183
DBMS_SPACE

The DBMS_SPACE package enables you to analyze segment growth and space requirements.

This chapter contains the following topics:

- Security Model
- Data Structures
- Summary of DBMS_SPACE Subprograms

DBMS_SPACE Security Model

This package runs with SYS privileges. The execution privilege is granted to PUBLIC. Subprograms in this package run under the caller security. The user must have ANALYZE privilege on the object.

DBMS_SPACE Data Structures

The DBMS_SPACE package defines an OBJECT type, a RECORD type, and a TABLE type.

OBJECT Types

CREATE_TABLE_COST_COLINFO Object Type

RECORD Types

ASA_RECO_ROW Record Type

TABLE Types

ASA_RECO_ROW_TB Table Type

DBMS_SPACE CREATE_TABLE_COST_COLINFO Object Type

This type describes the datatype and size of a column in the table.

Syntax

```
TYPE create_table_cost_colinfo IS OBJECT(
    col_type    VARCHAR(200),
    col_size    NUMBER)
```

Attributes

Table 183-1 CREATE_TABLE_COST_COLINFO Object Type

Attribute	Description
col_type	Column type

Table 183-1 (Cont.) CREATE_TABLE_COST_COLINFO Object Type

Attribute	Description
col_size	Column size

DBMS_SPACE ASA_RECO_ROW Record Type

This type contains the column type of individual columns returned by the ASA_RECOMMENDATIONS Function.

Syntax

```
TYPE asa_reco_row IS RECORD (
   tablespace_name       VARCHAR2(30),
   segment_owner         VARCHAR2(30),
   segment_name          VARCHAR2(30),   segment_type          VARCHAR2(18),
   partition_name        VARCHAR2(30),
   allocated_space       NUMBER,
   used_space            NUMBER,
   reclaimable_space     NUMBER,
   chain_rowexcess       NUMBER,
   recommendations       VARCHAR2(1000),
   c1                    VARCHAR2(1000),
   c2                    VARCHAR2(1000),
   c3                    VARCHAR2(1000),
   task_id               NUMBER,
   mesg_id               NUMBER);
```

Attributes

Table 183-2 ASA_RECO_ROW Attributes

Field	Description
tablespace_name	Name of the tablespace containing the object
segment_owner	Name of the schema
segment_name	Name of the object
segment_type	Type of the segment 'TABLE','INDEX' and so on
partition_name	Name of the partition
allocated_space	Space allocated to the segment
used_space	Space actually used by the segment
reclaimable_space	Reclaimable free space in the segment
chain_rowexcess	Percentage of excess chain row pieces that can be eliminated
recommendations	Recommendation or finding for this segment
c1	Command associated with the recommendation
c2	Command associated with the recommendation
c3	Command associated with the recommendation
task_id	Advisor Task that processed this segment
mesg_id	Message ID corresponding to the recommendation

Related Topics

- DBMS_SPACE ASA_RECOMMENDATIONS Function
  This function returns recommendations using the stored results of the auto segment advisor. This function returns results from the latest run on any given object.

DBMS_SPACE ASA_RECO_ROW_TB Table Type

The type `asa_reco_row_tb` is a table of `asa_reco_row`.

Syntax

```
TYPE asa_reco_row_tb IS TABLE OF asa_reco_row;
```

Summary of DBMS_SPACE Subprograms

This table lists the `DBMS_SPACE` subprograms and briefly describes them.

Table 183-3 ***DBMS_SPACE Package Subprograms***

Subprogram	Description
ASA_RECOMMENDATIONS Function	Returns recommendations/findings of segment advisor run automatically by the system or manually invoked by the user
CREATE_INDEX_COST Procedure	Determines the cost of creating an index on an existing table
CREATE_TABLE_COST Procedures	Determines the size of the table given various attributes
FREE_BLOCKS Procedure	Returns information about free blocks in an object (table, index, or cluster)
ISDATAFILEDROPPABLE_NAME Procedure	Checks whether a datafile is droppable
OBJECT_DEPENDENT_SEGMENTS Function	Returns the list of segments that are associated with the object
OBJECT_GROWTH_TREND Function	A table function where each row describes the space usage of the object at a specific point in time
SHRINK_TABLESPACE Procedure	Analyzes a bigfile tablespace before resizing or resizes a bigfile tablespace and optionally returns information about the resize operation
SPACE_USAGE Procedures	Returns information about free blocks in an auto segment space managed segment
UNUSED_SPACE Procedure	Returns information about unused space in an object (table, index, or cluster)

