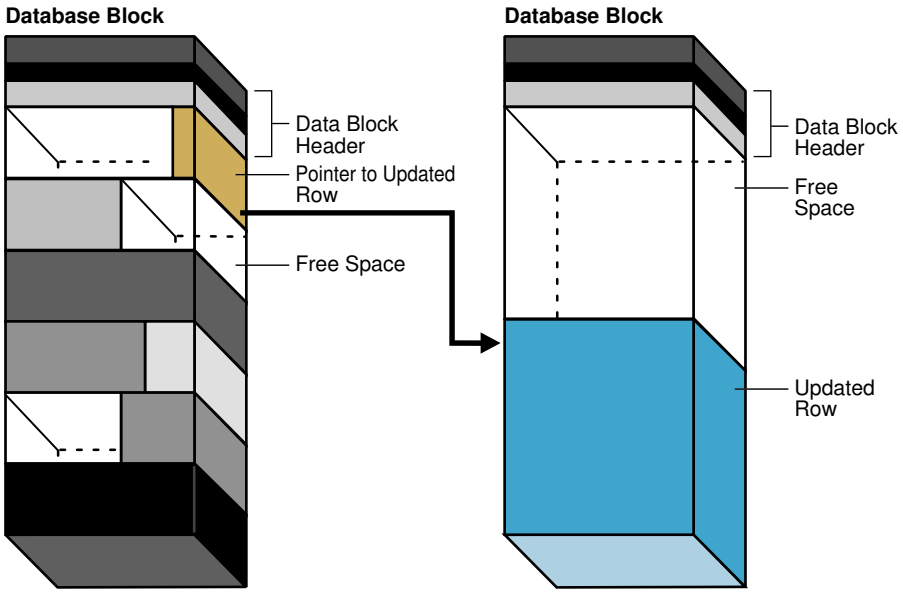


Figure 15-14, the left block contains a row that is updated so that the row is now too large for the block. The database moves the entire row to the right block and leaves a pointer to the migrated row in the left block.

Figure 15-14 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" and "Rowids of Row Pieces"
- "Overview of Blockchain Tables"
- *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 Index Blocks

An **index block** is a special type of data block that manages space differently from table blocks. Oracle Database uses index blocks to manage the logical storage space in an index.

- Types of Index Blocks
  An index contains a root block, branch blocks, and leaf blocks.
- Storage of Index Entries
  Index entries are stored in index blocks in the same way as table rows in a data block The index entries in the block portion are not stored in binary order, but in a heap.
- Reuse of Slots in an Index Block
  The database can reuse space within an index block.
- Coalescing an Index Block
  Index coalescing compacts existing index data in place and, if the reorganization frees blocks, leaves the free blocks in the index structure. Thus, coalescing does not release index blocks for other uses or cause the index to reallocate blocks.

Types of Index Blocks

An index contains a root block, branch blocks, and leaf blocks.

The block types are defined as follows:

- Root block

  This block identifies the entry point into the index.
- Branch blocks

  The databases navigates through branch blocks when searching for an index key.
- Leaf blocks

  These blocks contain the indexed key values rowids that point to the associated rows. The leaf blocks store key values in sorted order so that the database can search efficiently for all rows in a range of key values.

Storage of Index Entries

Index entries are stored in index blocks in the same way as table rows in a data block The index entries in the block portion are not stored in binary order, but in a heap.

The database manages the row directory in an index block differently from the directory in a data block. The entries in the row directory (not the entries in the body of the index block) are ordered by key value. For example, in the row directory, the directory entry for index key `000000` precedes the directory entry for index key `111111`, and so on.

The ordering of entries in the row directory improves the efficiency of index scans. In a range scan, the database must read all the index keys specified in the range. The database traverses the branch blocks to identify the leaf block that contains the first key. Because entries in the row directory are sorted, the database can use a binary search to find the first index key in the range, and then progress sequentially through the entries in the row directory until it finds the last key. In this way, the database avoids reading all the keys in the leaf block body.

> **See Also:**
>
> "Data Block Overhead"

Reuse of Slots in an Index Block

The database can reuse space within an index block.

For example, an application may insert a value into a column and then delete the value. When a row requires space, the database can reuse the index slot formerly occupied by the deleted value.

An index block usually has many more rows than a heap-organized table block. The ability to store many rows in a single index block makes it easier for the database to maintain an index because it avoids frequent splits of the block to store new data.

An index cannot coalesce itself, although you can manually coalesce it with an `ALTER INDEX` statement with the `REBUILD` or `COALESCE` options. For example, if you populate a column with values `1` to `500000`, and if you then delete the rows that contain even numbers, then the index will contain 250,000 empty slots. The database reuses a slot only if it can insert data that fits into an index block that contains an empty slot.

Coalescing an Index Block

Index coalescing compacts existing index data in place and, if the reorganization frees blocks, leaves the free blocks in the index structure. Thus, coalescing does not release index blocks for other uses or cause the index to reallocate blocks.

Oracle Database does not automatically compact the index: you must run an `ALTER INDEX` statement with the `REBUILD` or `COALESCE` options.

Figure 15-15 shows 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 15-15 Index Before Coalescing

