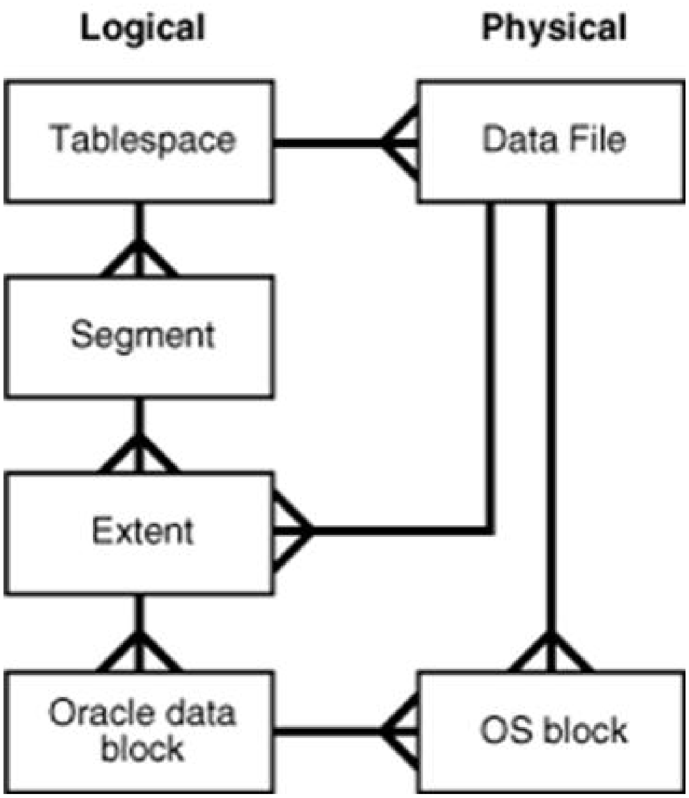
module9-storage.htm; updated June 13, 2013; Some figures shown in these notes are from Oracle document D11321GC11. Reference: Oracle® Database Concepts 11*g* Release 2 (11.2) E25789-01

Module 9 – Storage Structures

Objectives

- Learn the physical structures of a database including segments, extents and data blocks.
- Learn the different segment types.
- Learn about control block space usage.

Logical versus Physical Structures



Segment Types

Objects in an Oracle database such as tables, indexes, clusters, sequences, etc., are comprised of **segments**. There are several different types of segments.

Table: Data are stored in tables. When a table is created with the CREATE TABLE command, a **table segment** is allocated to the new object.

- Table segments do not store table rows in any particular order.
- Table segments do not store data that is clustered or partitioned.

- The DBA has almost no control over the location of rows in a table.
- The segment belongs to a single tablespace.

Table Partition: If a table has high concurrent usage, that is simultaneous access by many different system users as would be the case for a **SALES_ORDER** table in an online-transaction processing environment, you will be concerned with scalability and availability of information as the DBA. This may lead you to create a table that is partitioned into more than one **table partition** segment.

- A partitioned table has a separate segment for each partition.
- Each partition may reside in a different tablespace.
- Each partition may have different storage parameters.
- The Oracle Enterprise Edition must have the partitioning option installed in order to create a partitioned table.

Cluster: Rows in a cluster segment are stored based on **key value columns**. Clustering is sometimes used where two tables are related in a strong-weak entity relationship.

- A cluster may contain rows from two or more tables.
- All of the tables in a cluster belong to the same segment and have the same storage parameters.
- Clustered table rows can be accessed by either a hashing algorithm or by indexing.

Index: When an index is created as part of the **CREATE TABLE** or **CREATE INDEX** command, an index segment is created.

- Tables may have more than one index, and each index has its own segment.
- Each index segment has a single purpose – to speed up the process of locating rows in a table or cluster.

Index-Organized Table: This special type of table has data stored within the index based on primary key values. All data is retrievable directly from the index structure (a tree structure).

Index Partition: Just as a table can be partitioned, so can an index. The purpose of using a partitioned index is to minimize contention for the I/O path by spreading index input-output across more than one I/O path.

- Each partition can be in a different tablespace.
- The partitioning option of Oracle Enterprise Edition must be installed.

Undo: An undo segment is used to store "before images" of data or index blocks prior to changes being made during transaction processing. This allows a rollback using the before image information.

Temporary: Temporary segments are created when commands and clauses such as **CREATE INDEX**, **SELECT DISTINCT**, **GROUP BY**, and **ORDER BY** cause Oracle to perform memory sort operations.

- Often sort actions require more memory than is available.
- When this occurs, intermediate results of sort actions are written to disk so that the sort operation can continue – this allows information to swap in and out of memory by

writing/reading to/from disk.

- Temporary segments store intermediate sort results.

LOB: Large objects can be stored as one or more columns in a table. Large objects (LOBs) include images, separate text documents, video, sound files, etc.

- These LOBs are not stored in the table – they are stored as separate segment objects.
- The table with the column actually has a "**pointer**" value stored in the column that points to the location of the LOB.

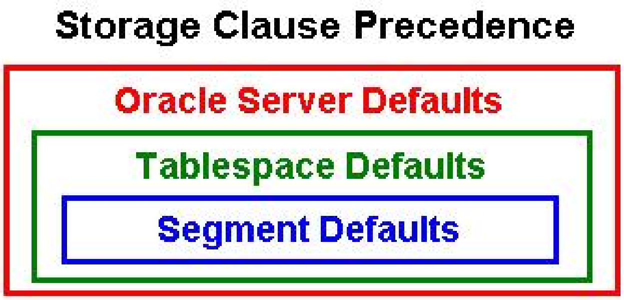
Nested Table: A column in one table may consist of another table definition. The inner table is called a "nested table" and is stored as a separate segment. This would be done for a **SALES_ORDER** table that has the **SALES_DETAILS** (order line rows) stored as a nested table.

Bootstrap Segment: This is a special cache segment created by the sql.bsq script that runs when a database is created.

- It stores initial data dictionary cache information when a database is opened.
- This segment cannot be queried or updated and requires no DBA maintenance.

Storage Clauses/Parameters

When database objects are created, the object always has a set of storage parameters. This figure shows three ways that an object can obtain storage clause parameters.



Tablespaces have space managed depending on the type of tablespace:

- **Locally Managed Tablespaces** – use **bitmaps** to track used and free space – Locally managed is the default for non-SYSTEM permanent tablespaces when the type of extent management is not specified at the time a tablespace is created.
    - Tablespace extents for Locally Managed are either (1) Uniform specified with the **UNIFORM** clause or (2) variable extent sizes determined by the system with the **AUTOALLOCATE** clause.
        - **Uniform**:
            - Specify an extent size or use the **1MB** default size.
            - Each extent contains at least **5** database blocks.
        - **System Managed**:
            - Oracle determines optimal size of additional extents with a minimum extent size of 64KB.

  - With **SEGMENT SPACE MANAGEMENT AUTO**, the minimum extent size is **1MB** if the Database Block size is **16K** or larger.
- **Dictionary Managed Tablespaces** – tables in the data dictionary track space utilization.

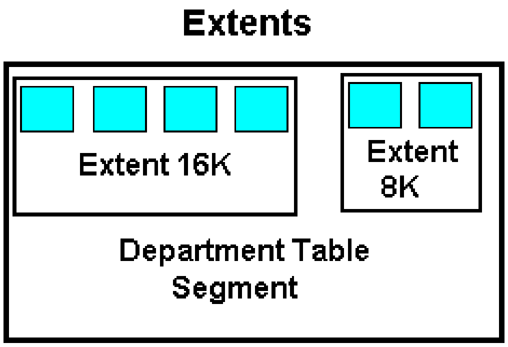
Facts about storage parameters:

- Segment storage parameters can override the tablespace level defaults with the exception of two parameters. You cannot override the **MINIMUM EXTENT** or **UNIFORM SIZE** tablespace parameters.
- If you do not specify segment storage parameters, then a segment will inherit the tablespace default parameters.
- If tablespace default storage parameters are not set, the Oracle server system default parameters are used.
- Locally managed tablespaces **cannot** have the storage parameters **INITIAL**, **NEXT**, **PCTINCREASE**, and **MINEXTENTS** specified; however, these parameters **can be** specified at the segment level.
- When storage parameters of a segment are modified, the modification only applies to extents that are allocated after the modification takes place.

Extents

Extents are allocated in chunks that are **not** necessarily uniform in size, but the space allocated is contiguous on the disk drive as is shown in this figure.

- When a database object such as a table grows, additional disk space is allocated to its segment of the tablespace in the form of an extent.
- This figure shows two extents of different sizes for the **Department** table segment.



