107
DBMS_INMEMORY_ADMIN

DBMS_INMEMORY_ADMIN provides interfaces for managing an In-Memory FastStart (IM FastStart) area and In-Memory Expressions (IM expressions).

This chapter contains the following topics:

> **See Also:**
>
> *Oracle Database In-Memory Guide* to learn more about Oracle Database In-Memory features

DBMS_INMEMORY_ADMIN Overview

This package provides interfaces for managing In-Memory Expressions (IM expressions) and the In-Memory FastStart (IM FastStart) area.

IM Expressions

Analytic queries often contain complex expressions or calculations that consume significant CPU and memory during execution. Use IME_CAPTURE_EXPRESSIONS to identify these frequently used ("hot") expressions and IME_POPULATE_EXPRESSIONS to populate them in the IM column store. By using IM expressions, the database avoids repeated computations and improves performance.

The database represents IM expressions as system-generated virtual columns. The name of an IM virtual column begins with SYS_IME. You can also use DBMS_INMEMORY_ADMIN.IME_DROP_ALL_EXPRESSIONS and DBMS_INMEMORY.IME_DROP_EXPRESSIONS to remove existing SYS_IME columns.

The DBA_IM_EXPRESSIONS view shows the SYS_IME columns that have the INMEMORY attribute. After using the IME_CAPTURE_EXPRESSIONS procedure, you can query this view to see the hot expressions added to different tables in the database.

> **See Also:**
>
> *Oracle Database In-Memory Guide* to learn more about IM expressions

IM FastStart Area

The IM FastStart area stores data that optimizes the population of the IM column store when the database restarts. Because the database reads columnar data directly from persistent storage without needing to compress or format it, population is faster when a database instance restarts.

When you enable IM FastStart for the IM column store, you must specify an ASSM tablespace for the IM FastStart area. The tablespace stores the data in a SecureFiles LOB named `SYSDBIMFS_LOGSEG$`. The `SYSAUX` tablespace stores the metadata. When data is populated or repopulated in the IM column store, the database automatically writes the data to the IM FastStart area. You cannot manually force a write. If you specify an object as `NO INMEMORY`, then the database removes it from the IM FastStart area.

When the IM FastStart area is under space pressure, the database automatically drops the oldest 15% of segments and continues saving columnar data. If space is unavailable, then the database stops writing to the IM FastStart area.

> **See Also:**
>
> *Oracle Database In-Memory Guide* to learn more about IM expressions

Automatic In-Memory

Automatic In-Memory uses access tracking and column statistics to manage objects in the IM column store. If the IM column store is full, and if other more frequently accessed segments would benefit from population in the IM column store, then the IM column store evicts inactive segments. If the IM column store is configured to hold all `INMEMORY` segments, however, then Automatic In-Memory takes no action.

By default, Automatic In-Memory checks usage statistics for the past 31 days. You can change the current setting by supplying the `AIM_STATWINDOW_DAYS` parameter to `DBMS_INMEMORY_ADMIN.AIM_SET_PARAMETER`.

> **See Also:**
>
> *Oracle Database In-Memory Guide* to learn more about Automatic In-Memory

Database In-Memory Wait on Populate

The `POPULATE_WAIT` function initiates population of all `INMEMORY` objects that have a priority greater than or equal to the specified priority, and then returns a status value for the population. A user-specified interval specifies the maximum time that the function waits before returning the value to the caller.

> **See Also:**
>
> *Oracle Database In-Memory Guide* to learn more about the wait on populate feature

DBMS_INMEMORY_ADMIN Security Model

This package requires administrator privileges. Package subprograms execute with invoker's rights.

DBMS_INMEMORY_ADMIN Operational Notes

It is possible for a `DBMS_INMEMORY_ADMIN` FastStart operation to fail or be interrupted.

In a failure or interruption scenario, the following rules determine which subprograms you can use:

- If `FASTSTART_ENABLE` does not succeed, then the only permitted operation is re-executing `FASTSTART_ENABLE`.
- If `FASTSTART_MIGRATE_STORAGE` does not succeed, then the only permitted operation is re-executing `FASTSTART_MIGRATE_STORAGE`.
- If `FASTSTART_DISABLE` does not succeed, then all `DBMS_INMEMORY_ADMIN` operations are permitted.