DBMS_SPACE ASA_RECOMMENDATIONS Function

This function returns recommendations using the stored results of the auto segment advisor. This function returns results from the latest run on any given object.

Syntax

```
DBMS_SPACE.ASA_RECOMMENDATIONS (
   all_runs         IN    VARCHAR2 DEFAULT := TRUE,
   show_manual      IN    VARCHAR2 DEFAULT := TRUE,
   show_findings    IN     VARCHAR2 DEFAULT := FALSE)
 RETURN ASA_RECO_ROW_TB PIPELINED;
```

Parameters

Table 183-4 ASA_RECOMMENDATIONS Procedure Parameters

Parameter	Description
`all_runs`	Returns the results of all the auto advisor runs or only the results of the latest run. The valid values are `TRUE` and `FALSE`. The default value is `TRUE`. If `TRUE`, returns recommendations/findings for all runs of auto segment advisor. If `FALSE`, returns the results of the `LATEST` run only. `LATEST` does not make sense for manual invocation of segment advisor. This is applicable only for auto advisor.
`show_manual`	This parameter is used to indicate if the results of manual jobs should be included. If `TRUE`, results of manual tasks are shown. If `FALSE`, results of manual tasks are not shown. Specifying `manual=true` does not negate the specification of auto advisor tasks. However, the `all_runs` settings may override manual. If `all_runs` is `FALSE`, implying we only want to see the latest of auto advisor job, then manual may not be specified as `TRUE`. The valid values are `TRUE` and `FALSE`. The default value is `TRUE`.
`show_findings`	Shows only the findings instead of the recommendations. The valid values are `TRUE` and `FALSE`. The default value is `FALSE`.

Table 183-5 Parameter Usage

all_runs	show_manual	show_ findings	Outcome
`TRUE`	`TRUE`	`TRUE`	All findings from auto advisor and manual tasks.
`TRUE`	`TRUE`	`FALSE`	All recommendations from auto advisor and manual tasks.
`TRUE`	`FALSE`	`TRUE`	All findings from auto advisor tasks.
`TRUE`	`FALSE`	`FALSE`	All recommendations from all auto advisor tasks.
`FALSE`	`TRUE`	`TRUE`	N/A
`FALSE`	`TRUE`	`FALSE`	N/A
`FALSE`	`FALSE`	`TRUE`	Findings for the latest auto advisor task.
`FALSE`	`FALSE`	`FALSE`	Recommendations from the latest auto advisor task.

CREATE_INDEX_COST Procedure

This procedure determines the cost of creating an index on an existing table. The input is the DDL statement that will be used to create the index. The procedure will output the storage required to create the index.

Syntax

```
DBMS_SPACE.CREATE_INDEX_COST (
   ddl              IN    VARCHAR2,
   used_bytes       OUT   NUMBER,
   alloc_bytes      OUT   NUMBER,
   plan_table       IN    VARCHAR2 DEFAULT NULL);
```

Pragmas

```
pragma restrict_references(create_index_cost,WNDS);
```

Parameters

Table 183-6 CREATE_INDEX_COST Procedure Parameters

Parameter	Description
`ddl`	The create index DDL statement
`used_bytes`	The number of bytes representing the actual index data
`alloc_bytes`	Size of the index when created in the tablespace
`plan_table`	Which plan table to use, default `NULL`

Usage Notes

- The table on which the index is created must already exist.
- The computation of the index size depends on statistics gathered on the segment.
- It is imperative that the table must have been analyzed recently.
- In the absence of correct statistics, the results may be inaccurate, although the procedure will not raise any errors.