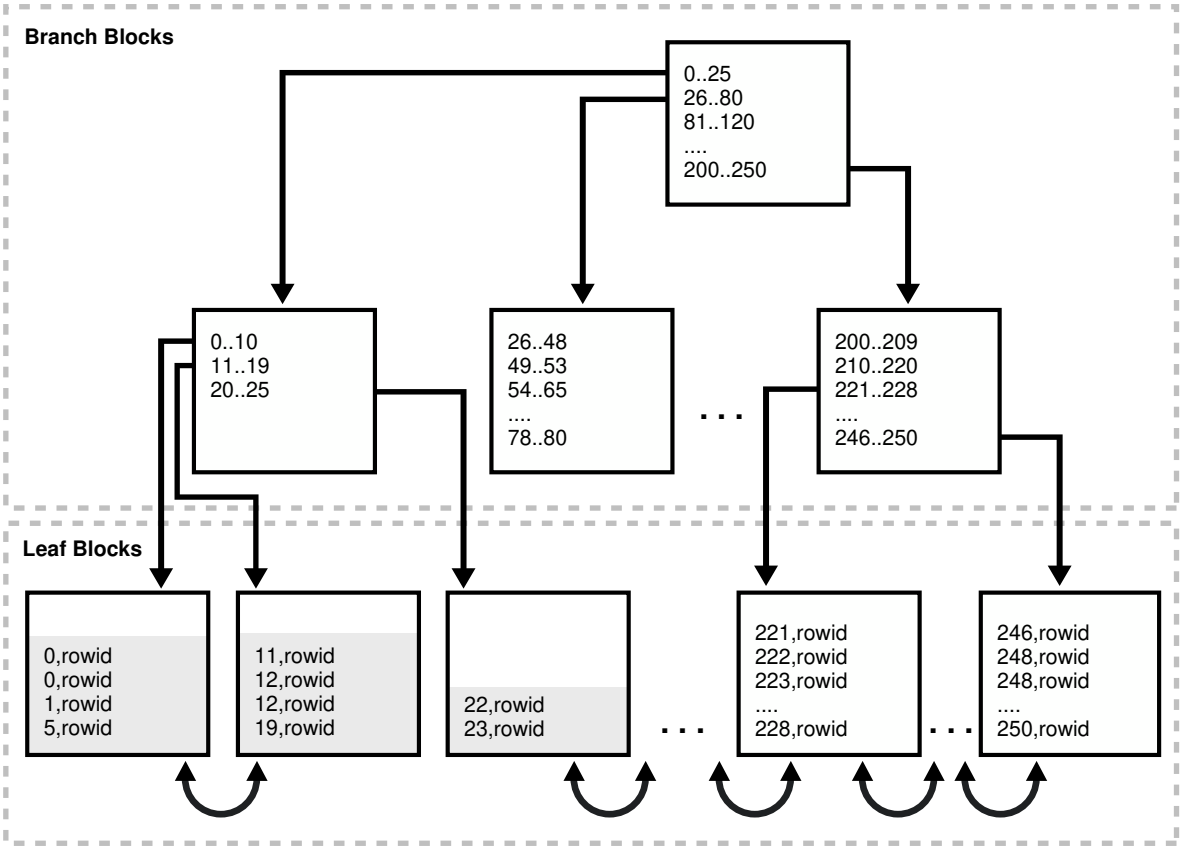
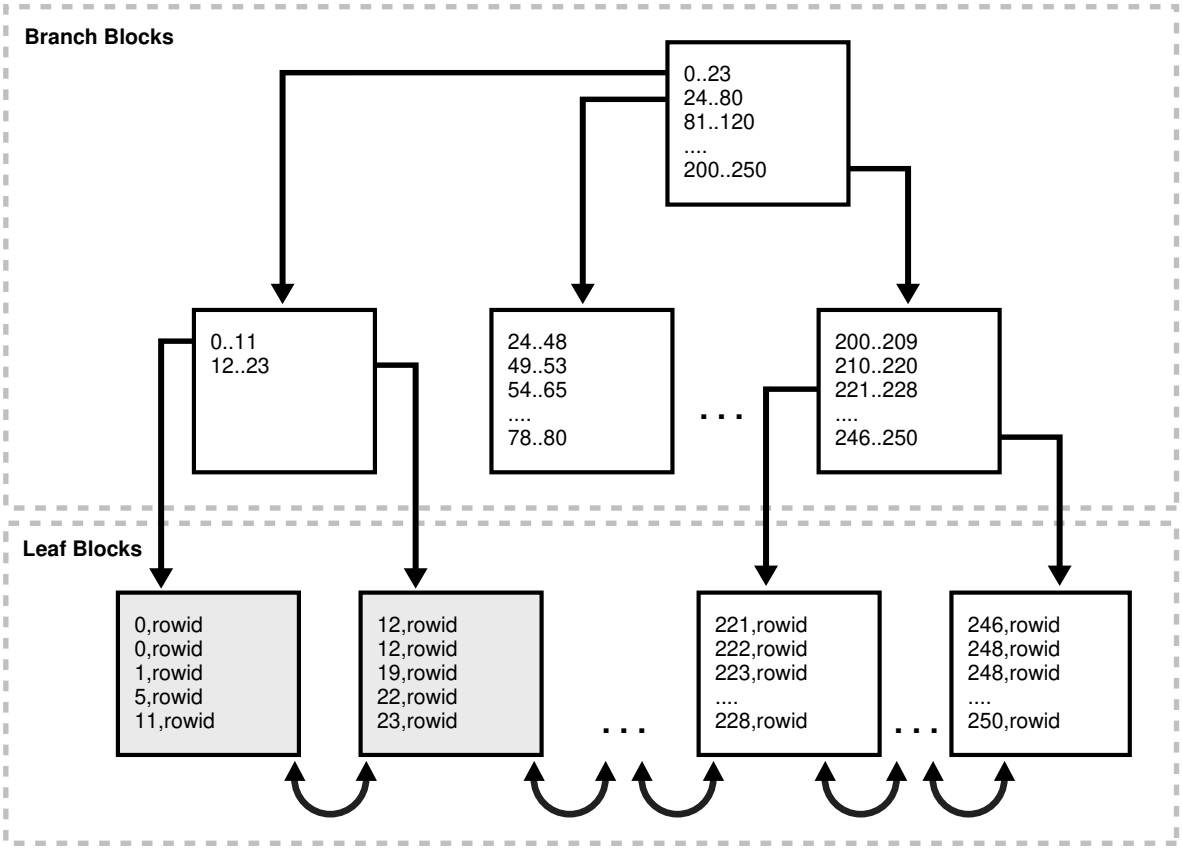


Figure 15-16 shows the index in Figure 15-15 after the index has been coalesced. The first two leaf blocks are now full, as indicated by the gray fill lines, and the third leaf block has been freed.

Figure 15-16 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

Overview of Extents

An extent is a unit of database storage made up of logically contiguous data blocks. Data blocks 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.
- 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` statement.

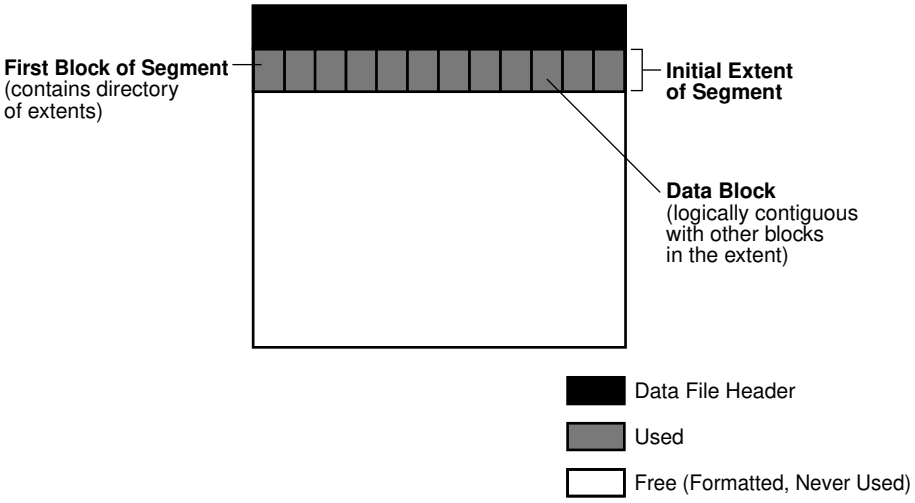
- Storage Parameters for Extents
  Every segment is defined by storage parameters expressed in terms of extents. These parameters control how Oracle Database allocates free space for a segment.

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, 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 15-17 shows the initial extent in a segment in a data file that previously contained no data.