Summary of DBMS_INMEMORY_ADMIN Subprograms

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

Table 107-1 DBMS_INMEMORY_ADMIN Package Subprograms

Subprogram	Description
AIM_GET_PARAMETER Procedure	This procedure obtains the current values for parameters that control Automatic In-Memory.
AIM_SET_PARAMETER Procedure	The procedure customizes the execution environment of Automatic In-Memory
FASTSTART_DISABLE Procedure	This procedure disables the In-Memory FastStart (IM FastStart) feature.
FASTSTART_ENABLE Procedure	This procedure enables IM FastStart and assigns a tablespace.
FASTSTART_MIGRATE_STORAGE Procedure	This procedure moves all IM FastStart data and metadata from the existing tablespace to the specified new tablespace.
GET_FASTSTART_TABLESPACE Function	This function returns the name of the tablespace that is currently designated for IM FastStart.
IME_CAPTURE_EXPRESSIONS Procedure	This procedure captures the 20 most frequently accessed ("hottest") expressions in the database in the specified time interval.
IME_CLOSE_CAPTURE_WINDOW Procedure	This procedure signals the end of the current expression capture window.
IME_DROP_ALL_EXPRESSIONS Procedure	This procedure drops all `SYS_IME` expression virtual columns in the database.
IME_GET_CAPTURE_STATE Procedure	This procedure returns the current capture state of the expression capture window and the timestamp of the most recent modification.

Table 107-1 (Cont.) DBMS_INMEMORY_ADMIN Package Subprograms

Subprogram	Description
IME_OPEN_CAPTURE_WINDOW Procedure	This procedure signals the beginning of an expression capture window.
IME_POPULATE_EXPRESSIONS Procedure	This procedure forces the population of expressions captured in the latest invocation of `DBMS_INMEMORY_ADMIN.IME_CAPTURE_EXPRESSIONS`.
POPULATE_WAIT Function	Initiates population of all `INMEMORY` objects that have a priority greater than or equal to the specified priority, and sets a timeout interval within which population must occur

AIM_GET_PARAMETER Procedure

This procedure obtains the current values for parameters that control Automatic In-Memory.

Syntax

```
DBMS_INMEMORY_ADMIN.AIM_GET_PARAMETER(
   parameter    IN    NUMBER,
   value        OUT   NUMBER);
```

Parameters

Parameter	Description
`parameter`	Specifies a predefined constant that controls Automatic In-Memory. The only valid constant is `AIM_STATWINDOW_DAYS`, which specifies the number of days in the sliding statistics window. Automatic In-Memory uses this duration to filter statistics for `INMEMORY` objects as part of its algorithms. For example, if the duration is set to 7 days, then Automatic In-Memory considers only statistics of the past 7 days for its algorithms. The default is `1`.
`value`	Specifies the value assigned to `AIM_STATWINDOW_DAYS`.

Example 107-1 Getting the Number of Days in the Statistics Window

The following code prints the number of days in the statistics window to the screen:

```
VARIABLE b_statwin NUMBER

BEGIN

DBMS_INMEMORY_ADMIN.AIM_GET_PARAMETER(DBMS_INMEMORY_ADMIN.AIM_STATWINDOW_DAYS,
 :b_statwin);
END;
/

PRINT b_statwin
```

Sample output appears below:

```
B_STATWIN
---------
       14
```

> **See Also:**
>
> *Oracle Database In-Memory Guide* to learn how to use `AIM_GET_PARAMETER`

AIM_SET_PARAMETER Procedure

The procedure customizes the execution environment of Automatic In-Memory.

Syntax

```
DBMS_INMEMORY_ADMIN.AIM_SET_PARAMETER(
   parameter    IN    NUMBER,
   value        IN    NUMBER);
```

Parameters

Parameter	Description
`parameter`	Specifies a predefined constant that controls Automatic In-Memory. The only valid constant is `AIM_STATWINDOW_DAYS`, which specifies the number of days in the sliding statistics window. The default is `1`.
`value`	Assigns the value assigned to `AIM_STATWINDOW_DAYS`.

Example 107-2 Setting the Number of Days in the Statistics Window

The following example gets the current number of days in the window, sets it to 14, and then prints the value to the screen:

```
VARIABLE b_statwin NUMBER

BEGIN

DBMS_INMEMORY_ADMIN.AIM_GET_PARAMETER(DBMS_INMEMORY_ADMIN.AIM_STATWINDOW_DAYS,
 :b_statwin);
END;
/

PRINT b_statwin

BEGIN

DBMS_INMEMORY_ADMIN.AIM_SET_PARAMETER(DBMS_INMEMORY_ADMIN.AIM_STATWINDOW_DAYS,
 14);
END;
/
```

```
BEGIN

DBMS_INMEMORY_ADMIN.AIM_GET_PARAMETER(DBMS_INMEMORY_ADMIN.AIM_STATWINDOW_DAYS,
 :b_statwin);
END;
/

PRINT b_statwin
```

Sample output appears below:

```
 B_STATWIN
----------
         1

 B_STATWIN
----------
        14
```

> **See Also:**
>
> *Oracle Database In-Memory Guide* to learn how to use `AIM_GET_PARAMETER`

FASTSTART_DISABLE Procedure

This procedure disables the In-Memory FastStart (IM FastStart) feature.