CREATE_TABLE_COST Procedures

This procedure is used in capacity planning to determine the size of the table given various attributes. The size of the object can vary widely based on the tablespace storage attributes, tablespace block size, and so on. There are two overloads of this procedure.

- The first version takes the column information of the table as argument and outputs the table size.
- The second version takes the average row size of the table as argument and outputs the table size.

This procedure can be used on tablespace of dictionary managed and locally managed extent management as well as manual and auto segment space management.

Syntax

```
DBMS_SPACE.CREATE_TABLE_COST (
   tablespace_name    IN VARCHAR2,
   avg_row_size       IN NUMBER,
   row_count          IN NUMBER,
   pct_free           IN NUMBER,
   used_bytes         OUT NUMBER,
   alloc_bytes        OUT NUMBER);

DBMS_SPACE.CREATE_TABLE_COST (
   tablespace_name    IN VARCHAR2,
   colinfos           IN CREATE_TABLE_COST_COLUMNS,
   row_count          IN NUMBER,
   pct_free           IN NUMBER,
   used_bytes         OUT NUMBER,
   alloc_bytes        OUT NUMBER);

CREATE TYPE create_table_cost_colinfo IS OBJECT (
   COL_TYPE   VARCHAR(200),
   COL_SIZE   NUMBER);
```

Parameters

Table 183-7 CREATE_TABLE_COST Procedure Parameters

Parameter	Description
`tablespace_name`	The tablespace in which the object will be created. The default is `SYSTEM` tablespace.
`avg_row_size`	The anticipated average row size in the table
`colinfos`	The description of the columns
`row_count`	The anticipated number of rows in the table
`pct_free`	The percentage of free space in each block for future expansion of existing rows due to updates
`used_bytes`	The space used by user data
`alloc_bytes`	The size of the object taking into account the tablespace extent characteristics

Usage Notes

- The `used_bytes` represent the actual bytes used by the data. This includes the overhead due to the block metadata, pctfree etc.
- The `alloc_bytes` represent the size of the table when it is created in the tablespace. This takes into account, the size of the extents in the tablespace and tablespace extent management properties.

Examples

```
-- review the parameters
SELECT argument_name, data_type, type_owner, type_name
FROM all_arguments
WHERE object_name = 'CREATE_TABLE_COST'
AND overload = 2

-- examine the input parameter type
```

```
SELECT text
FROM dba_source
WHERE name = 'CREATE_TABLE_COST_COLUMNS';

-- drill down further into the input parameter type
SELECT text
FROM dba_source
WHERE name = 'create_table_cost_colinfo';

set serveroutput on

DECLARE
 ub NUMBER;
 ab NUMBER;
 cl sys.create_table_cost_columns;
BEGIN
  cl := sys.create_table_cost_columns( sys.create_table_cost_colinfo('NUMBER',10),
        sys.create_table_cost_colinfo('VARCHAR2',30),
        sys.create_table_cost_colinfo('VARCHAR2',30),
        sys.create_table_cost_colinfo('DATE',NULL));

  DBMS_SPACE.CREATE_TABLE_COST('SYSTEM',cl,100000,0,ub,ab);

  DBMS_OUTPUT.PUT_LINE('Used Bytes: ' || TO_CHAR(ub));
  DBMS_OUTPUT.PUT_LINE('Alloc Bytes: ' || TO_CHAR(ab));
END;
/
```

FREE_BLOCKS Procedure

This procedure returns information about free blocks in an object (table, index, or cluster).

See SPACE_USAGE Procedures for returning free block information in an auto segment space managed segment.