In order to develop an understanding of extent allocation to segments, review this **CREATE TABLESPACE** command.

```
CREATE TABLESPACE data
DATAFILE '/u01/student/dbockstd/oradata/USER350data01.dbf'
    SIZE 20M
EXTENT MANAGEMENT LOCAL UNIFORM SIZE 40K;
```

- The above specifies use of local management for extents.
- The default size for all extents is specified through the **UNIFORM SIZE** parameter as **40K**.
- Since this parameter cannot be overridden, all segments in this tablespace will be allocated with segments that are **40K** in size.

This next **CREATE TABLESPACE** command creates a dictionary managed tablespace.

Note: ***You will not be able to execute this command for your database as dictionary managed tablespaces are not allowed with Oracle 11g.***

- Here the **DEFAULT STORAGE** parameter is used to specify the size of extents allocated to segments created within the tablespace.
- These parameters **can be** overridden by parameter specifications in the object creation command, for example, a CREATE TABLE command.

```
CREATE TABLESPACE data
DATAFILE '/u01/student/dbockstd/oradata/USER350data01.dbf' SIZE 20M
EXTENT MANAGEMENT DICTIONARY
DEFAULT STORAGE (
    INITIAL 128K
    NEXT 40K
    PCTINCREASE 50
    MINEXTENTS 1
    MAXEXTENTS 999);
```

- **INITIAL** specifies the initial extent size (the first extent allocated).
  - A size that is too large here can cause failure of the database if there is not any area on the disk drive with sufficient contiguous disk space to satisfy the INITIAL parameter.
  - When a database is built to store information from an older system that is being converted to Oracle, a DBA may have some information about how large initial extents need to be in general and may specify a larger size as is done here at **128K**.
- **NEXT** specifies the size of the next extent (2nd, 3rd, etc).
  - This is termed an **incremental extent**.
  - This can also cause failure if the size is too large.
  - Usually a smaller value is used, but if the value is too small, segment fragmentation can result.
  - This must be monitored periodically by a DBA which is why dictionary managed tablespaces are **NOT** preferred.
- **PCTINCREASE** can be very troublesome.
  - If you set this very high, e.g. **50%** as is shown here, the segment extent size can increase by **7,655%** over just **10** extents.
  - Best solution: a single **INITIAL** extent of the correct size followed by a small value for **NEXT** and a value of **0** (or a small value such as **5**) for **PCTINCREASE**.

Use smaller default **INITIAL** and **NEXT** values for a dictionary-managed tablespace's default storage clauses as these defaults can be over-ridden during the creation of individual objects (tables, indexes, etc.) where the **STORAGE** clause is used in creating the individual objects.

- **MINEXTENTS** and **MAXEXTENTS** parameters specify the minimum and maximum number of extents allocated by default to segments that are part of the tablespace.

The default storage parameters can be overridden when a segment is created as is illustrated in this next section.

Example of a CREATE TABLE Command

This shows the creation of a table named **Orders** in the **Data01** tablespace.

- **Data01** is locally managed.
- The storage parameters specified here override the storage parameters for the **Data01** tablespace.

```
CREATE TABLE Orders (
    Order_Id        NUMBER(3) PRIMARY KEY,
    Order_Ddate     DATE DEFAULT (SYSDATE),
    Ship_Date       DATE,
    Client          VARCHAR(3) NOT NULL,
    Amount_Due      NUMBER(10,2),
    Amount_Paid     NUMBER(10,2) )
PCTFREE 5 PCTUSED 65
STORAGE (
    INITIAL 48K
    NEXT 48K
    PCTINCREASE 5
    MINEXTENTS 1
    MAXEXTENTS UNLIMITED )
TABLESPACE Data01;
```

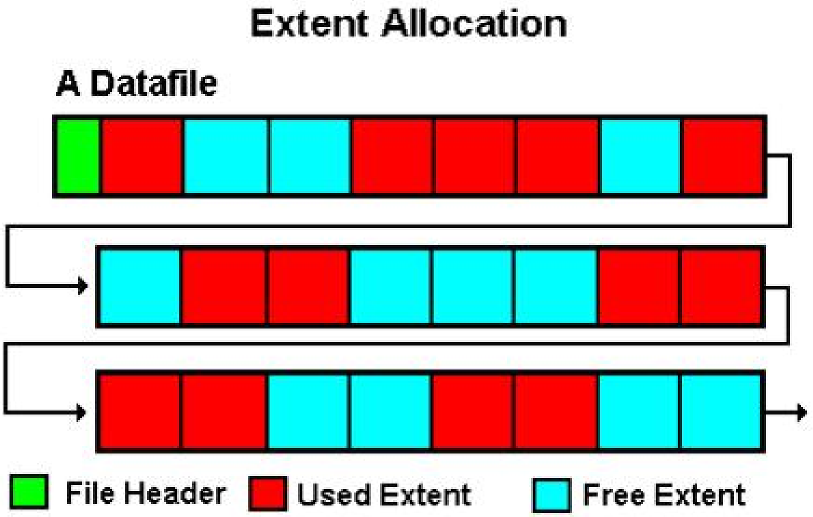
Allocation/Deallocation: When a tablespace is initially created, the first datafile (and subsequent datafiles) created to store the tablespace has a header which may be one or more blocks at the beginning of the file as is shown in the figure below.

- As segments are created, extended, or altered **free extents** are allocated.
- The below figure shows that extents can vary in size.
- This figure represents a **Locally Managed** tablespace where the Locally Managed tablespace's extent size is specified by the `EXTENT MANAGEMENT LOCAL AUTOALLOCATE` clause—recall that **AUTOALLOCATE** enables Oracle to decide the appropriate extent size for a segment. In an older Oracle database, it could also represent a **Dictionary Managed** tablespace.
- As segments are dropped, altered, or truncated, extents are released to become free extents available for reallocation.
- The first extent is allocated to a segment, even though the data blocks may be empty.

- Oracle formats the blocks for an extent only as they are used - they can actually contain old data.
- Extents for a segment must always be in the same tablespace, but can be in different datafiles.
- The first data block of every segment contains a directory of the extents in the segment.
- If you delete data from a segment, the extents/blocks are not returned to the tablespace for reuse. Deallocation occurs when:
    - You DROP a segment.
    - You use an **online segment shrink** to reclaim fragmented space in a segment.

```
ALTER TABLE employees ENABLE ROW MOVEMENT;
ALTER TABLE employees SHRINK SPACE CASCADE;
```

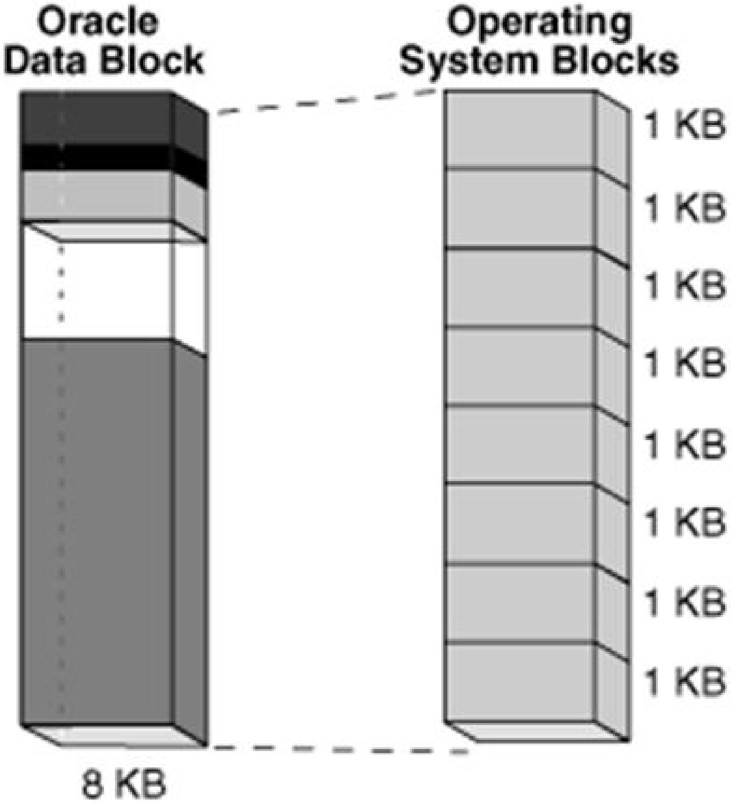
-
    - You can rebuild or coalesce an index segment.
    - You **truncate** a table or table cluster, which removes all rows.
- Over time, segments in a tablespace's datafiles can become fragmented due to the addition of extents as is shown in this figure.