Syntax

```
DBMS_INMEMORY_ADMIN.FASTSTART_DISABLE();
```

Security Model

Administrator privileges are required to execute this procedure.

Usage Notes

When you execute the procedure, the database executes the following actions:

1. Waits until all IM FastStart operations complete
2. Disables the IM FastStart feature, and performs the following operations:
   - Cleans the IM FastStart area
   - Deletes IM FastStart metadata stored in the `SYSAUX` tablespace
   - Releases the IM FastStart tablespace (but does not delete it)

This procedure does not interrupt or affect any concurrent IM column store operations.

Examples

The following PL/SQL program disables the IM FastStart feature:

```
EXEC DBMS_INMEMORY_ADMIN.FASTSTART_DISABLE;
```

The following query shows that the LOB for the IM FastStart tablespace has been deleted (sample output included):

```
COL OWNER FORMAT a5
COL SEGMENT_NAME FORMAT a20
SELECT   l.OWNER, l.SEGMENT_NAME, SUM(s.BYTES)/1024/1024 MB
FROM     DBA_LOBS l, DBA_SEGMENTS s
WHERE    l.SEGMENT_NAME = s.SEGMENT_NAME
AND      l.TABLESPACE_NAME = 'FS_TBS'
GROUP BY l.OWNER, l.SEGMENT_NAME;

no rows selected
```

FASTSTART_ENABLE Procedure

This procedure enables In-Memory FastStart (IM FastStart), and designates a tablespace for the IM FastStart (FastStart) area.

> **Note:**
>
> A multitenant container database is the only supported architecture in Oracle Database 21c and later releases. While the documentation is being revised, legacy terminology may persist. In most cases, "database" and "non-CDB" refer to a CDB or PDB, depending on context. In some contexts, such as upgrades, "non-CDB" refers to a non-CDB from a previous release.

Syntax

```
DBMS_INMEMORY_ADMIN.FASTSTART_ENABLE(
   tbs_name    IN    VARCHAR2,
   nologging   IN    BOOLEAN DEFAULT TRUE);
```

Parameters

Table 107-2 FASTSTART_ENABLE Procedure Parameters

Parameter	Description
`tbs_name`	The name of the ASSM tablespace for the FastStart area.
`nologging`	The logging mode of the LOB created for the FastStart area. If the `nologging` parameter is set to `FALSE`, then the database creates the FastStart LOB with the `LOGGING` option. If set to `TRUE` (default), then the database creates the LOB with the `NOLOGGING` option.

Security Model

Administrator privileges are required to execute this procedure.

Usage Notes

To enable IM FastStart, the ASSM tablespace specified in `FASTSTART_ENABLE` must exist, and the `SYSAUX` tablespace must be online. Only one FastStart tablespace can exist for every PDB or non-CDB. The specified tablespace must have enough space to store data for the IM column store, and it must not contain any other data before it is designated for the FastStart area. Oracle recommends sizing the tablespace at least twice of the size of the `INMEMORY_SIZE` initialization parameter.

The database does not create the FastStart area on disk until the IM column store is populated. After population, the data periodically saves the columnar data (but not metadata such as the transaction journal) to the FastStart area, which is represented on disk as the `SYSDBIMFS_LOBSEG$` segment. The database stores the FastStart metadata in the `SYSAUX` tablespace. In an Oracle Real Application Clusters (Oracle RAC) environment, IM FastStart data is shared across all nodes.

> **Note:**
>
> IM FastStart is not supported in a standby database instance.

Whereas the initial loading of IMCUs into memory is expensive and CPU-bound, an IM FastStart tablespace requires intermittent I/O. The database periodically writes columnar data to the IM FastStart area. If a database instance must restart, then Oracle Database reads the columnar data directly from the IM FastStart area rather than reconstructing the IMCUs from scratch. No compression or formatting of the columnar data is required.