Syntax

```
DBMS_SPACE.FREE_BLOCKS (
   segment_owner     IN  VARCHAR2,
   segment_name      IN  VARCHAR2,
   segment_type      IN  VARCHAR2,
   freelist_group_id IN  NUMBER,
   free_blks         OUT NUMBER,
   scan_limit        IN  NUMBER DEFAULT NULL,
   partition_name    IN  VARCHAR2 DEFAULT NULL);
```

Pragmas

```
pragma restrict_references(free_blocks,WNDS);
```

Parameters

Table 183-8 FREE_BLOCKS Procedure Parameters

Parameter	Description
`segment_owner`	Schema name of the segment to be analyzed
`segment_name`	Segment name of the segment to be analyzed

Table 183-8 (Cont.) FREE_BLOCKS Procedure Parameters

Parameter	Description
`segment_type`	Type of the segment to be analyzed (`TABLE`, `INDEX`, or `CLUSTER`): • `TABLE` • `TABLE PARTITION` • `TABLE SUBPARTITION` • `INDEX` • `INDEX PARTITION` • `INDEX SUBPARTITION` • `CLUSTER` • `LOB` • `LOB PARTITION` • `LOB SUBPARTITION`
`freelist_group_id`	Freelist group (instance) whose free list size is to be computed
`free_blks`	Returns count of free blocks for the specified group
`scan_limit`	Maximum number of free list blocks to read (optional). Use a scan limit of X you are interested only in the question, "Do I have X blocks on the free list?"
`partition_name`	Partition name of the segment to be analyzed. This is only used for partitioned tables. The name of subpartition should be used when partitioning is composite.

Examples

The following uses the `CLUS` cluster in `SCOTT` schema with 4 freelist groups. It returns the number of blocks in freelist group 3 in `CLUS`.

```
DBMS_SPACE.FREE_BLOCKS('SCOTT', 'CLUS', 'CLUSTER', 3, :free_blocks);
```

Note:

An error is raised if `scan_limit` is not a positive number.

ISDATAFILEDROPPABLE_NAME Procedure

This procedure checks whether a datafile is droppable. This procedure may be called before actually dropping the file.

Syntax

```
DBMS_SPACE.ISDATAFILEDROPPABLE_NAME (
   filename      IN     VARCHAR2,
   value         OUT    NUMBER);
```

Pragmas

```
pragma restrict_references(free_blocks,WNDS);
```

Parameters

Table 183-9 ISDATAFILEDROPPABLE_NAME Procedure Parameters

Parameter	Description
`filename`	Name of the file
`value`	Values: `0` if the file is not droppable, `1` if the file is droppable.

Examples

```
DECLARE   fname    VARCHAR2(100);   retval   NUMBER;BEGIN   SELECT file_name   INTO
fname   FROM dba_data_files   WHERE file_name like
'%empty%';DBMS_SPACE.ISDATAFILEDROPPABLE_NAME(fname,
retval);DBMS_OUTPUT.PUT_LINE(retval);END;/
```

OBJECT_DEPENDENT_SEGMENTS Function

This table function, given an object, returns the list of segments that are associated with the object.

Syntax

```
DBMS_SPACE.OBJECT_DEPENDENT_SEGMENTS(
   objowner    IN     VARCHAR2,
   objname     IN     VARCHAR2,
   partname    IN     VARCHAR2,
   objtype     IN     NUMBER)
  RETURN dependent_segments_table PIPELINED;
```

Parameters

Table 183-10 OBJECT_DEPENDENT_SEGMENTS Function Parameters

Parameter	Description
`objowner`	The schema containing the object
`objname`	The name of the object
`partname`	The name of the partition
`objtype`	Type of the object: • `OBJECT_TYPE_TABLE constant positive := 1;` • `OBJECT_TYPE_NESTED_TABLE constant positive := 2;` • `OBJECT_TYPE_INDEX constant positive := 3;` • `OBJECT_TYPE_CLUSTER constant positive := 4;` • `OBJECT_TYPE_TABLE_PARTITION constant positive := 7;` • `OBJECT_TYPE_INDEX_PARTITION constant positive := 8;` • `OBJECT_TYPE_TABLE_SUBPARTITION constant positive := 9;` • `OBJECT_TYPE_INDEX_SUBPARTITION constant positive := 10;` • `OBJECT_TYPE_MV constant positive := 13;` • `OBJECT_TYPE_MVLOG constant positive := 14;`