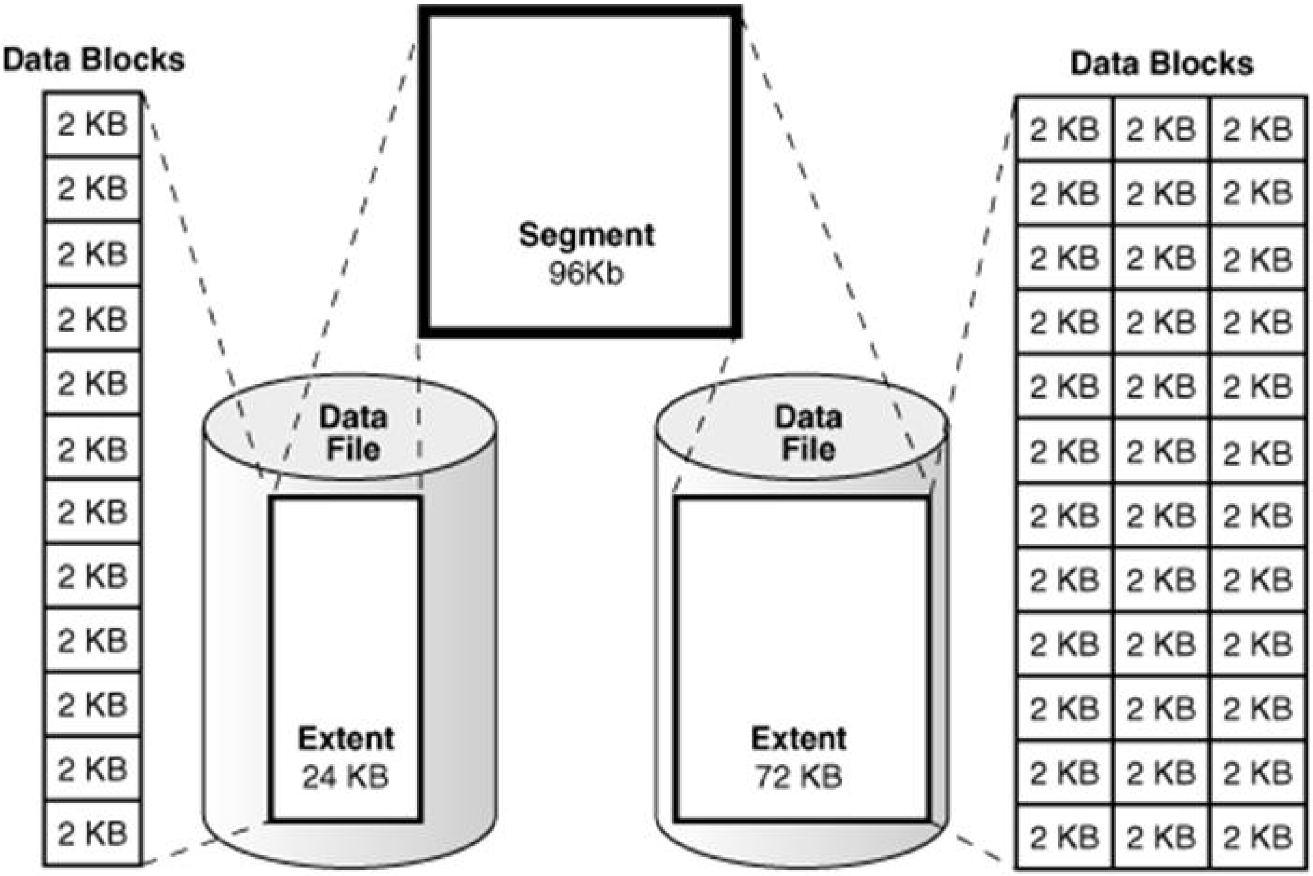


Database Block

The **Database Block** or simply **Data Block**, as you have learned, is the smallest size unit for input/output from/to disk in an Oracle database.

- A data block may be equal to an operating system block in terms of size, or may be larger in size, and should be a **multiple** of the operating system block.
- The **DB_BLOCK_SIZE** parameter sets the size of a database's standard blocks at the time that a database is created.
- **DB_BLOCK_SIZE** has to be a multiple of the physical block size allowed by the operating system for a server’s storage devices.
- 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 **4KB** or **8KB**.





- Oracle also supports the creation of databases that have more than one block size. This is primarily done when you need to specify tablespaces with different block sizes in order to maximize I/O performance.

- You've already learned that a database can have up to **four** **nonstandard block sizes** specified.
- Block sizes must be sized as a power of two between **2K** and **32K** in size, e.g., **2K**, **4K**, **8K**, **16K**, or **32K**.
- A sub cache of the Database Buffer Cache is configured by Oracle for each nonstandard block size.

Standard Block Size: The **DB_CACHE_SIZE** parameter specifies the size of the **Database Buffer Cache**. However, if **SGA_TARGET** is set and **DB_CACHE_SIZE** is not, then Oracle decides how much memory to allocate to the **Database Buffer Cache**. The minimum size for **DB_CACHE_SIZE** must be specified as follows:

- One granule where a granule is a unit of contiguous virtual memory allocation in RAM.
- If the total System Global Area (**SGA**) based on **SGA_MAX_SIZE** is less than **128MB**, then a granule is **4MB**.
- If the total SGA is greater than **128MB**, then a granule is **16MB**.
- The default value for **DB_CACHE_SIZE** is **48MB** rounded up to the nearest granule size.

Nonstandard Block Size: If a DBA wishes to specify one or more nonstandard block sizes, the parameter following parameters are set.

- The data block sizes should be a multiple of the operating system's block size within the maximum limit to avoid unnecessary I/O.
- Oracle data blocks are the smallest units of storage that Oracle can use or allocate.
- Do not use the specified **DB_BLOCK_SIZE** value to set nonstandard block sizes.
- For example, if the standard block size is **8K**, do not use the **DB_8K_CACHE_SIZE** parameter.
  - **DB_2K_CACHE_SIZE** -- parameter for **2K** nonstandard block sizes.
  - **DB_4K_CACHE_SIZE** -- parameter for **4K** nonstandard block sizes.
  - **DB_8K_CACHE_SIZE** -- parameter for **8K** nonstandard block sizes.
  - **DB_16K_CACHE_SIZE** -- parameter for **16K** nonstandard block sizes.
  - **DB_32K_CACHE_SIZE** -- parameter for **32K** nonstandard block sizes.

Nonstandard Block Size Tablespaces: The **BLOCKSIZE** parameter is used to create a tablespace with a nonstandard block size. Example:

```
CREATE TABLESPACE special_apps
  DATAFILE '/u01/student/dbockstd/oradata/USER350_spec_apps01.dbf'
  SIZE 20M
  BLOCKSIZE 32K;
```

- Here the nonstandard block size specified with the **BLOCKSIZE** clause is **32K**.
- This command will not execute unless the **DB_32K_CACHE_SIZE** parameter has already been specified because buffers of size **32K** must already be allocated in the **Database Buffer Cache** as part of a sub cache.

There are some **additional rules** regarding the use of multiple block sizes:

- If an object is partitioned and resides in more than one tablespace, all of the tablespaces where the object resides must be the same block size.
- Temporary tablespaces must be the standard block size. This also applies to permanent tablespaces that have been specified as default temporary tablespaces for system users.