Figure 15-17 Initial Extent of a Segment

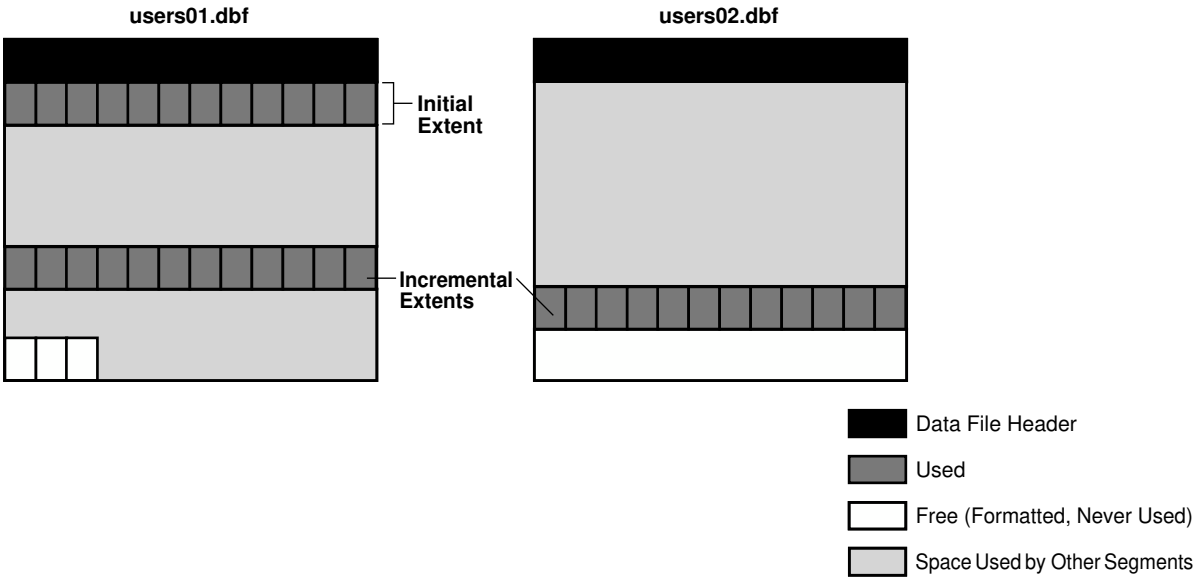


If the initial extent become full, and if more space is required, then the database automatically allocates an incremental extent for this segment. An incremental extent is a subsequent extent created for the segment.

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 15-18 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 then allocate the next extent in `users01.dbf`.

Figure 15-18 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.

> **Note:**
>
> This section applies to serial operations, in which one server process parses and runs a statement. The database allocates extents differently in parallel SQL statements, which entail multiple server processes.

> **See Also:**
>
> - "Segment Space and the High Water Mark"
> - *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` statement.

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. You can also drop the segment using the `DBMS_SPACE_ADMIN` package.

> **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.

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:

- 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.
- 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.
- Rebuild or coalesce the index.
- 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. Starting in Oracle Database 11g Release 2 (11.2.0.2), you can also use `TRUNCATE` with the `DROP ALL STORAGE` option to drop entire segments.
- 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.

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:**
>
> - "Coalescing an Index Block"
> - "Undo Tablespaces"
> - "Segment Space and the High Water Mark"
> - *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 settings 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 database 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 an index on a table.

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.
- Temporary Segments
  When processing a query, Oracle Database often requires temporary workspace for intermediate stages of SQL statement execution.
- Undo Segments
  Oracle Database maintains records of the actions of transactions, collectively known as **undo data**.
- 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.

> **See Also:**
>
> "Logical Space Management" to learn more about 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, indexes, and partitions.

User Segment Creation

By default, the database uses deferred segment creation to update only database metadata when creating tables, indexes, and 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 avoids 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.

The `DBMS_SPACE_ADMIN` package manages segments for empty objects. You can use this PL/SQL package to do the following:

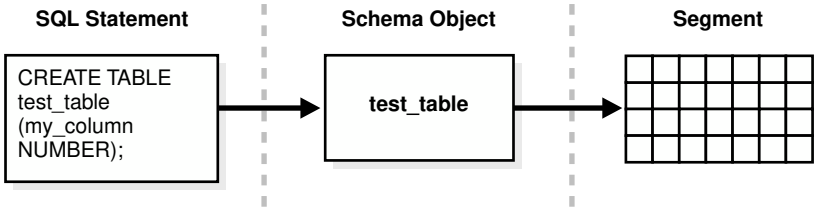
- Manually materialize segments for empty tables or partitions that do not have segments created
- Remove segments from empty tables or partitions that currently have an empty segment allocated

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 15-19, the database creates one segment for the table.

Figure 15-19 Creation of a User Segment

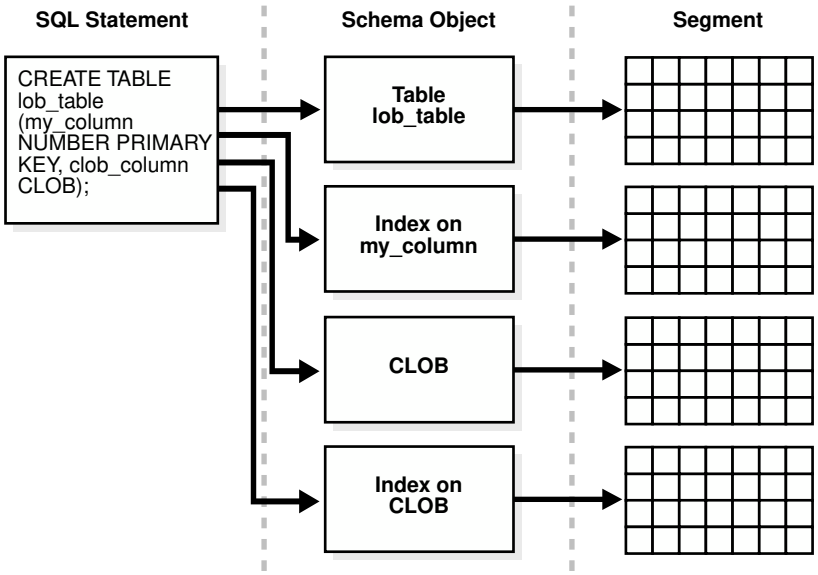


When you create a table with a primary key or unique key, Oracle Database automatically creates an index for this key. Again assume that deferred segment creation is disabled. You create a table as follows:

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

Figure 15-20 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. Thus, the `CREATE TABLE` statement results in the creation of *four* different segments.

Figure 15-20 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. The parameters

affect the efficiency of data retrieval and storage for the data segment associated with the object.

> **See Also:**
>
> - *Oracle Database SecureFiles and Large Objects Developer's Guide* to learn about internal LOBs
> - "Storage Parameters for Extents "
> - *Oracle Database Administrator's Guide* to learn how to manage deferred segment creation
> - *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.
- Allocation of Segments for Temporary Tables and Indexes
  Oracle Database can allocate temporary segments for temporary tables and their indexes.

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.

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:**
>
> - "Overview of the Online Redo Log"
> - *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 Segments for Temporary Tables and Indexes

Oracle Database can 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 itself and cannot access extents allocated for other sessions.

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

A temporary tablespace of the current user allocates segments for a temporary table. For example, 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:**
>
> - "Overview of Temporary Tables"
> - *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 data 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 records. In this way, Oracle Database can efficiently access undo data without needing to read external logs.

Undo data for permanent objects 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.

The database separates undo data into two streams. A temporary undo stream encapsulates only undo records generated by changes to temporary objects, whereas a permanent undo stream encapsulates only undo records for permanent objects. The database manages temporary and permanent undo independently. Undo separation decreases storage and enhances performance by doing the following:

- Enabling you to configure permanent and undo tablespace sizes that best fit the workloads for permanent and temporary tables
- Reducing the size of redo written to the online redo log
- Avoiding the need to back up temporary undo data

On an Active Data Guard instance, DML on global temporary tables requires undo to be generated in temporary undo segments.