Examples

This example creates `fs_tbs` as an ASSM tablespace, and then uses `FASTSTART_ENABLE` to specify this tablespace as the IM FastStart area:

```
CREATE TABLESPACE fs_tbs
  DATAFILE 'fs_tbs.dbf' SIZE 500M
  EXTENT MANAGEMENT LOCAL
  SEGMENT SPACE MANAGEMENT AUTO;

EXEC DBMS_INMEMORY_ADMIN.FASTSTART_ENABLE('fs_tbs');
```

The following query shows that the IM FastStart LOB was created (sample output included):

```
COL OWNER FORMAT a5
COL SEGMENT_NAME FORMAT a20
SELECT   l.OWNER, l.SEGMENT_NAME, SUM(s.BYTES)/1024/1024 MB
FROM     DBA_LOBS l, DBA_SEGMENTS s
WHERE    l.SEGMENT_NAME = s.SEGMENT_NAME
AND      l.TABLESPACE_NAME = 'FS_TBS'
GROUP BY l.OWNER, l.SEGMENT_NAME;
```

```
OWNER SEGMENT_NAME         MB
----- -------------------- ----------
SYS   SYSDBIMFS_LOBSEG$    .125
```

FASTSTART_MIGRATE_STORAGE Procedure

This procedure moves the In-Memory FastStart (IM FastStart) data and catalogs from the current tablespace to a new tablespace.

Syntax

```
DBMS_INMEMORY_ADMIN.FASTSTART_MIGRATE_STORAGE(
   tbs_name    IN    VARCHAR2 );
```

Parameters

Table 107-3 FASTSTART_MIGRATE_STORAGE Procedure Parameters

Parameter	Description
tbs_name	The name of the new ASSM tablespace for the IM FastStart area.

Security Model

DBA privileges are required to execute this procedure.

Usage Notes

When you execute the procedure, the database executes the following actions:

1. Waits until all IM FastStart operations complete
2. Disables the IM FastStart feature
3. Copies IM FastStart data and metadata to the new tablespace, leaving the old tablespace intact
4. Re-enables IM FastStart the feature

Examples

The following program obtains the name of the IM FastStart tablespace, if one exists, and prints the result (sample output included):

```
VARIABLE b_fstbs VARCHAR2(20)
BEGIN
   :b_fstbs := DBMS_INMEMORY_ADMIN.GET_FASTSTART_TABLESPACE;
END;
/
PRINT b_fstbs

B_FSTBS
----------------------------
FS_TBS
```

The following statements create a new tablespace named `fs_tbs2`, and then migrate the IM FastStart area to this tablespace:

```
CREATE TABLESPACE fs_tbs2
  DATAFILE 'fs_tbs2.dbf' SIZE 500M
  EXTENT MANAGEMENT LOCAL
  SEGMENT SPACE MANAGEMENT AUTO;

EXEC DBMS_INMEMORY_ADMIN.FASTSTART_MIGRATE_STORAGE('fs_tbs2');
```

The following program prints the name of the current IM FastStart tablespace (sample output included):

```
BEGIN
   :b_fstbs := DBMS_INMEMORY_ADMIN.GET_FASTSTART_TABLESPACE;
END;
/
PRINT b_fstbs

B_FSTBS
----------------------------
FS_TBS2
```

GET_FASTSTART_TABLESPACE Function

This function returns the tablespace assigned to In-Memory FastStart (IM FastStart). If the feature is disabled, then the function returns `NOT ENABLED`.

Syntax

```
DBMS_INMEMORY_ADMIN.GET_FASTSTART_TABLESPACE();
```

Security Model

DBA privileges are required to execute this function.

Examples

This program obtains the name of the IM FastStart tablespace, if one exists, and prints the result:

```
VARIABLE b_fstbs VARCHAR2(20)
BEGIN
   :b_fstbs := DBMS_INMEMORY_ADMIN.GET_FASTSTART_TABLESPACE;
END;
/
PRINT b_fstbs

B_FSTBS
----------------------------
NOT ENABLED
```

IME_CAPTURE_EXPRESSIONS Procedure

This procedure captures the 20 most frequently accessed (“hottest”) expressions in the database in the specified time interval.