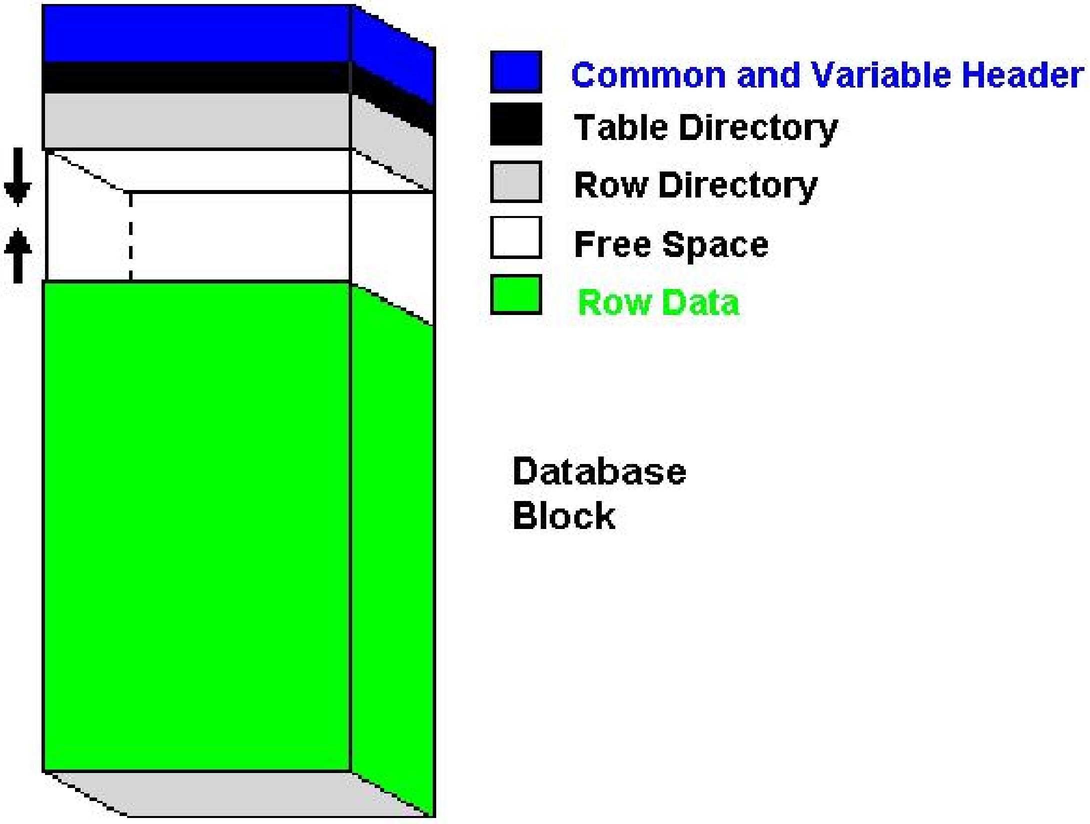
What Block Size To Use?

Use the largest block size available with your operating system for a new database.

- Using a larger database block size should improve almost every performance factor.
- Larger database block sizes keep indexes from splitting levels.
- Larger database block sizes keep more data in memory longer.
- If the database has excessive buffer busy waits (due to a large # of users performing updates and inserts), then increase the **freelists** parameter setting for the table or other busy objects.

Data Block Contents

This figure shows the components of a data block. This is the structure regardless of the type of segment to which the block belongs.



Block header – contains common and variable components including the block address, segment type, and transaction slot information.

- The **block header** also includes the table directory and row directory.

- On average, the fixed and variable portions of block overhead total **84** to **107 bytes**.
- **Table Directory** – used to track the tables to which row data in the block belongs.
  - Data from more than one table may be in a single block if the data are clustered.
  - **The Table Directory is only used if data rows from more than one table are stored in the block, for example, a cluster**.
- **Row Directory** - used to track which rows from a table are in this block.
  - The **Row Directory** includes for each row or row fragment in the row data area.
  - When space is allocated in the **Row Directory** to store information about a row, this space is **not reclaimed** upon deletion of a row, but is reclaimed when new rows are inserted into the block.
  - A block can be empty of rows, but if it once contained rows, then data will be allocated in the **Row Directory** (2 bytes per row) for each row that ever existed in the block.
- **Transaction Slots** are space that is used when transactions are in progress that will modify rows in the block.
- The block header grows from top down.
- **Data space (Row Data)** – stores row data that is inserted from the bottom up.

Free space in the middle of a block can be allocated to either the header or data space, and is contiguous when the block is first allocated.

- Free space is allocated to allow variable character and numeric data to expand and contract as data values in existing rows are modified.
- New rows are also inserted into free space.
- Free space may fragment as rows in the block are modified or deleted.

Oracle (the **SMON** background process) automatically and transparently coalesces the free space of a data block periodically *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. This figure shows before and after coalescing free space.

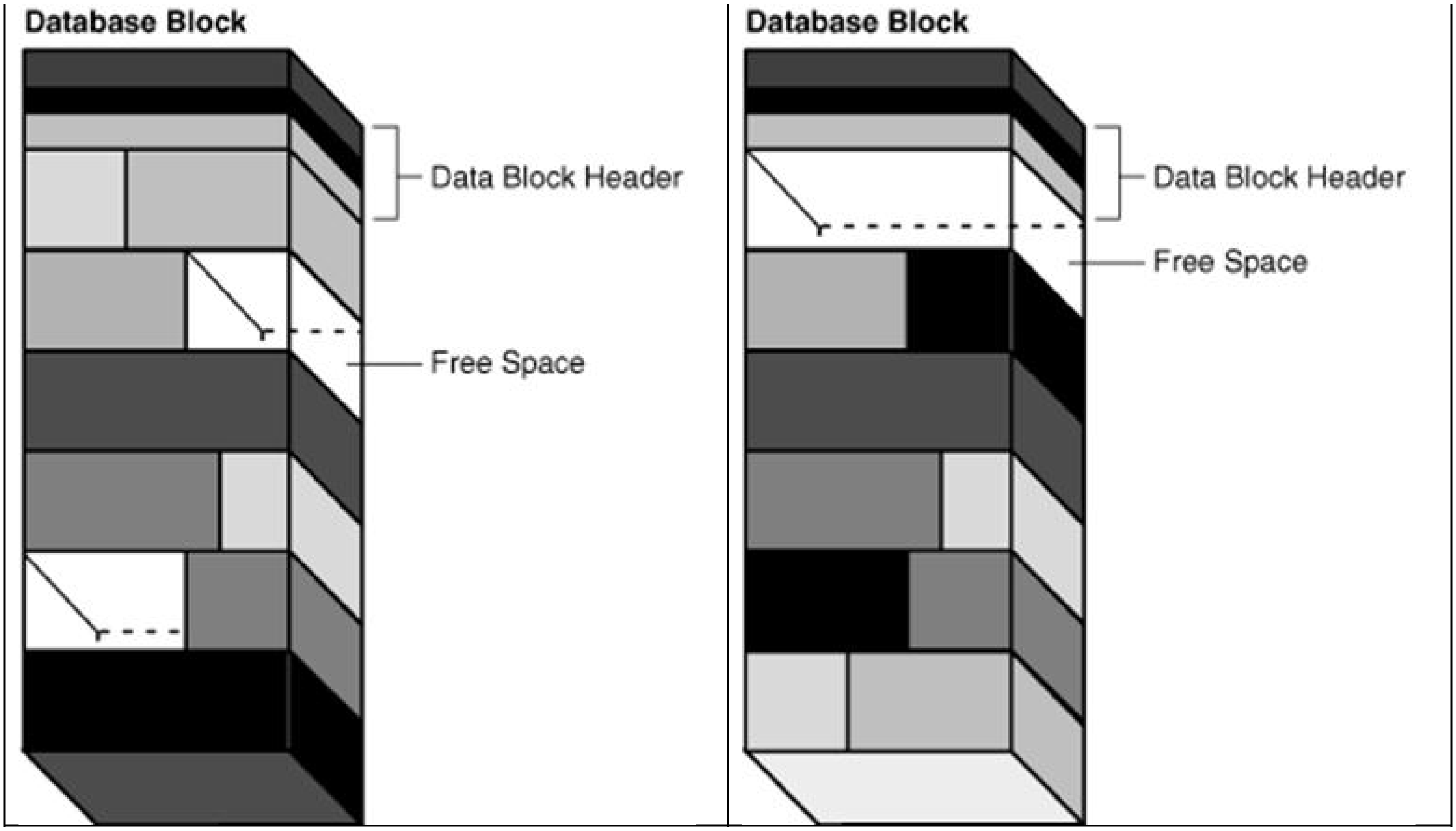
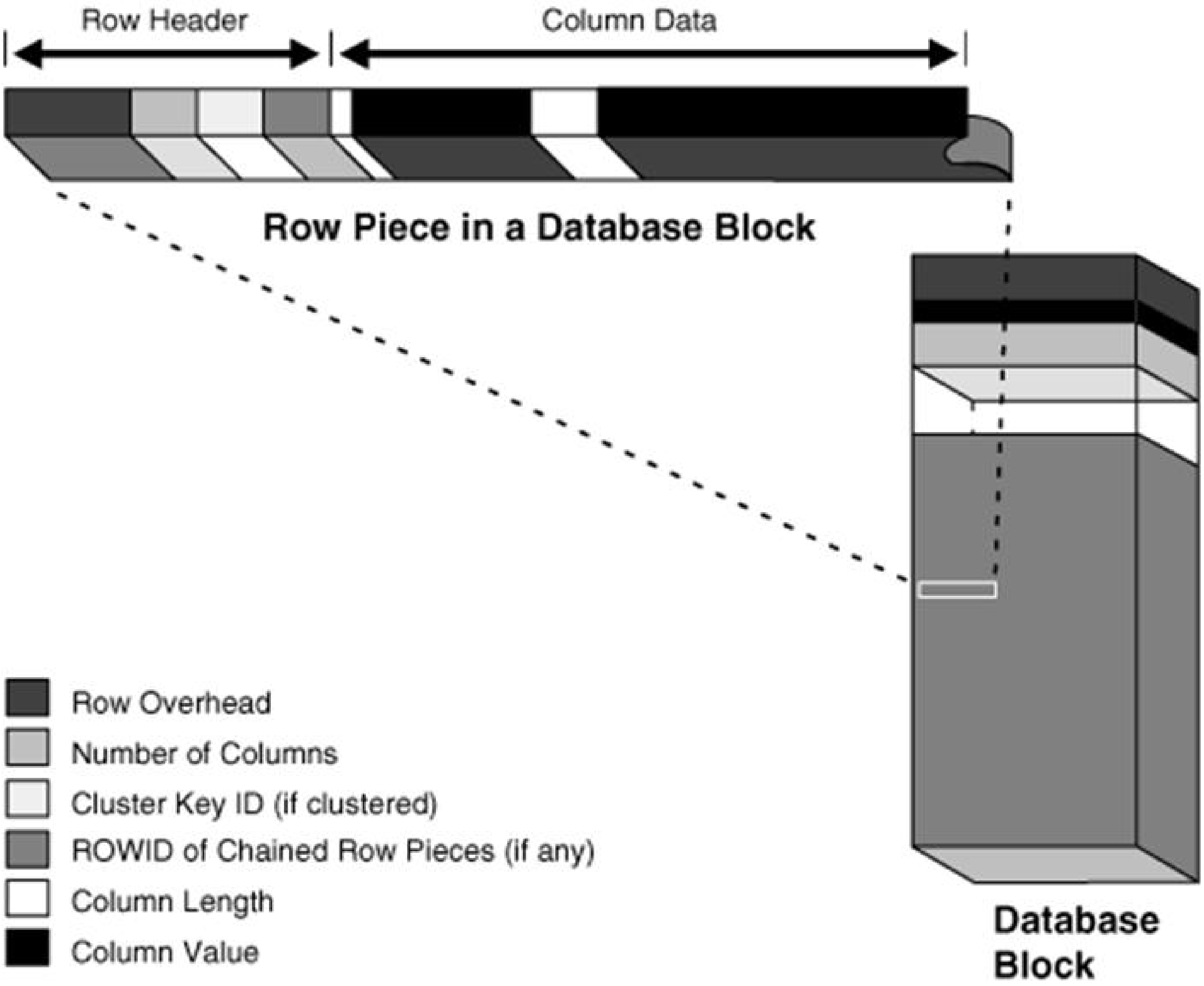
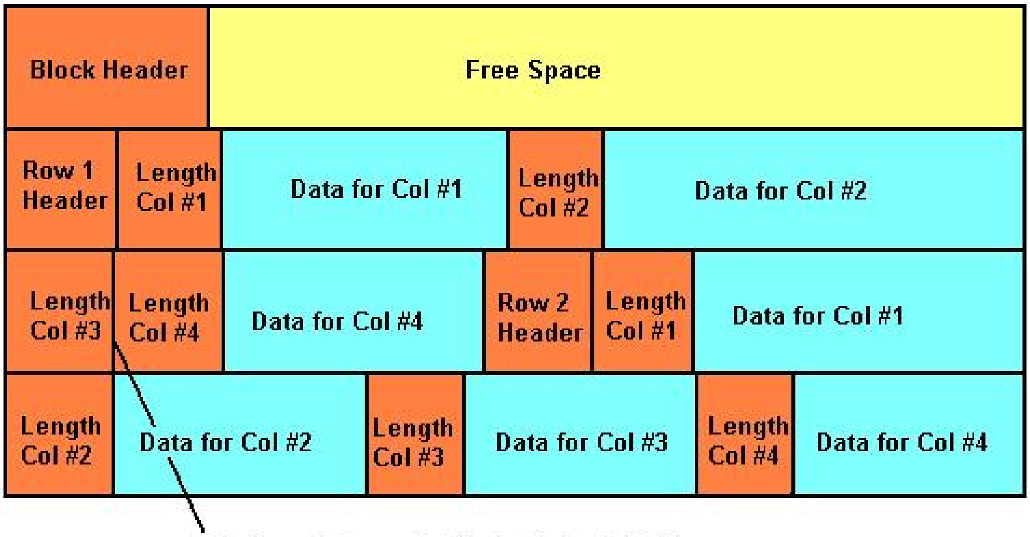