- 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.
- 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.
- Temporary Undo Segments
  A **temporary undo segment** is an optional space management container for temporary undo data only.

See Also:

- "Use of the Online Redo Log"
- "Temporary Undo Segments"
- *Oracle Database Administrator's Guide* to learn about temporary undo segments
- *Oracle Database Reference* to learn about the `TEMP_UNDO_ENABLED` initialization parameter

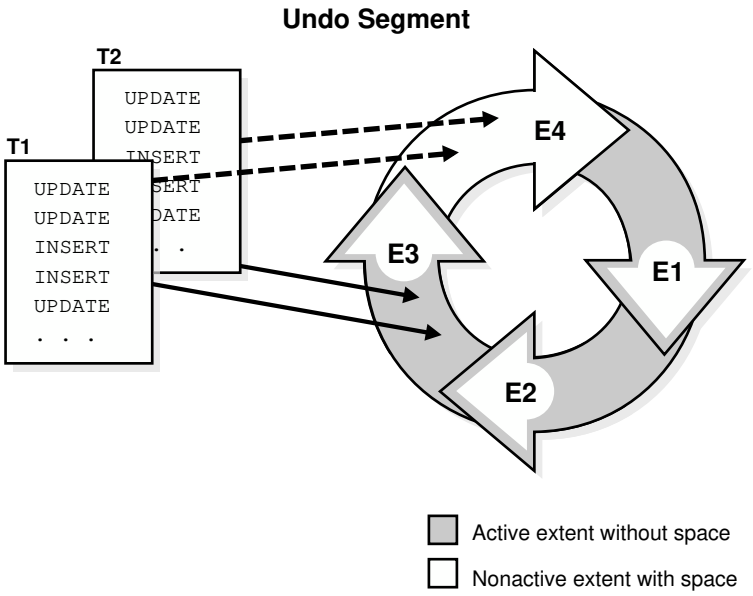
Undo Segments and Transactions

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 15-21 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).

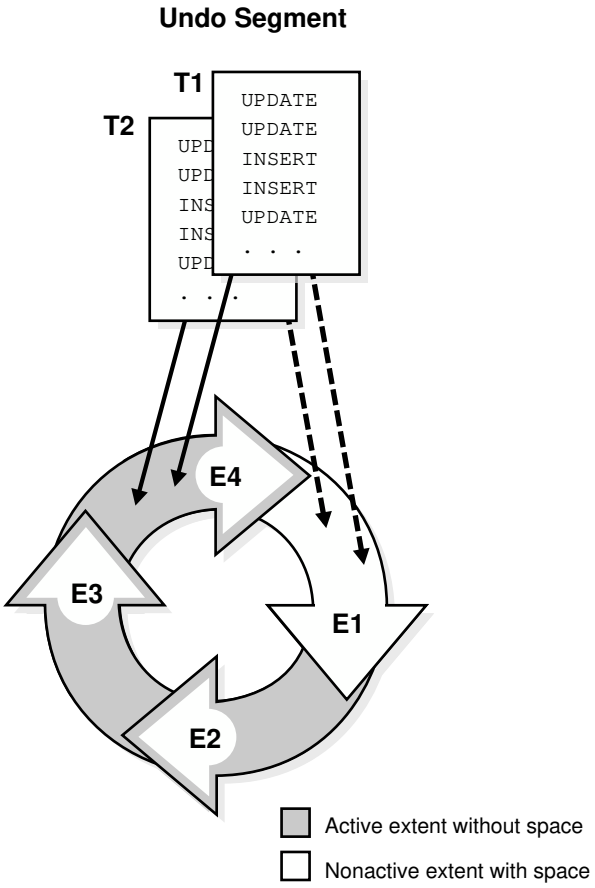
Figure 15-21 Ring of Allocated Extents in an Undo Segment



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 15-21 shows transactions T1 and T2 writing simultaneously to extent E3. Within an undo extent, a data block contains data for only one transaction.

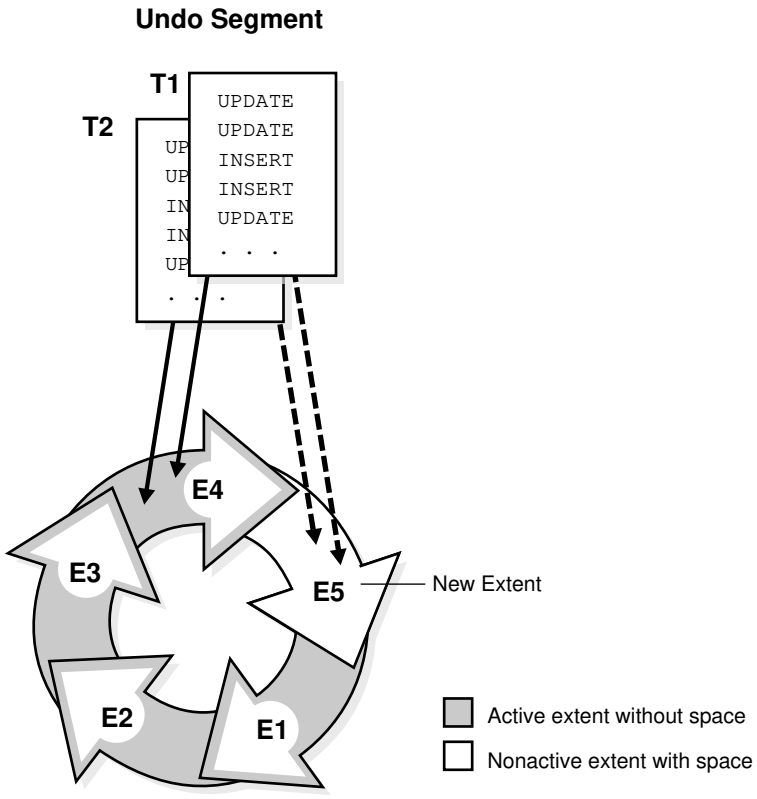
As the current undo extent fills, the first transaction needing space checks the availability of the next allocated extent in the ring. If the next extent does *not* contain data from an active transaction, then this extent becomes the current extent. Now all transactions that need space can write to the new current extent. In Figure 15-22, when E4 is full, T1 and T2 continue writing to E1, overwriting the nonactive undo data in E1.

Figure 15-22 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 15-23 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 15-23 Allocation of a New Extent for an Undo Segment



> **See Also:**
>
> *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.

Temporary Undo Segments

A **temporary undo segment** is an optional space management container for temporary undo data only.

Undo records for changes to temporary tables are both session-specific and useful only for read consistency and transaction rollback. Before Oracle Database 12c, the database always stored these records in the online redo log. Because changes to temporary objects are not logged in the online redo log, writing undo for temporary objects into temporary undo segments

saves space in the online redo log and archived redo log files. The database does not log changes to the undo or changes to the temporary table, which improves performance.