Return Values

The content of one row of a `dependent_segments_table`:

```
TYPE object_dependent_segment IS RECORD (
   segment_owner      VARCHAR2(100),
   segment_name       VARCHAR2(100),
   segment_type       VARCHAR2(100),
   tablespace_name    VARCHAR2(100),
   partition_name     VARCHAR2(100),
   lob_column_name    VARCHAR2(100));
```

Table 183-11 OBJECT_DEPENDENT_SEGMENT Type Parameters

Parameter	Description
`segment_owner`	The schema containing the segment
`segment_name`	The name of the segment
`segment_type`	The type of the segment, such as table, index or LOB
`tablespace_name`	The name of the tablespace
`partition_name`	The name of the partition, if any
`lob_column_name`	The name of the `LOB` column, if any

OBJECT_GROWTH_TREND Function

This is a table function. The output is one or more rows where each row describes the space usage of the object at a specific point in time.

Either the space usage totals will be retrieved from Automatic Workload Repository Facilities (AWRF), or the current space usage will be computed and combined with space usage deltas retrieved from AWRF.

Syntax

```
DBMS_SPACE.OBJECT_GROWTH_TREND (
   object_owner           IN    VARCHAR2,
   object_name            IN    VARCHAR2,
   object_type            IN    VARCHAR2,
   partition_name         IN    VARCHAR2 DEFAULT NULL,
   start_time             IN    TIMESTAMP DEFAULT NULL,
   end_time               IN    TIMESTAMP DEFAULT NULL,
   interval               IN    DSINTERVAL_UNCONSTRAINED DEFAULT NULL,
   skip_interpolated      IN    VARCHAR2 DEFAULT 'FALSE',
   timeout_seconds        IN    NUMBER DEFAULT NULL,
   single_datapoint_flag  IN    VARCHAR2 DEFAULT 'TRUE')
 RETURN object_growth_trend_table PIPELINED;
```

Parameters

Table 183-12 OBJECT_GROWTH_TREND Function Parameters

Parameter	Description
`object_owner`	The schema containing the object
`object_name`	The name of the object

Table 183-12 (Cont.) OBJECT_GROWTH_TREND Function Parameters

Parameter	Description
`object_type`	The type of the object
`partition_name`	The name of the partition
`start_time`	Statistics generated after this time will be used in generating the growth trend
`end_time`	Statistics generated until this time will be used in generating the growth trend
`interval`	The interval at which to sample
`skip_interpolated`	Whether interpolation of missing values should be skipped
`timeout_seconds`	The time-out value for the function in seconds
`single_data_point_flag`	Whether in the absence of statistics the segment should be sampled

Return Values

The `object_growth_trend_row` and `object_growth_trend_table` are used by the `OBJECT_GROWTH_TREND` table function to describe its output.

```
TYPE object_growth_trend_row IS RECORD(
   timepoint      TIMESTAMP,
   space_usage    NUMBER,
   space_alloc    NUMBER,
   quality        VARCHAR(20));
```

Table 183-13 OBJECT_GROWTH_TREND_ROW Type Parameters

Parameter	Description
`timepoint`	The time at which the statistic was recorded
`space_usage`	The space used by data
`space_alloc`	The size of the segment including overhead and unused space
`quality`	The quality of result: "`GOOD`", "`INTERPOLATED`", "`PROJECTION`"

```
TYPE object_growth_trend_table IS TABLE OF object_growth_trend_row;
```

SHRINK_TABLESPACE Procedure

This procedure can resize a tablespace or analyze a tablespace before resizing.