Table Data in a Segment: Table data is stored in the form of rows in a data block.

- The figures below show the block header then the data space (row data) and the free space.
- Each row consists of columns with associated overhead.
- The storage overhead is in the form of "hidden" columns accessible by the DBMS that specify the length of each succeeding column.


Row Header
Column Data
Row Piece in a Database Block
Database Block
Row Overhead
Number of Columns
Cluster Key ID (if clustered)
ROWID of Chained Row Pieces (if any)
Column Length
Column Value



Note: Row 1, Col #3 is NULL

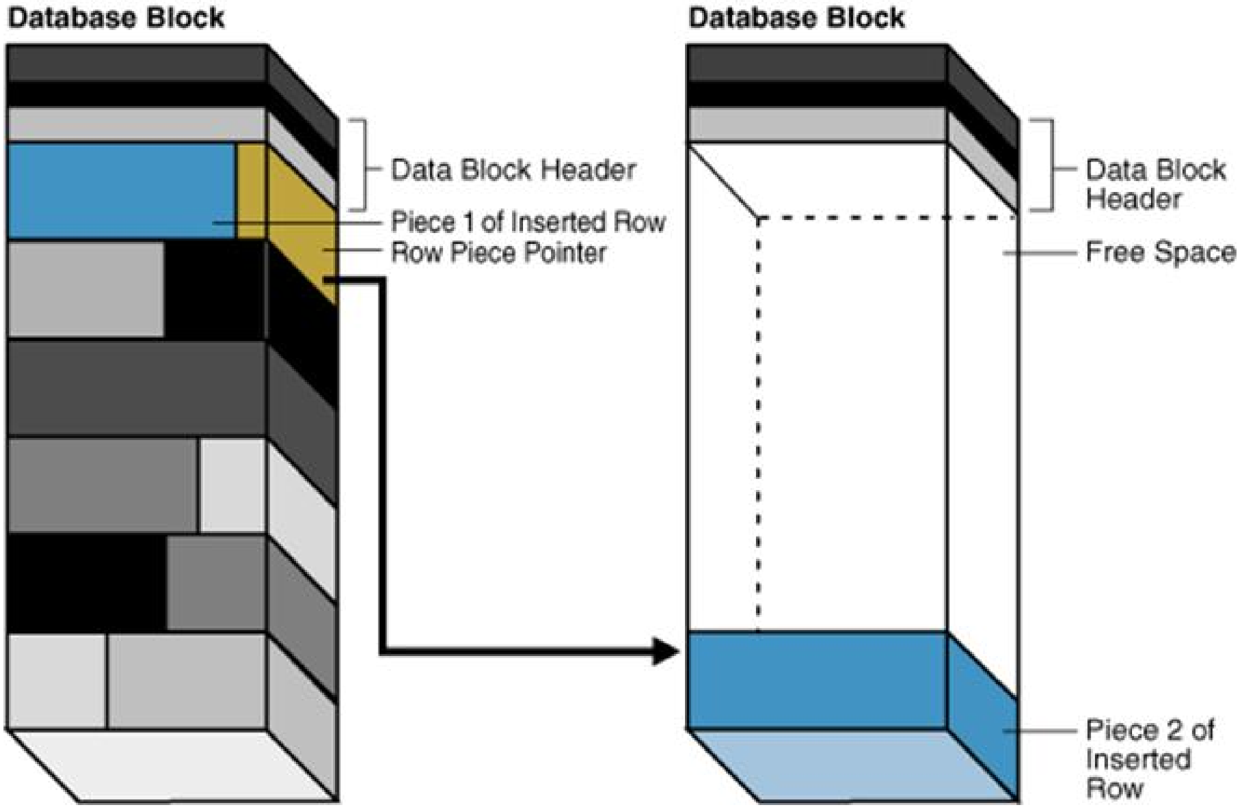
- Rows are stored right next to each other with no spaces in between.
- **Column values** are stored right next to each other in a **variable length** format.