You can set the `TEMP_UNDO_ENABLED` initialization parameter so that temporary tables store undo data in a temporary undo segment. When this parameter is `TRUE`, the database allocates temporary undo segments from temporary tablespaces.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* to learn about temporary undo segments
> - *Oracle Database Reference* to learn about the `TEMP_UNDO_ENABLED` initialization parameter

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:

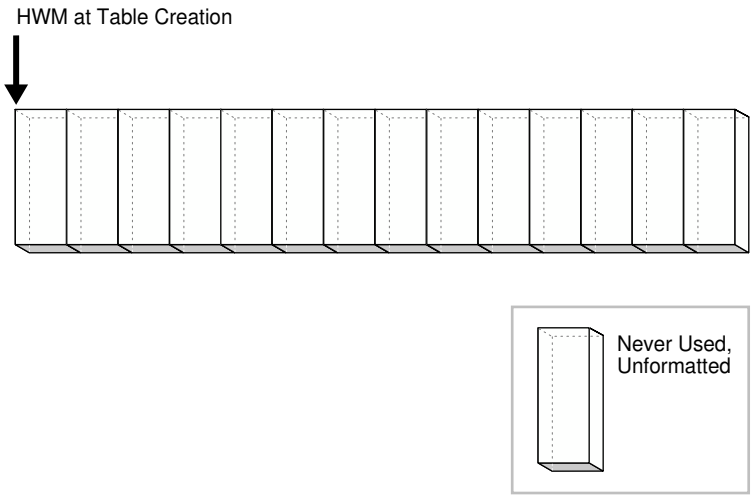
- Above the HWM

  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 15-24 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.

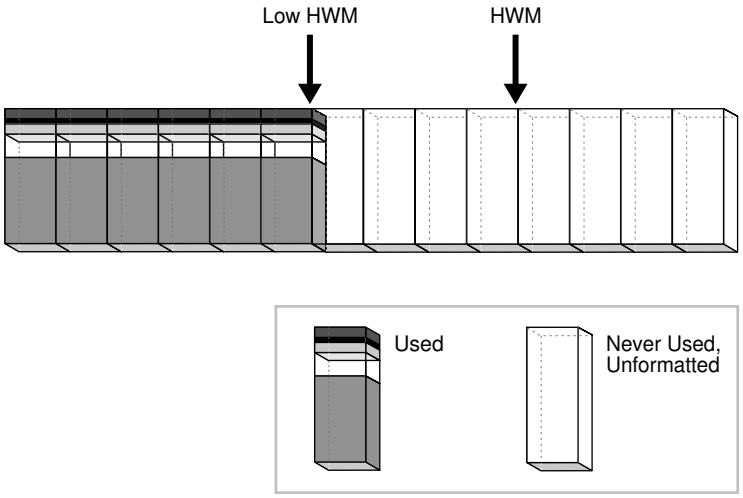
Figure 15-24 HWM at Table Creation



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.

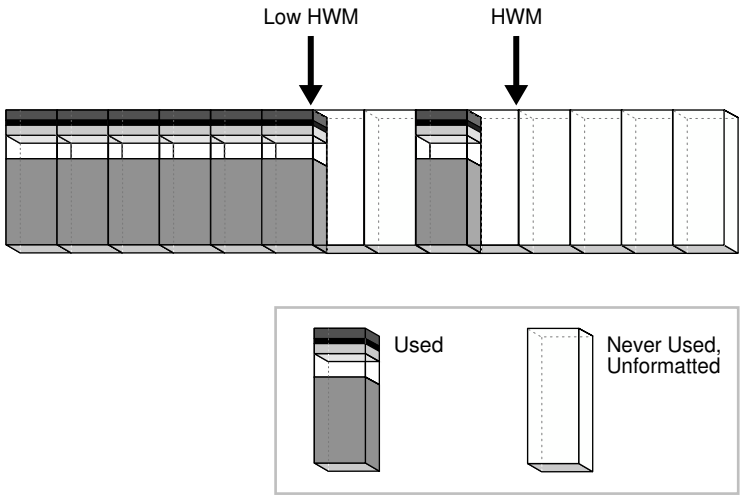
In Figure 15-25, the blocks below the HWM are allocated, whereas blocks above the HWM are neither allocated or formatted. As inserts occur, the database can write to any block with available space. The low high water mark (low HWM) marks the point below which all blocks are known to be formatted because they either contain data or formerly contained data.

Figure 15-25 HWM and Low HWM



In Figure 15-26, 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 15-26, the blocks to either side of the newly filled block are unformatted.

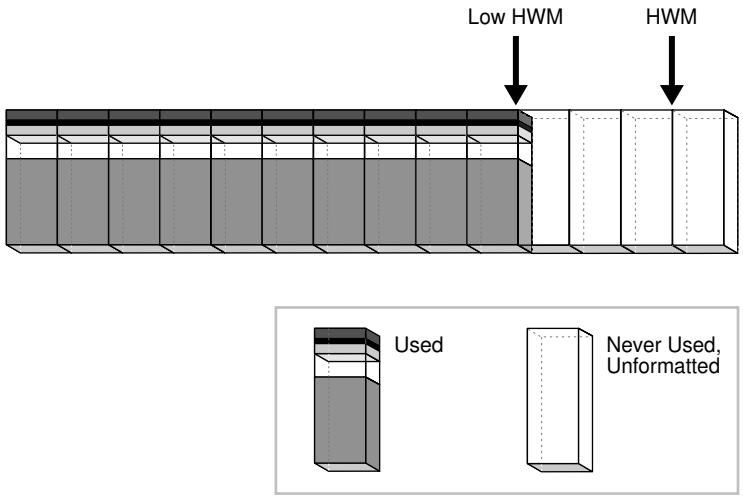
Figure 15-26 HWM and Low HWM



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 15-26. 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 15-27, the database advances the HWM to the right, allocating a new group of unformatted blocks.

Figure 15-27 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.

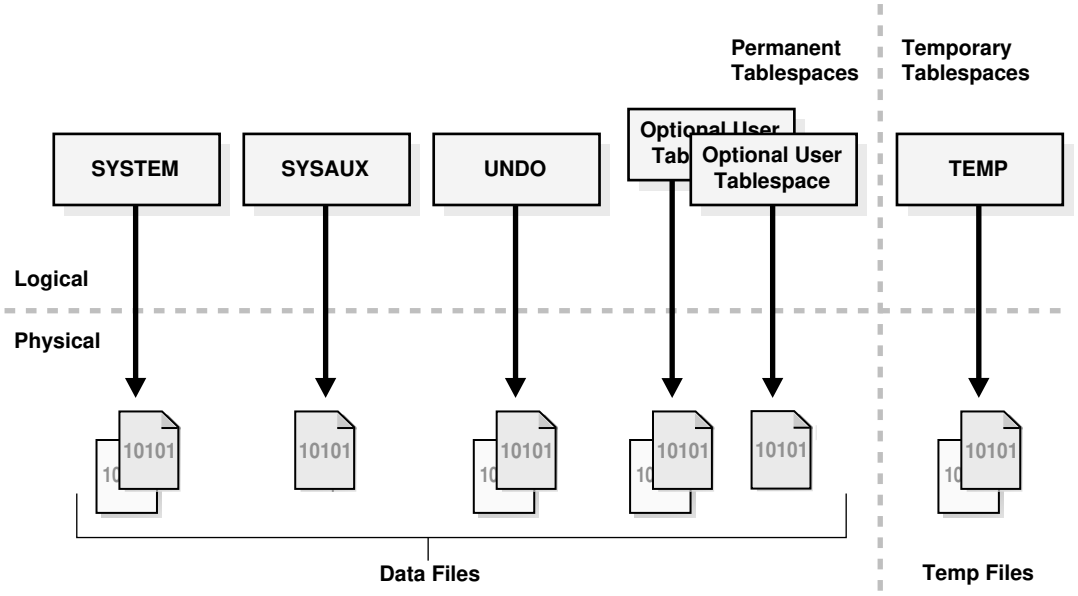
- Tablespaces in a Multitenant Environment
  In a CDB, each PDB and application root has its own set of tablespaces.
- Permanent Tablespaces
  A **permanent tablespace** groups persistent schema objects. The segments for objects in the tablespace are stored physically in data files.
- Temporary Tablespaces
  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. A **temp file** stores temporary tablespace data.
- Tablespace Modes
  The tablespace mode determines the accessibility of the tablespace.
- 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.