Syntax

```
DBMS_SPACE.SHRINK_TABLESPACE (
   ts_name             IN  VARCHAR2,
   shrink_mode         IN  NUMBER,
   target_size         IN  NUMBER,
   shrink_result       OUT CLOB);


DBMS_SPACE.SHRINK_TABLESPACE (
   ts_name             IN  VARCHAR2,
   shrink_mode         IN   NUMBER,
   target_size         IN   NUMBER);
```

Parameters

Table 183-14   SHRINK_TABLESPACE Procedure Parameters

Parameter	Description
`ts_name`	The name of the tablespace to be analyzed or resized
`shrink_mode`	The shrink mode to execute. The values are: • `TS_SHRINK_MODE_ANALYZE` • `TS_SHRINK_MODE_ONLINE` • `TS_SHRINK_MODE_AUTO` • `TS_SHRINK_MODE_OFFLINE` The default mode is `TS_SHRINK_MODE_ONLINE` which moves objects online by default, except for index-organized tables. `TS_SHRINK_MODE_AUTO` will move objects online by default, but if the online move fails, it will attempt to move them offline. `TS_SHRINK_MODE_OFFLINE` offers the best shrink outcome and performance.
`target_size`	The desired tablespace size specified in **bytes**. The default value is `TS_TARGET_MAX_SHRINK`.
`shrink_result`	Output result of the procedure returned as a CLOB. The output results for `TS_SHRINK_MODE_ONLINE` include: • total number and size of moved objects • original and new datafile size • process time The output results for `TS_SHRINK_MODE_ANALYZE` include: • list of movable objects • total number and size of movable objects • suggested target size • process time

Deprecated Parameters

Table 183-15   Deprecated Parameters

Oracle Database 23ai release 23.6	Oracle Database 23ai release 23.7 and later
`TS_MODE_ANALYZE`	`TS_SHRINK_MODE_ANALYZE`
`TS_MODE_SHRINK`	`TS_SHRINK_MODE_ONLINE`
`TS_MODE_SHRINK_FORCE`	`TS_SHRINK_MODE_AUTO`
No equivalent parameter	`TS_SHRINK_MODE_OFFLINE`

Errors

Table 183-16   SHRINK_TABLESPACE Procedure Errors

Error	Description
`ORA-00054`	**Message:** Failed to acquire a lock (Type: "Type", Name: "Name", Description: "Description") because it is currently held by another session. The resource being locked can be identified by ID1_value ("ID1_description") and ID2_value ("ID2_description") **Reasons:** • Procedure exited because it can't move an object. • Tablespace is currently help by another session.

Usage Notes

The `SHRINK_TABLESPACE` procedure leverages online DDL to reorganize the objects in the datafile, and once the necessary objects have been reorganized, issues a datafile resize.

If the `SHRINK_TABLESPACE` procedure is interrupted, the currently running online DDL will be canceled and rolled back to the consistent state prior to the current online DDL invocation. All objects which have been reorganized prior to the interruption of the `SHRINK_TABLESPACE` procedure will remain reorganized. Subsequent invocations of `SHRINK_TABLESPACE` will benefit from already reorganized objects from the earlier interrupted shrink operation by not having to reorganize these objects again, essentially continuing the reorganization of objects where the prior invocation was canceled. Likewise, if an object has been fragmented again in the time between the two runs, `SHRINK_TABLESPACE` will reorganize it again.

Examples

This example analyzes bigfile tablespace TBS_1.

```
set serveroutput on
execute dbms_space.shrink_tablespace('TBS_1', shrink_mode =>
DBMS_SPACE.TS_SHRINK_MODE_ANALYZE);

------------------------------ANALYZE
RESULT--------------------------------------
1. { BG_TEST.SYS_IL0000081422C00004$$ | type: INDEX | blocks: 256 |
tablespace_name: TBS_1 }
2. { BG_TEST.SYS_IL0000081422C00005$$ | type: INDEX | blocks: 512 |
tablespace_name: TBS_1 }
3. { BG_TEST.T2 | type: TABLE | blocks: 512 | tablespace_name: TBS_1 }
4. { BG_TEST.T2_LOB1 | type: LOBSEGMENT | blocks: 45824 | tablespace_name:
TBS_1}
5. { BG_TEST.T2_LOB2 | type: LOBSEGMENT | blocks: 41216 | tablespace_name:
TBS_1}
Total Movable Objects: 5
Total Movable Size(GB): .67
Orginal Datafile Size(GB): 10
Suggested Target Size(GB): 2.09
Process Time: +00 00:00:03.94897
```

This example shrinks the bigfile tablespace TBS_1 to its current minimum possible size.

```
set serveroutput on
execute dbms_space.shrink_tablespace('TBS_1');

-------------------SHRINK RESULT-------------------
Total Moved Objects: 5
Total Moved Size(GB): 1.35
Orginal Datafile Size(GB): 10
New Datafile Size(GB): 1.81
Process Time: +00 00:00:50.94897
```

SPACE_USAGE Procedures

This procedure has two variations to show space usage.