- The length of a field indicates the length of each column value (variable length - Note the Length Column 1, Length Column 2, etc., entries in the figure).
- Column length of **0** indicates a **null** field.
- Trailing null fields are not stored.

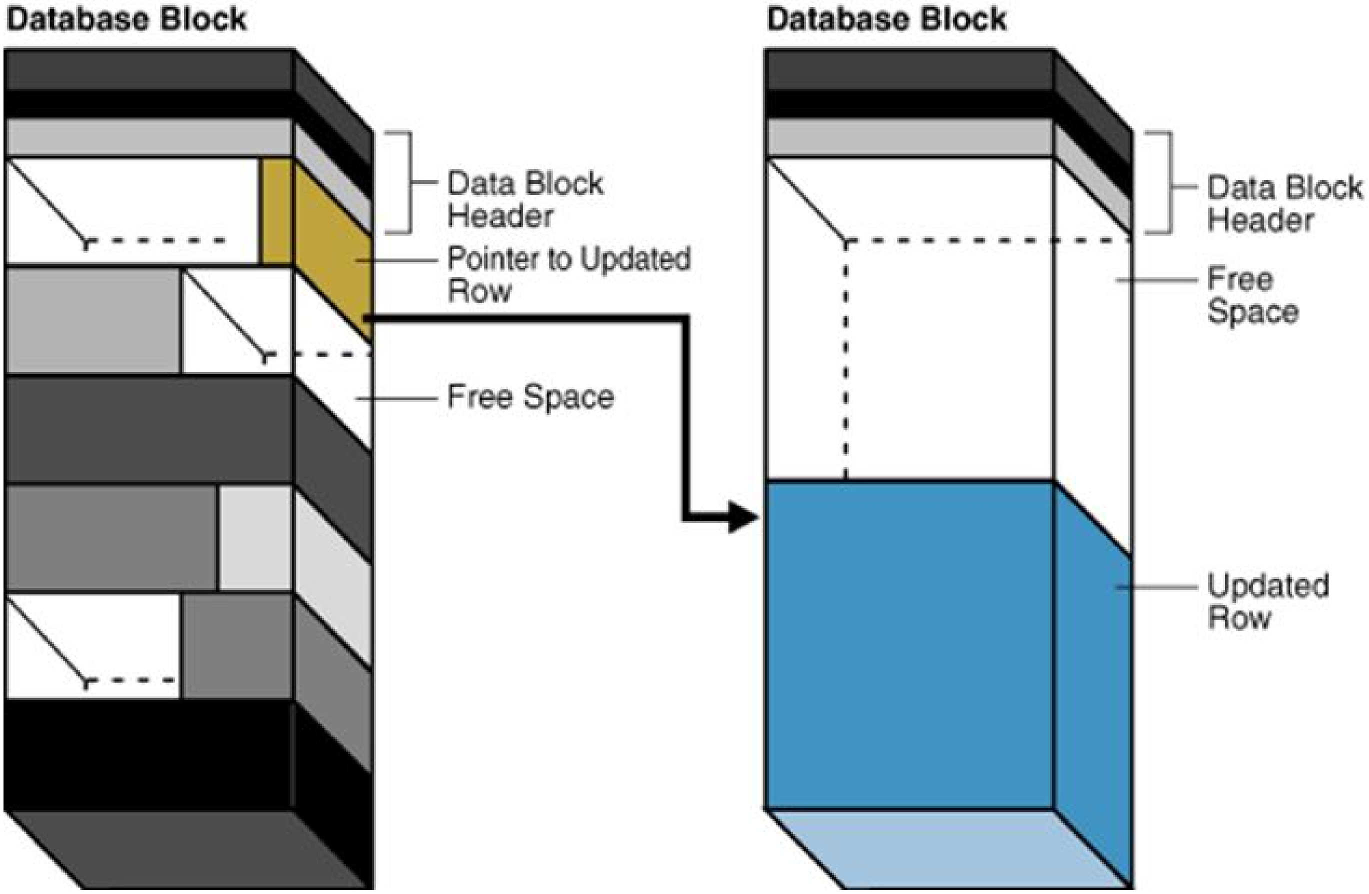
Row Chaining and Migrating

There are two situations where a data row may not fit into a single data block:

- The row is **too large** to fit into one data block when it is first inserted, or the table contains **more than 255 columns** (the maximum for a row piece).
  - In this case, Oracle stores the data for the row in a **chain** of data blocks (one or more) reserved for that segment.
  - Row chaining most often occurs with large rows, such as rows that contain a column of datatype LONG or LONG RAW.
  - Row chaining in these cases is unavoidable.



- A row that originally fit into one data block has one or more columns updated so that the overall row length increases, and the block's free space is already completely filled.
  - In this case, Oracle **migrates** the data for the entire row to a new data block, assuming the entire row can fit in a new block.
  - Oracle preserves the original row piece of a migrated row to point to the new block containing the migrated row.
  - The **rowid** of a migrated row does not change.



When a row is chained or migrated, I/O performance associated with this row **decreases** because Oracle must scan more than one data block to retrieve the information for the row.

Manual Data Block Free Space Management -- Database Block Space Utilization Parameters

Manual data block management requires a DBA to specify how block space is used and when a block is available for new row insertions.

- This is the default method for data block management for **dictionary managed tablespace** objects (another reason for using locally managed tablespaces with UNIFORM extents).
- Database block space utilization parameters are used to control space allocation for data and index segments.

The **INITTRANS** parameter:

- specifies the initial number of **transaction slots** created when a database block is initially allocated to either a data or index segment.
- These slots store information about the transactions that are making changes to the block at a given point in time.
- The amount of space allocated for a transaction slot is **23 bytes**.
- If you set **INITRANS** to **2**, then there are **46 bytes** (2 * 23) pre-allocated in the header, etc.

- These slots are in the **database block header**.
-

The **INITTRANS** parameter:

- specifies a minimum level of concurrent access.
- The default is **1** for a data segment and **2** for an index segment.
- If a DBA specifies **INITTRANS** at 4, for example, this means that 4 transactions can be concurrently making modifications to the database block.
- Also, setting this to a figure that is larger than the default can eliminate the processing overhead that occurs whenever additional transaction slots have to be allocated to a block's header when the number of concurrent transactions exceeds the **INITTRANS** parameter.

The **MAXTRANS** parameter:

- specifies the maximum number of concurrent transactions that can modify rows in a database block.
- Surprisingly, the default maximum is **255**. This value is quite large.
- This parameter is set to guarantee that there is sufficient space in the block to store data or index data.

Example: Suppose a DBA sets **INITTRANS** at **4** and **MAXTRANS** at **10**. Initially, **4** transaction slots are allocated in the block header. If **6** system users process concurrent transactions for a given block, then the number of transaction slots increases by **2** slots to **6** slots. Once this space is allocated in the header, it is not deallocated.

What happens if **11** system users attempt to process concurrent transactions for a given block? The 11th system user is denied access – an Oracle error message is generated – until current transactions complete (either are committed or rolled back).

The PCTFREE and PCTUSED Parameters

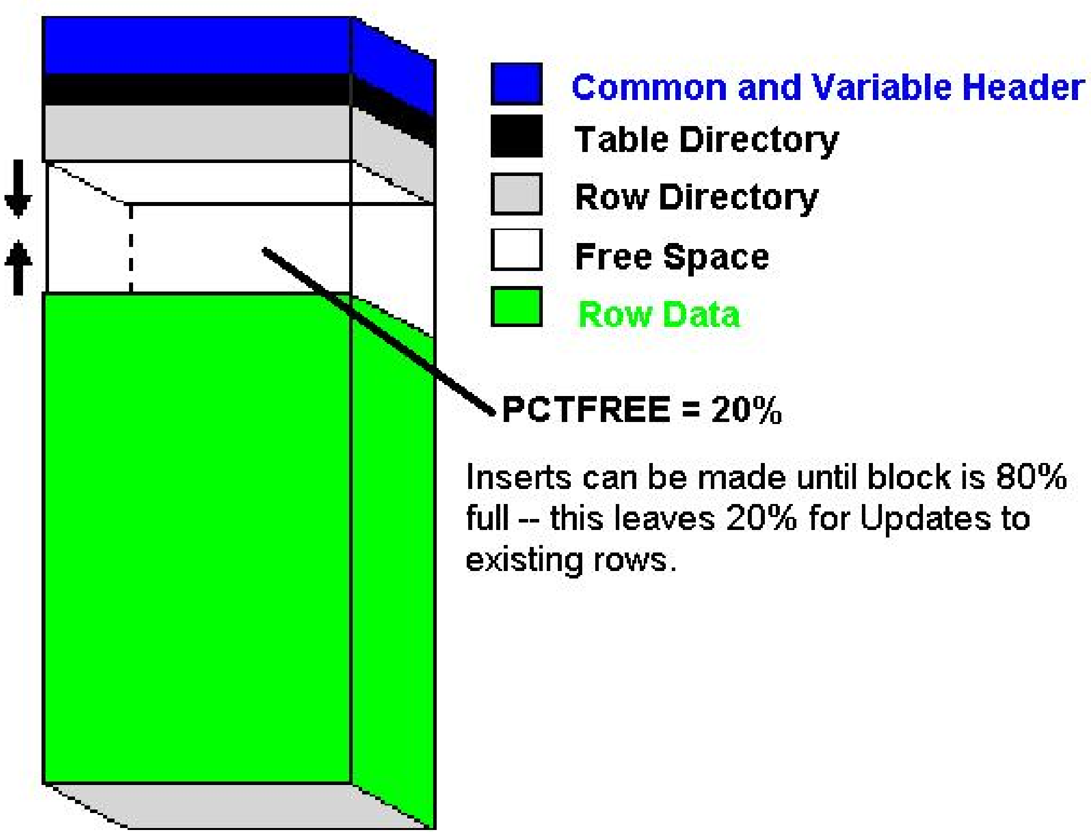
You, as the DBA, must decide how much **Free Space** is needed for data blocks in manual management of data blocks.