Tablespaces in a Multitenant Environment

In a CDB, each PDB and application root has its own set of tablespaces.

Every CDB root, PDB, and application root must have the `SYSTEM` and `SYSAUX` tablespaces. The following figure shows the tablespaces in a typical container.

Figure 15-28 Tablespaces



A CDB contains the following:

- One control file
- One online redo log
- One or more undo tablespaces

  Only a common user who has the appropriate privileges and whose current container is the CDB root can create an undo tablespace. At any given time, a CDB is either in either of the following undo modes:

  – Local undo mode

    In this case, each PDB has its own undo tablespace. If a CDB is using local undo mode, then the database automatically creates an undo tablespace in every PDB. Local undo provides advantages such as the ability to perform a hot clone of a PDB, and relocate PDBs quickly. Also, local undo provides level of isolation and enables faster unplug and point-in-time recovery operations.

    A local undo tablespace is required for each node in an Oracle Real Application Clusters (RAC) cluster in which the PDB is open. For example, if you move a PDB from a two-node cluster to a four-node cluster, and if the PDB is open in all nodes, then the database automatically creates the additional required undo tablespaces. If you move the PDB back again, then you can drop the redundant undo tablespaces.

    > **Note:**
    >
    > By default, Database Configuration Assistant (DBCA) creates new CDBs with local undo enabled.

  – Shared undo mode

In a single-instance CDB, only one active undo tablespace exists. For an Oracle RAC CDB, one active undo tablespace exists for every instance. All undo tablespaces are visible in the data dictionaries and related views of all containers.

The undo mode applies to the entire CDB, which means that either every container uses shared undo, or every container uses local undo. You can switch between undo modes in a CDB, which necessitates re-starting the database.

- `SYSTEM` and `SYSAUX` tablespaces for every container

  The CDB root, each application root, and each PDB has its own `SYSTEM` and `SYSAUX` tablespaces. Each container also has its own set of data dictionary tables describing the objects that reside in the container.

- Zero or more user-created tablespaces

  In a typical use case, each PDB has its own set of non-system tablespaces. These tablespaces contain the data for user-defined schemas and objects in the PDB. You can limit the amount of storage used by the data files for a PDB by using the `STORAGE` clause in a `CREATE PLUGGABLE DATABASE` or `ALTER PLUGGABLE DATABASE` statement.

  The storage of the data dictionary within the PDB enables it to be portable. You can unplug a PDB from a CDB, and plug it in to a different CDB.

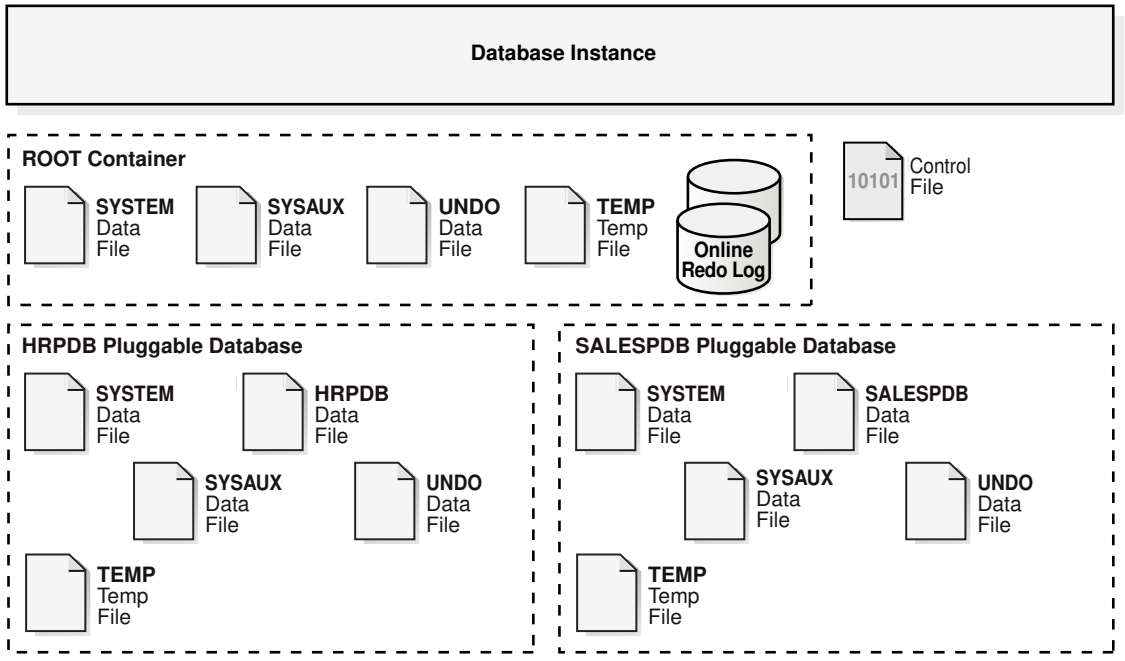
- A set of temp files for every container

  One default temporary tablespace exists for the CDB root, and one for each application root, application PDB, and PDB.

Example 15-3 CDB in Local Undo Mode

This example shows aspects of the physical storage architecture of a CDB with two PDBs: `hrpdb` and `salespdb`. In this example, the database uses local undo mode, and so has undo data files in the CDB root, `hrpdb`, and `salespdb`.

Figure 15-29 Physical Architecture of a CDB in Local Undo Mode



> **See Also:**
>
> "Data Dictionary Separation in a CDB"

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
- 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.

- 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 SYSAUX Tablespace
  The `SYSAUX` tablespace is an auxiliary tablespace to the `SYSTEM` tablespace.
- Undo Tablespaces
  An **undo tablespace** is a locally managed tablespace reserved for system-managed undo data.
- Shadow Tablespaces
  A **shadow tablespace** is a bigfile tablespace intended for **shadow lost write protection**.

> **See Also:**
>
> - *Oracle Database Utilities*
> - *Oracle Database Administrator's Guide* to learn how to transport tablespaces
> - *Oracle Database Utilities* to learn about Oracle Data Pump

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" for information about the permanent online condition of the SYSTEM tablespace
> - *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.

Because SYSAUX is the default tablespace for many Oracle Database features and products that previously required their own tablespaces, it reduces the number of tablespaces required by the database. It also reduces the load on the SYSTEM tablespace.

Database creation or upgrade automatically creates the SYSAUX tablespace. 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.

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 mode**.
- 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.

> **See Also:**
>
> "Undo Segments"

Automatic Undo Management Mode

Undo tablespaces require the database to be in the **default automatic undo mode**.