The first form of the procedure shows the space usage of data blocks under the segment High Water Mark. You can calculate usage for `LOB`s, `LOB PARTITIONS` and `LOB SUBPARTITIONS`. This procedure can only be used on tablespaces that are created with auto segment space management. The bitmap blocks, segment header, and extent map blocks are not accounted for by this procedure. Note that this overload cannot be used on `SECUREFILE LOB`s.

Note:

For LOB segments, the number of blocks that is returned from `full_blocks` and `unformatted_blocks` is actually the number of chunks for the LOB segment.

The second form of the procedure returns information about `SECUREFILE LOB` space usage. It will return the amount of space in blocks being used by all the `SECUREFILE LOB`s in the `LOB` segment. The procedure displays the space actively used by the LOB column, freed space that has retention expired, and freed space that has retention unexpired. Note that this overload can be used only on `SECUREFILE LOB`s.

Syntax

```
DBMS_SPACE.SPACE_USAGE(
   segment_owner           IN  VARCHAR2,
   segment_name            IN  VARCHAR2,
   segment_type            IN  VARCHAR2,
   unformatted_blocks      OUT NUMBER,
   unformatted_bytes       OUT NUMBER,
   fs1_blocks              OUT NUMBER,
   fs1_bytes               OUT NUMBER,
   fs2_blocks              OUT NUMBER,
   fs2_bytes               OUT NUMBER,
   fs3_blocks              OUT NUMBER,
   fs3_bytes               OUT NUMBER,
   fs4_blocks              OUT NUMBER,
   fs4_bytes               OUT NUMBER,
   full_blocks             OUT NUMBER,
   full_bytes              OUT NUMBER,
   partition_name          IN  VARCHAR2 DEFAULT NULL);

DBMS_SPACE.SPACE_USAGE(
   segment_owner           IN    VARCHAR2,
segment_name          IN    VARCHAR2,
segment_type          IN    VARCHAR2,
segment_size_blocks   OUT   NUMBER,
segment_size_bytes    OUT   NUMBER,
used_blocks           OUT   NUMBER,
used_bytes            OUT   NUMBER,
expired_blocks        OUT   NUMBER,
expired_bytes         OUT   NUMBER,
unexpired_blocks      OUT   NUMBER,
unexpired_bytes       OUT   NUMBER,
partition_name        IN    VARCHAR2 DEFAULT NULL);
```

Parameters

Table 183-17 SPACE_USAGE Procedure Parameters

Parameter	Description
`segment_owner`	Schema name of the segment to be analyzed
`segment_name`	Name of the segment to be analyzed
`partition_name`	Partition name of the segment to be analyzed
`segment_type`	Type of the segment to be analyzed (`TABLE`, `INDEX`, or `CLUSTER`): • `TABLE` • `TABLE PARTITION` • `TABLE SUBPARTITION` • `INDEX` • `INDEX PARTITION` • `INDEX SUBPARTITION` • `CLUSTER` • `LOB` • `LOB PARTITION` • `LOB SUBPARTITION`
`unformatted_blocks`	For LOB segments, the number of blocks that is returned from unformatted_blocks is actually the number of chunks for the LOB segment.
`unformatted bytes`	Total number of bytes unformatted
`fs1_blocks`	Number of blocks having at least 0 to 25% free space
`fs1_bytes`	Number of bytes having at least 0 to 25% free space
`fs2_blocks`	Number of blocks having at least 25 to 50% free space
`fs2_bytes`	Number of bytes having at least 25 to 50% free space
`fs3_blocks`	Number of blocks having at least 50 to 75% free space
`fs3_bytes`	Number of bytes having at least 50 to 75% free space
`fs4_blocks`	Number of blocks having at least 75 to 100% free space
`fs4_bytes`	Number of bytes having at least 75 to 100% free space
`full_blocks`	The number of blocks that is returned from full_blocks is actually the number of chunks for the LOB segment
`full_bytes`	Total number of bytes full in the segment
`segment_size_blocks`	Number of blocks allocated to the segment
`segment_size_bytes`	Number of bytes allocated to the segment