You set the free space with the **PCTFREE** and **PCTUSED** parameters at the time that you create an object like a Table or Index.

PCTFREE: The **PCTFREE** parameter is used at the time an object is created to set the percentage of usable block space to be reserved during row insertion for possible later updates to rows in the block.

- PCTFREE is the only space parameter used for **Automatic Segment Space Management.**
- The parameter guarantees that at least **PCTFREE** space is reserved for updates to existing data rows. **PCTFREE** reserves space for growth of existing rows through the modification of data values.
- This figure shows the situation where the **PCTFREE** parameter is set to **20** (20%).

- The default value for **PCTFREE** is **10%.**
- New rows can be added to a data block as long as the amount of space remaining is at or greater than **PCTFREE**.
- After **PCTFREE** is met (this means that there is less space available than the **PCTFREE** setting), Oracle considers the block full and will not insert new rows to the block.



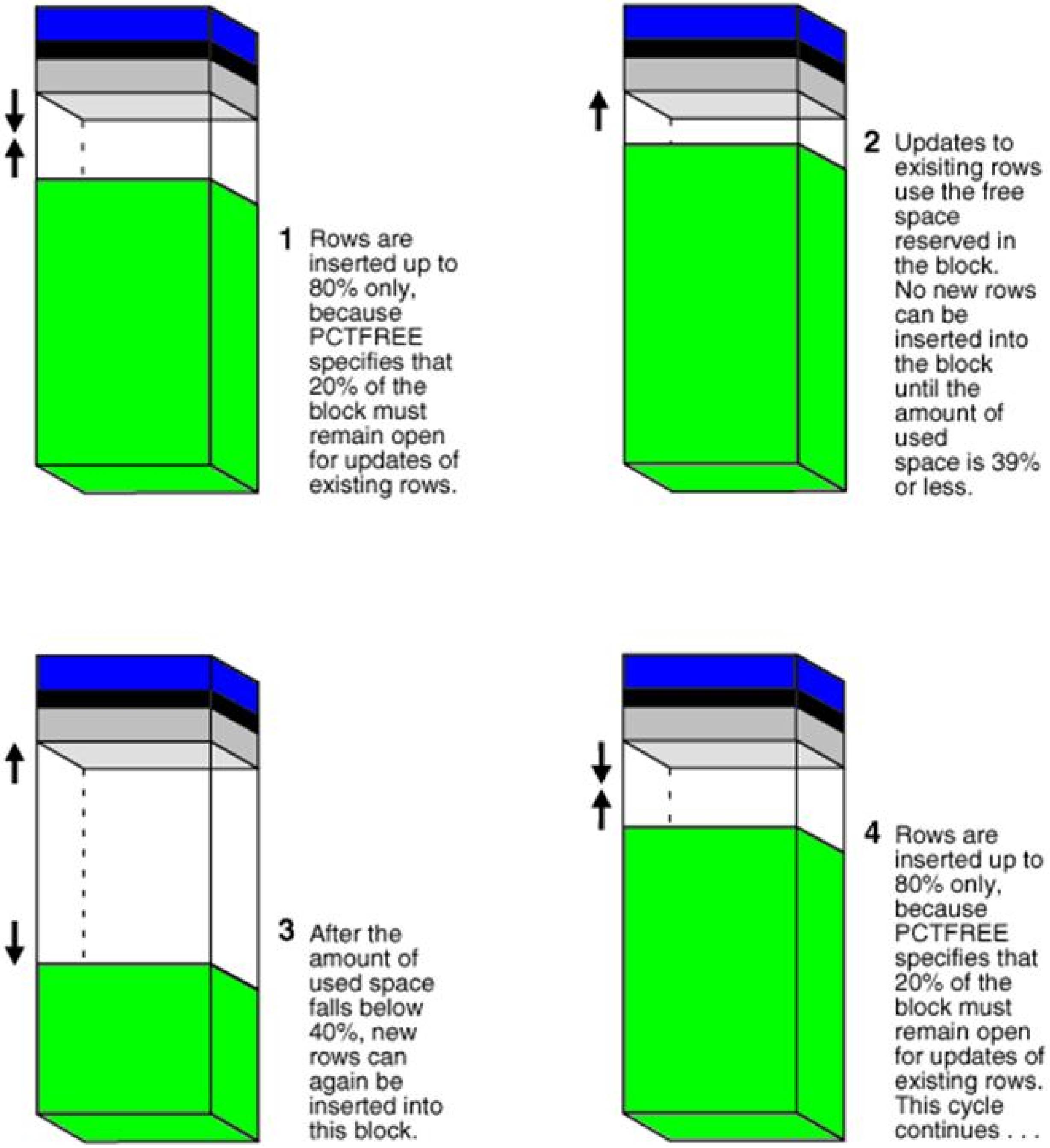
Inserts can be made until block is 80% full -- this leaves 20% for Updates to existing rows.

PCTUSED: The parameter **PCTUSED** is used to set the level at which a block can again be considered by Oracle for insertion of new rows. It is like a low water mark whereas **PCTFREE** is a high water mark. The PCTUSED parameter sets the minimum percentage of a block that can be used for row data plus overhead before new rows are added to the block.

- After a data block is filled to the limit determined by PCTFREE, Oracle Database considers the block unavailable for the insertion of new rows until the percentage of that block falls beneath the parameter PCTUSED.
- As free space grows (the space allocated to rows in a database block **decreases** due to deletions or updates), the block can again have new rows inserted but only if the percentage of the data block in use falls below **PCTUSED**.
- Example: if **PCTUSED** is set at **40**, once **PCTFREE** is hit, the percentage of block space used **must** drop to **39%** or less before row insertions are again made.
- The system default for **PCTUSED** is **40**.
- Oracle tries to keep a data block at least **PCTUSED** full before using new blocks.
- The **PCTUSED** parameter is not set when **Automatic Segment Space Management** is enabled. This parameter only applies when **Manual Segment Space Management** is in use.

This figure depicts the situation where **PCTUSED** is set to 4**0** and **PCTFREE** is set to **20** (40% and 20% respectively).

Data Block
PCTFREE = 20, PCTUSED = 40



1 Rows are inserted up to 80% only, because PCTFREE specifies that 20% of the block must remain open for updates of existing rows.

2 Updates to exisiting rows use the free space reserved in the block. No new rows can be inserted into the block until the amount of used space is 39% or less.

3 After the amount of used space falls below 40%, new rows can again be inserted into this block.

4 Rows are inserted up to 80% only, because PCTFREE specifies that 20% of the block must remain open for updates of existing rows. This cycle continues . . .

Both **PCTFREE** and **PCTUSED** are calculated as percentages of the available data space – Oracle deducts the space allocated to the block header from the total block size when computing these parameters.

Generally **PCTUSED** plus **PCTFREE** should add up to **80**. The sum of **PCTFREE** and **PCTUSED** cannot exceed **100**. If **PCTFREE** is **20**, and **PCTUSED** is **60**, this will ensure at least **60%** of each block is used while saving **20%** for row updates.