Automatic 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.

A new installation of Oracle Database automatically creates an undo tablespace. Earlier versions of Oracle Database may not include an undo tablespace and use legacy rollback segments instead, known as manual undo management mode. 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

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

Shadow Tablespaces

A **shadow tablespace** is a bigfile tablespace intended for **shadow lost write protection**.

> **Note:**
>
> Shadow lost write protection is not related to lost write protection that is configured with the `DB_LOST_WRITE_PROTECT` initialization parameter and a standby database.

- Purpose of Shadow Tablespaces
  Shadow lost write protection provides fast detection and immediate response to a **lost write**.

- How Shadow Tablespaces Work
  Lost write protection requires two tablespaces: a shadow tablespace, and a non-shadow tablespace whose blocks are tracked by the shadow tablespace.
- User Interface for Shadow Tablespaces
  You enable and disable shadow lost write protection using the `ALTER PLUGGABLE DATABASE` command.
- Example: Configuring Lost Write Protection
  This example enables shadow lost write tracking for a set of tablespaces.

Purpose of Shadow Tablespaces

Shadow lost write protection provides fast detection and immediate response to a **lost write**.

A data block lost write occurs when an I/O subsystem acknowledges the completion of the block write even though the write did not occur or when a former image of the block overwrites the current image.

An undetected lost write can result in data corruption because the incorrect data can be used for other DML transactions. For example, a transaction can read old and incorrect data from one table, and then update hundreds of other tables based on this data. In this way, data corruption can spread throughout the database.

Shadow lost write protection provides the following benefits:

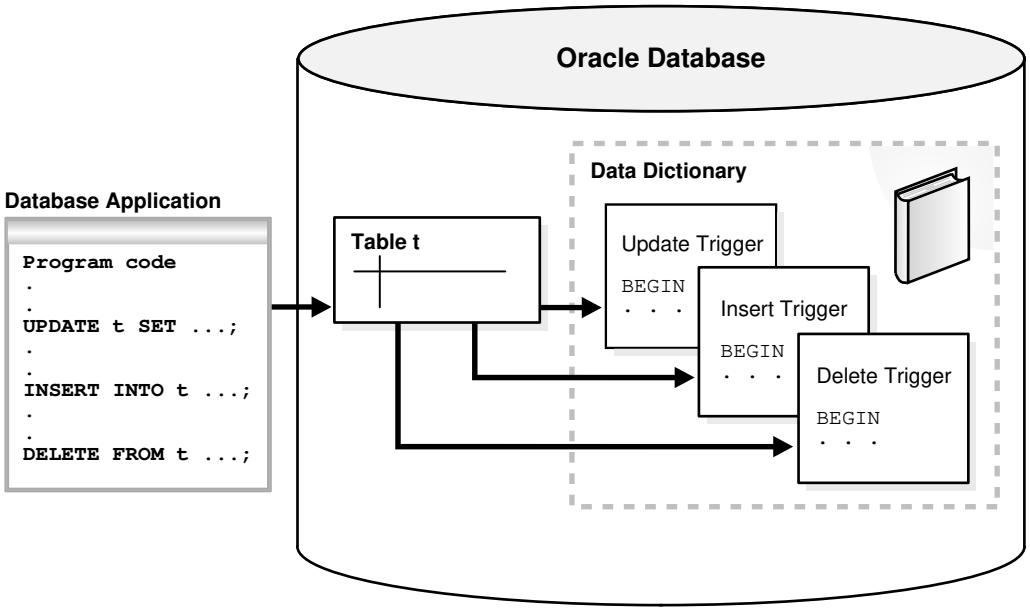
- It detects a lost write before it is consumed for standard DML, SQL*Loader conventional path load, direct path load, and RMAN backups.
- You can enable shadow lost write protection for specific tablespaces and data files. You do not need to track all data.
- You can replace one shadow tablespace with another to change its configuration or location.
- You can suspend and resume shadow lost write protection for a tablespace or data file.
- You can enable or disable it for a PDB with a single `ALTER DATABASE ... LOST WRITE PROTECTION` statement. The `PROP$` table indicates whether tracking is enabled for a PDB.

> **See Also:**
>
> *Oracle Database SQL Language Reference* to learn more about the `LOST WRITE PROTECTION` clause

How Shadow Tablespaces Work

Lost write protection requires two tablespaces: a shadow tablespace, and a non-shadow tablespace whose blocks are tracked by the shadow tablespace.

The following figure provides a sample scenario. The data files in tablespaces `TBS1` and `TBS2` are tracked by a shadow tablespace. Only data file `DBF6` in tablespace `TBS3` is tracked by the shadow tablespace.



One tracked data file maps to one shadow extent in a shadow tablespace. Every data block in a tracked data file has a corresponding entry in a shadow block. This entry contains the SCN of the tracked data block. When a tracked data block is read from disk, shadow lost write protection compares the SCN for the block in the shadow tablespace with the SCN of the most recent write in the tracked data block. If the shadow entry has an SCN greater than the data block being read, then a lost write has occurred, prompting an error.

The shadow extent is sized with significant extra space to prevent the automatic resizing of data files from causing the shadow extent to grow too large. If a tracked data file is resized either manually or automatically, and if the shadow extent needs to grow, then the database attempts to resize the tracking data. If sufficient space in the shadow tablespaces does not exist, then the database writes a warning to the alert log, and tracks as many data blocks as possible.

See Also:

Oracle Database Administrator's Guide to learn how to manage shadow lost write protection

User Interface for Shadow Tablespaces

You enable and disable shadow lost write protection using the `ALTER PLUGGABLE DATABASE` command.

For shadow lost write protection to protect a specific tablespace or data file, the following conditions must be met:

- You must have enabled shadow lost write protection for the entire PDB by using the `ALTER PLUGGABLE DATABASE ENABLE LOST WRITE PROTECTION` statement.

> **Note:**
>
> In a CDB, if you enable shadow lost write protection in the root, then the PDBs do not inherit it. You must enable shadow lost write protection for every PDB that you want to protect.

- You must have enabled shadow lost write protection for the tablespace or data file to be protected by using the `ENABLE LOST WRITE PROTECTION` clause.

  When you enable shadow lost write protection for a tablespace, all of the data files of the tablespace are protected, and any data files added to the tablespace are also protected. Note that you cannot enable lost write protection on temporary tablespaces or another lost write tablespace.
- You must have created one or more shadow tablespaces by using the `CREATE BIGFILE TABLESPACE` statement with the `LOST WRITE PROTECTION` clause.

Oracle Database assigns a tracked data file to a specific shadow tablespace automatically. You cannot specify which shadow tablespace is used for a particular data file.

The following data dictionary views monitor shadow tablespaces:

- `DBA_TABLESPACES`

  Shows which tablespaces are shadow tablespaces by querying.
- `DBA_DATA_FILES.LOST_WRITE_PROTECT`

  Shows whether lost write protections is enabled for a data file
- `USER_TABLESPACES.LOST_WRITE_PROTECT`

  Shows whether lost write protection is turned on for a specific tablespace. `DBA_DATA_FILES` does not indicate whether lost write is turned on for a tablespace: you must look at `USER_TABLESPACES` instead.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* to learn how to manage lost write protection with shadow tablespaces
> - *Oracle Database SQL Language Reference* to learn about the `CREATE TABLESPACE` statement
> - *Oracle Database Reference* to learn about `DBA_TABLESPACES`

Example: Configuring Lost Write Protection

This example enables shadow lost write tracking for a set of tablespaces.

