52 DBMS_COMPARISON

The DBMS_COMPARISON package provides interfaces to compare and converge database objects at different databases.

This chapter contains the following topics:

DBMS_COMPARISON Overview

The DBMS_COMPARISON package is an Oracle-supplied package that you can use to compare database objects at two databases. This package also enables you converge the database objects so that they are consistent at different databases. Typically, this package is used in environments that share a database object at multiple databases. When copies of the same database object exist at multiple databases, the database object is a **shared database object**. Several data dictionary views contain information about comparisons made with the DBMS_COMPARISON package.

Shared database objects might be maintained by data replication. For example, materialized views or Oracle Streams components might replicate the database objects and maintain them at multiple databases. A custom application might also maintain shared database objects. When a database object is shared, it can diverge at the databases that share it. You can use this package to identify differences in the shared database objects. After identifying the differences, you can optionally use this package to synchronize the shared database objects.

To compare a database object that is shared at two different databases, complete the following general steps:

1. Run the CREATE_COMPARISON Procedure in this package to create a **comparison**. The comparison identifies the database objects to compare and specifies parameters for the comparison.
2. Run the COMPARE Function in this package to compare the database object at the two databases and identify differences. This function returns TRUE when no differences are found and FALSE when differences are found. This function also populates data dictionary views with comparison results. Separate comparison results are generated for each execution of the COMPARE function.
3. If you want to examine the comparison results, query the following data dictionary views:
   - DBA_COMPARISON_SCAN
   - USER_COMPARISON_SCAN

- `DBA_COMPARISON_SCAN_VALUES`
- `USER_COMPARISON_SCAN_VALUES`
- `DBA_COMPARISON_ROW_DIF`
- `USER_COMPARISON_ROW_DIF`

4. If there are differences, and you want to synchronize the database objects at the two databases, then run the `CONVERGE` procedure in this package.

After you create a comparison with the `CREATE_COMPARISON` procedure in the `DBMS_COMPARISON` package, you can run the comparison at any time using the `COMPARE` function. Each time you run the `COMPARE` function, it records comparison results in the appropriate data dictionary views. Comparison results might be modified when subprograms in this package are invoked and the scans in the comparison results are specified. For example, comparison results might be modified when you run the `RECHECK` function.

The comparison results for a single execution of the `COMPARE` function can include one or more **scans**. A scan checks for differences in some or all of the rows in a shared database object at a single point in time. You can compare database objects multiple times, and a unique scan ID identifies each scan in the comparison results.

A **bucket** is a range of rows in a database object that is being compared. Buckets improve performance by splitting the database object into ranges and comparing the ranges independently. Every comparison divides the rows being compared into an appropriate number of buckets, and each bucket is compared by a scan.

Each time the `COMPARE` function splits a bucket into smaller buckets, it performs new scans of the smaller buckets. The scan that analyzes a larger bucket is the **parent scan** of each scan that analyzes the smaller buckets into which the larger bucket was split. The **root scan** in the comparison results is the highest level parent scan. The root scan does not have a parent.

You can recheck a scan using the `RECHECK` function, and you can converge a scan using the `CONVERGE` procedure. When you want to recheck or converge all of the rows comparison results, specify the root scan ID for the comparison results in the appropriate subprogram. When you want to recheck or converge a portion of the rows in comparison results, specify the scan ID of the scan that contains the differences.

DBMS_COMPARISON Security Model

Security on this package can be controlled in one of two ways.

- Granting `EXECUTE` on this package to selected users or roles.
- Granting `EXECUTE_CATALOG_ROLE` to selected users or roles.

If subprograms in the package are run from within a stored procedure, then the user who runs the subprograms must be granted `EXECUTE` privilege on the package directly. It cannot be granted through a role.

Each subprogram in the `DBMS_COMPARISON` package has a `comparison_name` parameter. The current user must be the owner of the specified comparison to run a subprogram in the `DBMS_COMPARISON` package.

To run the `COMPARE` function, `RECHECK` function, or `CONVERGE` procedure, the following users must have the `SELECT` or `READ` privilege on each copy of the shared database object:

- The comparison owner at the local database

- When a database link is used, the user at the remote database to which the comparison owner connects through a database link

The `CONVERGE` procedure also requires additional privileges for one of these users at the database where it makes changes to the shared database object. The user must have `INSERT`, `UPDATE`, and `DELETE` privileges on the shared database object at this database.

In addition, when the `CONVERGE` procedure is run with either the `local_converge_tag` or `remote_converge_tag` parameter set to a non-`NULL` value, then the following additional requirements must be met:

- If the local table "wins," then the user at the remote database to which the invoker of the `CONVERGE` procedure connects through a database link must be granted either `EXECUTE_CATALOG_ROLE` or `EXECUTE` privilege on the `DBMS_STREAMS_ADM` package.
- If the remote table "wins," then the invoker of the `CONVERGE` procedure at the local database must be granted either `EXECUTE_CATALOG_ROLE` or `EXECUTE` privilege on the `DBMS_STREAMS_ADM` package.