Syntax

```
DBMS_INMEMORY_ADMIN.IME_CAPTURE_EXPRESSIONS(
   snapshot     IN     VARCHAR2);
```

Parameters

Table 107-4 IME_CAPTURE_EXPRESSIONS Procedure Parameters

Parameter	Description
snapshot	Specifies a snapshot that defines the time interval in which expression statistics are considered. You can specify any of the following values: • CUMULATIVE The database considers all expression statistics since the creation of the database. • CURRENT The database considers only expression statistics from the past 24 hours. • WINDOW The database considers statistics for expressions tracked in the most recent expression capture window. The database adds hidden virtual columns for expressions tracked in the most recent window. If the capture window is currently open, then the database considers all expressions tracked in the current window up until this point, and then materializes the hottest expressions. To list the expressions that have been tracked in the current window, query DBA_EXPRESSION_STATISTICS with SNAPSHOT='WINDOW'.

Usage Notes

When you invoke this procedure, the database queries the Expression Statistics Store (ESS), and considers only expressions on tables that are at least partially populated in the IM column store. The database adds the 20 hottest expressions to their respective tables as hidden virtual columns, prefixed with the string SYS_IME, and applies the default INMEMORY column compression clause. If any SYS_IME columns added during a previous invocation are no longer in the latest top 20 list, then the database marks them as NO INMEMORY.

> **Note:**
>
> Executing the IME_CAPTURE_EXPRESSIONS procedure on a standby database has no effect.

The maximum number of SYS_IME columns for a table, regardless of whether the attribute is INMEMORY or NO INMEMORY, is 50. After the limit is reached for a table, the database will not add new SYS_IME columns. To make space for new expressions, you must manually drop SYS_IME

columns with the IME_DROP_ALL_EXPRESSIONS Procedure or IME_DROP_EXPRESSIONS Procedure.

The 50-expression limit for each table, which includes both `INMEMORY` and `NO INMEMORY` expressions, is different from the 20-expression limit for the database, which includes only `INMEMORY` expressions. For example, if 20 tables are populated in the IM column store, then each table might each have 1 `SYS_IME` column with the `INMEMORY` attribute, and 49 `SYS_IME` columns with the `NO INMEMORY` attribute.

IM expressions and virtual columns are stored in In-Memory structured called In-Memory Expression Units (IMEUs). Every IMEU is linked to a parent In-Memory Compression Unit (IMCU) from which it inherits compression characteristics.

ESS information is stored in the data dictionary and exposed in the `DBA_EXPRESSION_STATISTICS` view. This view shows the metadata that the optimizer has collected in the ESS. IM expressions are exposed as system-generated virtual columns, prefixed by the string `SYS_IME`, in the `DBA_IM_EXPRESSIONS` view.

Example 107-3 Capturing Expressions in a User-Defined Window

This example demonstrates use of the `WINDOW` capture mode. Your goal is to open and close an expression capture window, and then capture all expressions that the database tracked during this window. You perform the following steps:

1. Open an expression capture window, generate expressions, and then close the window:

```
EXEC DBMS_INMEMORY_ADMIN.IME_OPEN_CAPTURE_WINDOW();
-- Generate expressions for the database to track
EXEC DBMS_INMEMORY_ADMIN.IME_CLOSE_CAPTURE_WINDOW();
```

2. Query `DBA_EXPRESSION_STATICS` (sample output included):

```
COL OWNER FORMAT A6
COL TABLE_NAME FORMAT A9
COL COUNT FORMAT 99999
COL CREATED FORMAT A10
COL EXPRESSION_TEXT FORMAT A29

SELECT OWNER, TABLE_NAME, EVALUATION_COUNT AS COUNT,
       CREATED, EXPRESSION_TEXT
FROM   DBA_EXPRESSION_STATISTICS
WHERE  SNAPSHOT = 'WINDOW'
AND    OWNER = 'SH';

OWNER  TABLE_NAM  COUNT CREATED    EXPRESSION_TEXT
------ --------- ------ ---------- ------------------------
SH     SALES       4702 09-OCT-17  "QUANTITY_SOLD"
SH     SALES       4702 09-OCT-17  "QUANTITY_SOLD"*"AMOUNT_SOLD"
SH     SALES       4702 09-OCT-17  "PROD_ID"
SH     SALES       4702 09-OCT-17  "CUST_ID"
SH     SALES       4702 09-OCT-17  "CHANNEL_ID"
SH     SALES       4702 09-OCT-17  "AMOUNT_SOLD"
```

The preceding query shows both the columns tracked in the ESS and the expressions captured during the window for queries in the `sh` schema. During the most recent window, the database captured one expression: `QUANTITY_SOLD*AMOUNT_SOLD`.