Table 183-17 (Cont.) SPACE_USAGE Procedure Parameters

Parameter	Description
`used_blocks`	Number blocks allocated to the LOB that contains active data
`used_bytes`	Number bytes allocated to the LOB that contains active data
`expired_blocks`	Number of expired blocks used by the LOB to keep version data
`expired_bytes`	Number of expired bytes used by the LOB to keep version data
`unexpired_blocks`	Number of unexpired blocks used by the LOB to keep version data
`unexpired_bytes`	Number of unexpired bytes used by the LOB to keep version data
`partition_name`	Name of the partition (`NULL` if not a partition)

Examples

```
variable unf number;
variable unfb number;
variable fs1 number;
variable fs1b number;
variable fs2 number;
variable fs2b number;
variable fs3 number;
variable fs3b number;
variable fs4 number;
variable fs4b number;
variable full number;
variable fullb number;

begin
dbms_space.space_usage('U1','T',
                        'TABLE',
                        :unf, :unfb,
                        :fs1, :fs1b,
                        :fs2, :fs2b,
                        :fs3, :fs3b,
                        :fs4, :fs4b,
                        :full, :fullb);
end;
/
print unf ;
print unfb ;
print fs4 ;
print fs4b;
print fs3 ;
print fs3b;
print fs2 ;
print fs2b;
print fs1 ;
print fs1b;
print full;
print fullb;
```

UNUSED_SPACE Procedure

This procedure returns information about unused space in an object (table, index, or cluster).

Syntax

```
DBMS_SPACE.UNUSED_SPACE (
   segment_owner              IN  VARCHAR2,
   segment_name               IN  VARCHAR2,
   segment_type               IN  VARCHAR2,
   total_blocks               OUT NUMBER,
   total_bytes                OUT NUMBER,
   unused_blocks              OUT NUMBER,
   unused_bytes               OUT NUMBER,
   last_used_extent_file_id   OUT NUMBER,
   last_used_extent_block_id  OUT NUMBER,
   last_used_block            OUT NUMBER,
   partition_name             IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 183-18 UNUSED_SPACE Procedure Parameters

Parameter	Description
`segment_owner`	Schema name of the segment to be analyzed
`segment_name`	Segment name of the segment to be analyzed
`segment_type`	Type of the segment to be analyzed (`TABLE`, `INDEX`, or `CLUSTER`): • `TABLE` • `TABLE PARTITION` • `TABLE SUBPARTITION` • `INDEX` • `INDEX PARTITION` • `INDEX SUBPARTITION` • `CLUSTER` • `LOB` • `LOB PARTITION` • `LOB SUBPARTITION`
`total_blocks`	Returns total number of blocks in the segment
`total_bytes`	Returns total number of blocks in the segment, in bytes
`unused_blocks`	Returns number of blocks which are not used
`unused_bytes`	Returns, in bytes, number of blocks which are not used
`last_used_extent_file_id`	Returns the file ID of the last extent which contains data
`last_used_extent_block_id`	Returns the starting block ID of the last extent which contains data
`last_used_block`	Returns the last block within this extent which contains data
`partition_name`	Partition name of the segment to be analyzed. This is only used for partitioned tables; the name of subpartition should be used when partitioning is compose.

Examples

The following declares the necessary bind variables and executes.

```
DBMS_SPACE.UNUSED_SPACE('SCOTT', 'EMP', 'TABLE', :total_blocks,
   :total_bytes,:unused_blocks, :unused_bytes, :lastextf,
   :last_extb, :lastusedblock);
```