In this example, your goal is to protect the `salestbs` and `hrtbs` tablespaces within a PDB. You also want to protect the `oetbs01.dbf` data file, and only this data file, within the `oetbs` tablespace. You do the following:

1. Log in to the PDB as an administrator.

2. Create a single shadow tablespace as follows:

```
CREATE BIGFILE TABLESPACE shadow_lwp1
  DATAFILE 'shadow_lwp1_df' SIZE 10M LOST WRITE PROTECTION;
```

3. Enable lost write protection for the PDB as follows:

```
ALTER DATABASE ENABLE LOST WRITE PROTECTION;
```

4. Enable shadow lost write protection for the `salestbs` and `hrtbs` tablespaces as follows:

```
ALTER TABLESPACE salestbs ENABLE LOST WRITE PROTECTION;
ALTER TABLESPACE hrtbs ENABLE LOST WRITE PROTECTION;
```

5. Enable shadow lost write protection for the `oetbs01.dbf` data file as follows:

```
ALTER DATABASE DATAFILE 'oetbs01.dbf' ENABLE LOST WRITE PROTECTION;
```

See Also:

- *Oracle Database Administrator's Guide* to learn how to manage shadow tablespaces
- *Oracle Database SQL Language Reference* to learn about the `CREATE TABLESPACE` statement

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. A **temp file** stores temporary tablespace data.

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.

- Shared and Local Temporary Tablespaces
  Temporary tablespaces are either shared or local.
- Default Temporary Tablespaces
  Every database user account is assigned a default shared temporary tablespace. If the database contains local temporary tablespaces, then every user account is also assigned default local temporary storage.

Shared and Local Temporary Tablespaces

Temporary tablespaces are either shared or local.

A shared temporary tablespace stores temp files on shared disk, so that the temporary space is accessible to all database instances. In contrast, a local temporary tablespace stores separate, non-shared temp files for every database instance. Local temporary tablespaces are useful for Oracle Real Application Clusters or Oracle Flex Clusters.

You can create local temporary tablespaces for both read-only and read/write database instances. When many read-only instances access a single database, local temporary tablespaces can improve performance for queries that involve sorts, hash aggregations, and joins. The advantages are:

- Improving I/O performance by using local rather than shared disk storage
- Avoiding expensive cross-instance temporary space management
- Improving instance startup performance by eliminating on-disk space metadata management

The following table compares the characteristics of shared and local temporary tablespaces.

Table 15-2 Shared and Local Temporary Tablespaces

Shared Temporary Tablespace	Local Temporary Tablespace
Created with the `CREATE TEMPORARY TABLESPACE` statement.	Created with the `CREATE LOCAL TEMPORARY TABLESPACE` statement. **Note:** A local temporary tablespaces is always a bigfile tablespace, but the `BIGFILE` keyword is not required in the creation statement.
Creates a single temporary tablespace for the database.	Creates separate temporary tablespaces for every database instance. The `FOR LEAF` option creates tablespaces only for read-only instances. The `FOR ALL` option creates tablespaces for all instances, both read-only and read/write.
Supports tablespace groups.	Does not support tablespace groups.
Stores temp file metadata in the control file.	Stores temp file metadata common to all instances in the control file, and instance-specific metadata (for example, the bitmaps for allocation, current temp file sizes, and file status) in the SGA.

See Also:

"Introduction to the Oracle Database Instance"

Default Temporary Tablespaces

Every database user account is assigned a default shared temporary tablespace. If the database contains local temporary tablespaces, then every user account is also assigned default local temporary storage.

You can specify a different temporary tablespace for a user account with the `CREATE USER` or `ALTER USER` statements. Oracle Database use the system-level default temporary tablespace for users for whom you do not specify a different temporary tablespace.

- Creation of Default Temporary Tablespaces
  When creating a database, the default temporary storage depends on whether the `SYSTEM` tablespace is locally managed.

- Access to Temporary Storage
  If a user has a temporary tablespace assigned, then the database accesses it first; otherwise, the database accesses the default temporary tablespace. After the database accesses a temporary tablespace for a query, it does not switch to a different one.

> **See Also:**
>
> *Oracle Database SQL Language Reference* to learn more about the `CREATE USER` statement

Creation of Default Temporary Tablespaces

When creating a database, the default temporary storage depends on whether the `SYSTEM` tablespace is locally managed.

The following table shows how Oracle Database chooses default temporary tablespaces at database creation.

Table 15-3 Creation of Default Temporary Tablespaces

Is the SYSTEM tablespace locally managed?	Does the CREATE DATABASE statement specify a default temporary tablespace?	Then the database ...
Yes	Yes	Uses the specified tablespace as the default.
Yes	No	Creates a temporary tablespace.
No	Yes	Uses the specified tablespace as the default.
No	No	Uses `SYSTEM` for default temporary storage. The database writes a warning in the alert log saying that a default temporary tablespace is recommended.

After database creation, you can change the default temporary tablespace for the database with the `ALTER DATABASE DEFAULT TEMPORARY TABLESPACE` statement.

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

> **See Also:**
>
> - "Permanent and Temporary Data Files"
> - *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`

Access to Temporary Storage

If a user has a temporary tablespace assigned, then the database accesses it first; otherwise, the database accesses the default temporary tablespace. After the database accesses a temporary tablespace for a query, it does not switch to a different one.

A user query can access either shared or local temporary storage. Furthermore, a user could have one default local temporary tablespace assigned for read-only instances, and a different default local temporary tablespace assigned for read/write instances.

For read/write instances, the database gives higher priority to shared temporary tablespaces. For read-only instances, the database gives higher priority to local temporary tablespaces. If the database instance is read/write, then the database searches for space in the following order:

1. Is a shared temporary tablespace assigned to the user?
2. Is a local temporary tablespace assigned to the user?
3. Does the database default temporary tablespace have space?

If the answer to any preceding question is yes, then the database stops the search and allocates space from the specified tablespace; otherwise, space is allocated from the database default local temporary tablespace.

If the database instance is read-only, then the database searches for space in the following order:

1. Is a local temporary tablespace assigned to the user?
2. Does the database default local temporary tablespace assigned have space?
3. Is a shared temporary tablespace assigned to the user?

If the answer to any preceding questions is yes, then the database stops the search and allocates space from the specified tablespace; otherwise, space is allocated from the database default shared temporary tablespace.

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.
- Online and Offline Tablespaces
  A tablespace can be online (accessible) or offline (not accessible) whenever the database is open.

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"
> - "Database Writer Process (DBW)"
> - *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 tablespace can do the following:
  - Increase the storage capacity of a database

    The maximum number of data files in a database is limited, 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.

  Starting with Oracle Database 23ai, the SYSTEM, SYSAUX, and USER tablespaces are created as bigfile tablespaces by default. A database upgraded from a previous release retains its tablespace type.

See Also:

- *Oracle Database Backup and Recovery User's Guide* to learn more about backup and recovery
- *Oracle Database Administrator's Guide* to learn how to manage bigfile tablespaces