3. Use `IME_CAPTURE_EXPRESSIONS` to make the database consider all expressions in the current window for materialization:

```
EXEC DBMS_INMEMORY_ADMIN.IME_CAPTURE_EXPRESSIONS('WINDOW');
```

4. Query `DBA_IM_EXPRESSIONS` (sample output included):

```
COL OWNER FORMAT a6
COL TABLE_NAME FORMAT a9
COL COLUMN_NAME FORMAT a25
SET LONG 50
SET LINESIZE 150

SELECT OWNER, TABLE_NAME, COLUMN_NAME, SQL_EXPRESSION
FROM   DBA_IM_EXPRESSIONS;

OWNER  TABLE_NAM COLUMN_NAME               SQL_EXPRESSION
------ --------- ------------------------- ------------------------------
SH     SALES     SYS_IME000100000025201B   "QUANTITY_SOLD"*"AMOUNT_SOLD"
```

   The preceding output shows all virtual columns that were added to the table and marked `INMEMORY` as part of the latest `IME_CAPTURE_EXPRESSIONS` invocation. The database gradually populates the captured expressions into the IM column store when it repopulates different IMCUs of the table.

5. Execute the following procedure to explicitly force a population of all captured IM expressions:

```
EXEC DBMS_INMEMORY_ADMIN.IME_POPULATE_EXPRESSIONS();
```

   Note that you can populate IM expressions from a specific table by executing the `DBMS_INMEMORY.REPOPULATE` procedure with the `force` parameter set to `TRUE`.

Example 107-4 Capturing Expressions for the Past Day

The following program captures expressions tracked during the last 24 hours:

```
EXEC DBMS_INMEMORY_ADMIN.IME_CAPTURE_EXPRESSIONS('CURRENT');
```

IME_CLOSE_CAPTURE_WINDOW Procedure

This procedure signals the end of the current expression capture window.

Syntax

```
DBMS_INMEMORY_ADMIN.IME_CLOSE_CAPTURE_WINDOW();
```

Usage Notes

On invocation of this procedure, the optimizer saves all gathered statistics to disk, and essentially freezes the expressions tracked in the window. The database preserves the statistics captured in this window until a new expression capture window is opened, at which point the database purges the statistics captured in the previous window.

Example 107-5 Example

This example opens an expression capture window, and then issues `IME_CAPTURE_EXPRESSIONS('WINDOW')` so that the database considers all expressions in the current window for materialization. Finally, the example closes the window.

```
EXEC DBMS_INMEMORY_ADMIN.IME_OPEN_CAPTURE_WINDOW();
-- Generate expressions so that the database can track them
EXEC DBMS_INMEMORY_ADMIN.IME_CLOSE_CAPTURE_WINDOW();
EXEC DBMS_INMEMORY_ADMIN.IME_CAPTURE_EXPRESSIONS('WINDOW');
```

IME_DROP_ALL_EXPRESSIONS Procedure

This procedure drops all `SYS_IME` expression virtual columns in the database.

Syntax

```
DBMS_INMEMORY_ADMIN.IME_DROP_ALL_EXPRESSIONS();
```

Usage Notes

The `IME_DROP_ALL_EXPRESSIONS` procedure drops all `SYS_IME` columns from all tables, regardless of whether they have the `INMEMORY` attribute. In effect, the procedure acts as a database-wide reset button.

Using `IME_DROP_ALL_EXPRESSIONS` triggers a drop of all IMEUs *and* IMCUs for segments that have `SYS_IME` columns. For example, if 50 populated tables have one `SYS_IME` column each, then `IME_DROP_ALL_EXPRESSIONS` removes all 50 tables from the IM column store. To populate these segments again, you must use the `DBMS_INMEMORY.POPULATE` procedure or perform a full table scan.

IME_GET_CAPTURE_STATE Procedure

This procedure returns the current capture state of the expression capture window and the timestamp of the most recent modification.

Syntax

```
DBMS_INMEMORY_ADMIN.IME_GET_CAPTURE_STATE(
   p_capture_state  OUT  VARCHAR2,
   p_last_modified  OUT  TIMESTAMP);
```

Parameters

Parameter	Description
`p_capture_state`	Describes the current state of the expression capture window. The following states are possible: • `OPEN` — Indicates that the window is open. • `CLOSED` — Indicates that the window is closed. • `DEFAULT` — Indicates that the window has not been used. It is equivalent to the `CLOSED` state.
`p_last_modified`	Indicates the timestamp of the most recent action.