Effects of PCTFREE and PCTUSED:

A high **PCTFREE** has these effects:

- There is a lot of space for the growth of existing rows in a data block.
- Performance is improved since data blocks do **not** need to be reorganized very frequently.
- Performance is improved because chaining is reduced.
- Storage space within a data block may not be used efficiently as there is always some empty space in the data blocks.

A low **PCTFREE** has these effects (basically the opposite effect of high **PCTFREE**):

- There is less space for growth of existing rows.
- Performance may suffer due to the need to reorganize data in data blocks more frequently:
  - Oracle may need to **migrate** a row that will no longer fit into a data block due to modification of data within the row.
  - If the row will no longer fit into a single database block, as may be the case for very large rows, then database blocks are **chained** together logically with pointers. This also causes a performance hit. This may also cause a DBA to consider the use of a nonstandard block size. In these situations, I/O performance will degrade.
  - Examine the extent of chaining or migrating with the **ANALYZE** command. You may resolve row chaining and migration by exporting the object (table), dropping the object, and then importing the object.
- Chaining may increase resulting in additional Input/Output operations.
- Very little storage space within a data block is wasted.

A high **PCTUSED** has these effects:

- Decreases performance because data blocks may experience more migrated and chained rows.
- Reduces wasted storage space by filling each data block more fully.

A low **PCTUSED** has these effects:

- Performance improves due to a probable decrease in migrated and chained rows.
- Storage space usage is not as efficient due to more unused space in data blocks.

Guidelines for setting PCTFREE and PCTUSED:

If data for an object tends to be fairly stable (doesn't change in value very much), not much free space is needed (as little as **5%**). If changes occur extremely often and data values are very volatile, you may need as much as **40%** free space. Once this parameter is set, it cannot be changed without at least partially recreating the object affected.

- **Update activity with high row growth** – the application uses tables that are frequently updated affecting row size – set **PCTFREE** moderately high and **PCTUSED** moderately low to allow for space for row growth.

  PCTFREE = 20 to 25

PCTUSED = 35 to 40
(100 – PCTFREE) – PCTUSED = 35 to 45

- **Insert activity with low row growth** – the application has more insertions of new rows with very little modification of existing rows – set **PCTFREE** low and **PCTUSED** at a moderate level. This will avoid row chaining. Each data block has its space well utilized but once new row insertion stops, there are no more row insertions until there is a lot of storage space again available in a data blocks to minimize migration and chaining.
  PCTFREE = 5 to 10
  PCTUSED = 50 to 60
  (100 – PCTFREE) – PCTUSED = 30 to 45

- **Performance primary importance and disk space is readily available** – when disk space is abundant and performance is the critical issue, a DBA must ensure minimal migration or chaining occurs by using very high **PCTFREE** and very low **PCTUSED** settings. A lot of storage space will be wasted to minimize migration and chaining.
  PCTFREE = 30
  PCTUSED = 30
  (100 – PCTFREE) – PCTUSED = 40

- **Disk space usage is important and performance is secondary** – the application uses large tables and disk space usage is criticial. Here **PCTFREE** should be very low while **PCTUSED** is very high – the tables will experience some data row migration and chaining with a performance hit.
  PCTFREE = 5
  PCTUSED = 90
  (100 – PCTFREE) – PCTUSED = 5

Free lists: With Manual Segment Space Management, when a segment is created, it is created with a **Free List** that is used to track the blocks allocated to the segment that are available for row insertions.

- A segment can have more than one free list if the **FREELISTS** parameter is specified in the storage clause when an object is created.
- If a block has free space that falls below **PCTFREE**, that block is removed from the free list.
- Oracle improves performance by not considering blocks that are almost full as candidates for row insertions.

Automatic Segment Space Management

Free space can be managed either **automatically** or **manually**.

- Automatic simplifies the management of the **PCTUSED**, **FREELISTS**, **and FREELIST GROUPS** parameters.
- Automatic generally provides better space utilization where objects may vary considerably in terms of row size.

- This can also yield improved concurrent access handling for row insertions.
- A restriction is that you cannot use this approach if a tablespace will contain **LOBs**.

The free and used space for a segment is tracked with bitmaps instead of free lists.

- The bitmap is stored in the header section of the segment, in a separate set of blocks called **bitmapped blocks**.
- The bitmap tracks the status of each block in a segment with respect to available space.
- Think of an individual bit as either being "on" to indicate the block is available or "off" to indicate a block is or is not available.
- When a new row needs to be inserted into a segment, the bitmap is searched for a candidate block. This search occurs much more rapidly than can be done with a **Free List** because a **Bit Map Index** can often be entirely stored in memory and the use of a **Free List** requires searching a chain data structure (linked list).

Automatic segment management can only be enabled at the tablespace level, and only if the tablespace is locally managed. An example **CREATE TABLESPACE** command is shown here.

```
CREATE TABLESPACE user_data
DATAFILE '/u01/student/dbockstd/oradata/USER350data01.dbf' SIZE 20M
EXTENT MANAGEMENT LOCAL UNIFORM SIZE 40K
SEGMENT SPACE MANAGEMENT AUTO;
```

The **SEGMENT SPACE MANAGEMENT AUTO** clause specifies the creation of the bitmapped segments.

Automatic segment space management offers the following benefits:

- Ease of use.
- Better space utilization, especially for the objects with highly varying size rows.
- Better run-time adjustment to variations in concurrent access.
- Better multi-instance behavior in terms of performance/space utilization.

Statements that can increase the amount of free space in a database block:

- **DELETE** statements that delete rows, and
- **UPDATE** statements that update a column value to a **smaller value** than was previously required.
- **INSERT** statements, but only if the tablespace allows for compression and the INSERT causes data to be compressed, thereby freeing up some space in a block.
- Both of these statements release space that can be used subsequently by an INSERT statement.
- Released space may or may not be contiguous with the main area of free space in a data block.

Oracle coalesces the free space of a data block *only* when:

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

Oracle does this compression only in such situations, because otherwise the performance of a database system decreases due to the continuous compression of the free space in data blocks.

Using the Data Dictionary to Manage Storage

Periodically you will need to obtain information from the data dictionary about storage parameter settings. The following views are useful.

- **DBA_EXTENTS** – information on space allocation for segments.
- **DBA_SEGMENTS** – stores information on segments.
- **DBA_TABLESPACES** – a row is added when a tablespace is created.
- **DBA_DATA_FILES** – a row is added for each datafile in the database.
- **DBA_FREE_SPACE** – shows the space in each datafile that is free.

```
SELECT owner, segment_name, tablespace_name
FROM dba_extents;

OWNER      SEGMENT_NAME    TABLESPACE_NAME
---------- --------------- ---------------
SYS        FILE$           SYSTEM
SYS        BOOTSTRAP$      SYSTEM
SYS        OBJ$            SYSTEM
. . . more rows
```

```
SELECT owner, segment_name, tablespace_name "TS Name",
    initial_extent "Init", next_extent "Next",
    min_extents "Min", max_extents "Max", pct_increase "Pct"
FROM dba_segments
WHERE owner = 'DBOCK';

OWNER SEGMENT_NAME    TS Name    Init    Next  Min    Max  Pct
----- --------------- ------- ------- ------- ---- ------ ----
DBOCK DEPARTMENT      DATA01    49152   49152    1   1000    1
DBOCK DEPT_LOCATIONS  DATA01    49152   49152    1   1000    1
DBOCK PROJECT         DATA01    49152   49152    1   1000    1
DBOCK EMPLOYEE        DATA01    49152   49152    1   1000    1
DBOCK ASSIGNMENT      DATA01    49152   49152    1   1000    1
DBOCK DEPENDENT       DATA01    49152   49152    1   1000    1
```

```
SELECT owner, segment_name, extents, blocks
FROM dba_segments
```

```
WHERE owner = 'DBOCK';

OWNER SEGMENT_NAME       EXTENTS     BLOCKS
----- --------------- ---------- ----------
DBOCK TWONAME                  1         15
DBOCK WEATHER                  1         15
DBOCK WORKERANDSKILL           2         30
DBOCK PK_DEPARTMENT            1         15

SELECT tablespace_name "Ts Name", Count(*),
    MAX(blocks), SUM(blocks)
FROM dba_free_space
GROUP BY tablespace_name;

Ts Name         COUNT(*) MAX(BLOCKS) SUM(BLOCKS)
--------------- -------- ----------- -----------
DATA01                 1         172         172
DATA02                 1         120         120
INDEX01                1         248         248
SYSAUX                 1       36688       36688
SYSTEM                 1       16016       16016
UNDO01                52        1144        8944
USERS                  1         632         632
```

END OF NOTES