Usage Notes

This procedure is useful for avoiding conflicting calls for IME_OPEN_CAPTURE_WINDOW Procedure and IME_CLOSE_CAPTURE_WINDOW Procedure. For example, if the current expression capture window state is `OPEN`, then you cannot open another window, and if the window state is `CLOSED`, then you cannot close a window.

Example 107-6 Determining the State of an Expression Capture Window

This example opens an expression capture window, and then determines its capture state.

```
EXEC DBMS_INMEMORY_ADMIN.IME_OPEN_CAPTURE_WINDOW();

VARIABLE b_state VARCHAR2(25)
VARIABLE b_time  VARCHAR2(10)
EXECUTE DBMS_INMEMORY_ADMIN.IME_GET_CAPTURE_STATE(:b_state, :b_time)
PRINT b_state b_time
```

The following sample output indicates that an expression capture window is currently open:

```
B_STATE
-------------------------------------------------
OPEN

B_TIME
-------------------------------------------------
09-OCT-17
```

IME_OPEN_CAPTURE_WINDOW Procedure

This procedure signals the beginning of an expression capture window.

Syntax

```
DBMS_INMEMORY_ADMIN.IME_OPEN_CAPTURE_WINDOW();
```

Security Model

Administrator privileges are required to execute this procedure.

Usage Notes

On invocation of this procedure, the optimizer begins a new window snapshot and starts tracking expressions that occur within this window. An expression capture window is global across all instances in an Oracle RAC database.

Conflicting actions are not permitted. For example, in an Oracle RAC database, opening expression capture window on instance 1 at time t0 and attempting to open another expression capture window on instance 2 at time t1 before closing the first window is a conflicting action. To obtain the current capture state and reduce the potential for conflicting procedure invocations, use the IME_GET_CAPTURE_STATE Procedure.

Example

This following program opens an expression capture window:

```
EXEC DBMS_INMEMORY_ADMIN.IME_OPEN_CAPTURE_WINDOW();
```

IME_POPULATE_EXPRESSIONS Procedure

This procedure forces the population of expressions captured in the latest invocation of `DBMS_INMEMORY_ADMIN.IME_CAPTURE_EXPRESSIONS`.

Syntax

```
DBMS_INMEMORY_ADMIN.IME_POPULATE_EXPRESSIONS();
```

Usage Notes

If you do not invoke this procedure, then the database gradually repopulates `SYS_IME` columns when their parent IMCUs are repopulated. If a table is not repopulated, then any new `SYS_IME` columns captured by the `IME_CAPTURE_EXPRESSIONS` procedure are not populated. `IME_POPULATE_EXPRESSIONS` solves this problem by forcing population.

Internally, the procedure invokes `DBMS_INMEMORY.REPOPULATE` for all tables that have `SYS_IME` columns with the `INMEMORY` attribute. To populate `SYS_IME` columns in a specified subset of tables, use `DBMS_INMEMORY.REPOPULATE` instead of `IME_POPULATE_EXPRESSIONS`.

POPULATE_WAIT Function

This function initiates population of all `INMEMORY` objects that have a priority greater than or equal to the specified priority, and returns a status value for the population. A user-specified interval specifies the maximum time that the function waits before returning the value to the caller.

Syntax

```
DBMS_INMEMORY_ADMIN.POPULATE_WAIT(
   priority    IN    VARCHAR2 DEFAULT 'LOW',
   percentage  IN    NUMBER   DEFAULT 100,
   timeout     IN    NUMBER   DEFAULT 99999999,
   force       IN    BOOLEAN DEFAULT FALSE)
RETURN VARCHAR2;
```

Parameters

Table 107-5 POPULATE_WAIT Function Parameters

Parameter	Description
`priority`	Specifies that the database populate all `INMEMORY` objects with the specified priority setting or higher. The default priority is `LOW`. `NONE` is considered lowest priority. If you set to `priority` to `NONE`, then this function waits for all `INMEMORY` objects to populate.

Table 107-5 (Cont.) POPULATE_WAIT Function Parameters

Parameter	Description
`percentage`	Specifies the percentage of population required for the function to consider population to be complete. The default is `100`. For example, if `percentage` is `50` and `priority` is `NONE`, and if 50% of the `INMEMORY` objects are populated in the IM column store, then the function returns the value `0` (population successful).
`timeout`	Specifies the number of seconds that must pass before the function returns `-1`, which indicates that the populate operation timed out. The default is `99999999` seconds, which is 115.74 days. Assume that `timeout` is `600`, `priority` is `LOW`, and `percentage` is `100`. If 10 minutes pass, but all `PRIORITY LOW` objects are not yet fully populated, then the function returns `-1`.
`force`	Specifies that the database should drop all `INMEMORY` segments that have a priority greater than or equal than the specified priority, and then repopulate these segments. The default is `FALSE`. Assume that the `INMEMORY` attribute applies to the `sales` table, which is partitioned. Only half the `sales` partitions are currently populated in the IM column store. If you execute `POPULATE_WAIT` with `force` set to `TRUE`, then the database drops all `sales` segments, and then repopulates them.

Return Values

The following table describes the possible return values for `POPULATE_WAIT`. The function returns the values `0`, `1`, `2`, and `3` only if the condition is met before the end of the interval specified by `timeout`. For example, if `timeout` is `600`, then the function returns `1` only if an out-of-memory error occurs before 600 seconds pass. The function returns `-1` only if the end of the timeout interval occurs *before* the database completes the requested operation.

Table 107-6 Return Values for POPULATE_WAIT

Constant	Value	Description
`POPULATE_TIMEOUT`	`-1`	The function timed out while waiting for population to complete. Existing population jobs continue running in the background after `-1` is returned. Reissuing `POPULATE_TIMEOUT` after `-1` is returned reinitiates population; segments that are already populated are not dropped.
`POPULATE_SUCCESS`	`0`	All objects that met the `priority` criteria were populated to the specified `percentage` of completion.
`POPULATE_OUT_OF_MEMORY`	`1`	The In-Memory pool had insufficient memory to populate the objects that met the `priority` criteria to the specified `percentage` of completion.
`POPULATE_NO_INMEMORY_OBJECTS`	`2`	No `INMEMORY` objects met the specified `priority` criteria.

Table 107-6 (Cont.) Return Values for POPULATE_WAIT

Constant	Value	Description
POPULATE_INMEMORY_SIZE_ZERO	3	The In-Memory column store is not enabled.

Usage Notes

Sample use cases for ensuring that objects are populated include:

- When the database is closed, open the database with `STARTUP RESTRICT` so that only administrators can access the database, and then execute `POPULATE_WAIT` with the desired timeout setting. If `POPULATE_WAIT` returns `-1`, indicating a timeout, then reexecute `POPULATE_WAIT`. When the function returns `0`, disable the restricted session so that non-administrative users can query the database.
- Block database connections by using services or an application tier technique. When no analytic indexes exists, and when the application depends on the IM column store to provide reasonable performance, these techniques prevent runaway queries.

Example 107-7 Specifying a Timeout Interval for In-Memory Population

In this example, the database contains a number of In-Memory tables with a variety of priority settings. Your goal is to populate *every* In-Memory table to 100% completion in a restricted database session, and then disable the restricted session so that the application can be guaranteed of querying only the In-Memory representations.

Assume that the database is shut down. In SQL*Plus, you connect to an idle instance as `SYSDBA`, and then execute the following command (sample output included):

```
SQL> STARTUP RESTRICT
ORACLE instance started.

Total System Global Area 1157624280 bytes
Fixed Size                  8839640 bytes
Variable Size             754974720 bytes
Database Buffers           16777216 bytes
Redo Buffers                7933952 bytes
In-Memory Area            369098752 bytes
Database mounted.
Database opened.
```

The database is open, but is accessible only to administrative users. You execute the following statements in SQL*Plus (sample output shown in bold):

```
VARIABLE b_pop_status NUMBER

SELECT DBMS_INMEMORY_ADMIN.POPULATE_WAIT(
         priority   => 'NONE' ,
         percentage => 100    ,
         timeout    => 300    )
  INTO b_pop_status
FROM   DUAL;
```

```
PRINT b_pop_status
-1
```

After 5 minutes, the function returns the number `-1`. This code indicates that the function timed out while waiting for population to complete. 5 minutes is not long enough to populate all `INMEMORY` tables. You re-execute the `SELECT` statement, specifying a 30-minute timeout:

```
SELECT DBMS_INMEMORY_ADMIN.POPULATE_WAIT(
         priority   => 'NONE' ,
         percentage => 100    ,
         timeout    => 1800   )
  INTO b_pop_status
FROM   DUAL;

PRINT b_pop_status
0
```

After 8 minutes, the function returns the number `0`. This code indicates that all tables are completely populated. You now disable the restricted session so that the application can start query In-Memory objects with full confidence that only In-Memory representations will be accessed:

```
ALTER SYSTEM DISABLE RESTRICTED SESSION;
```