Note:

The database administrator (DBA) can assume control over some of the `DBMS_COMPARISON` functions and procedures owned by other users. This control applies to `DROP_COMPARISON` and `PURGE_COMPARISON`. This DBA override can be particularly useful in cleanup operations when comparisons created by another user need to be dropped

DBMS_COMPARISON Constants

The `DBMS_COMPARISON` package defines several enumerated constants to use specifying parameter values. Enumerated constants must be prefixed with the package name. For example, `DBMS_COMPARISON.CMP_SCAN_MODE_FULL`.

Table 52-1 lists the parameters and enumerated constants.

Table 52-1 DBMS_COMPARISON Parameters with Enumerated Constants

Parameter	Option	Type	Description
`comparison_mode`	• `CMP_COMPARE_MODE_OBJECT`	`VARCHAR2(30)`	`CMP_COMPARE_MODE_OBJECT` is a database object. This constant can be specified as `'OBJECT'`.

Table 52-1 (Cont.) DBMS_COMPARISON Parameters with Enumerated Constants

Parameter	Option	Type	Description
scan_mode	• CMP_SCAN_MODE_FULL • CMP_SCAN_MODE_RANDOM • CMP_SCAN_MODE_CYCLIC • CMP_SCAN_MODE_CUSTOM	VARCHAR2(30)	CMP_SCAN_MODE_FULL indicates that the entire database object is compared. This constant can be specified as 'FULL'. CMP_SCAN_MODE_RANDOM indicates that a random portion of the database object is compared. This constant can be specified as 'RANDOM'. CMP_SCAN_MODE_CYCLIC indicates that a portion of the database object is compared when you perform a single comparison. When you compare the database object again, another portion of the database object is compared, starting where the last comparison ended. This constant can be specified as 'CYCLIC'. CMP_SCAN_MODE_CUSTOM indicates that the user who runs the subprogram specifies the range to compare in the database object. This constant can be specified as 'CUSTOM'.
converge_options	• CMP_CONVERGE_LOCAL_WINS • CMP_CONVERGE_REMOTE_WINS	VARCHAR2(30)	CMP_CONVERGE_LOCAL_WINS indicates that the column values at the local database replace the column values at the remote database when these column values are different. This constant can be specified as 'LOCAL'. CMP_CONVERGE_REMOTE_WINS indicates that the column values at the remote database replace the column values at the local database when these column values are different. This constant can be specified as 'REMOTE'.
null_value	• CMP_NULL_VALUE_DEF	VARCHAR2(100)	CMP_NULL_VALUE_DEF indicates that ORA$STREAMS$NV is substituted for NULL values in database objects during comparison. This constant can be specified as 'ORA$STREAMS$NV'.
max_num_buckets	• CMP_MAX_NUM_BUCKETS	INTEGER	CMP_MAX_NUM_BUCKETS indicates that the maximum number of buckets is 1,000. This constant can be specified as 1000.
min_rows_in_bucket	• CMP_MIN_ROWS_IN_BUCKET	INTEGER	CMP_MIN_ROWS_IN_BUCKET indicates that the minimum number of rows in a bucket is 10,000. This constant can be specified as 10000.

DBMS_COMPARISON Views

The `DBMS_COMPARISON` package uses several views.

These views are listed below:

- `DBA_COMPARISON`
- `USER_COMPARISON`
- `DBA_COMPARISON_COLUMNS`
- `USER_COMPARISON_COLUMNS`
- `DBA_COMPARISON_SCAN`
- `USER_COMPARISON_SCAN`
- `DBA_COMPARISON_SCAN_VALUES`
- `USER_COMPARISON_SCAN_VALUES`
- `DBA_COMPARISON_ROW_DIF`
- `USER_COMPARISON_ROW_DIF`

See Also:

Oracle Database Reference

DBMS_COMPARISON Operational Notes

The `DBMS_COMPARISON` package has certain requirements and operational notes.

These include the following:

- Oracle Database Release Requirements for the DBMS_COMPARISON Package
- Database Character Set Requirements for the DBMS_COMPARISON Package
- Database Object Requirements for the DBMS_COMPARISON Package
- Index Column Requirements for the DBMS_COMPARISON Package
- Datatype Requirements for the DBMS_COMPARISON Package
- Only Converge Rows That Are Not Being Updated

Oracle Database Release Requirements for the DBMS_COMPARISON Package

Meet the following Oracle Database release requirements when running the subprograms in the `DBMS_COMPARISON` package:

- The local database that runs the subprograms in the `DBMS_COMPARISON` package must be an Oracle Database 11*g* Release 1 (11.1) database.
- The remote database must be an Oracle Database 10*g* Release 1 (10.1) or later database. Oracle databases before this release and non-Oracle databases are not supported.

Database Character Set Requirements for the DBMS_COMPARISON Package

The database character sets must be the same for the databases that contain the database objects being compared.

> **See Also:**
>
> *Oracle Database Globalization Support Guide* for information about database character sets

Database Object Requirements for the DBMS_COMPARISON Package

The `DBMS_COMPARISON` package can compare the following types of database objects:

- Tables
- Single-table views
- Materialized views
- Synonyms for tables, single-table views, and materialized views

Database objects of different types can be compared and converged at different databases. For example, a table at one database and a materialized view at another database can be compared and converged with this package.

To run the subprograms in the `DBMS_COMPARISON` package, the specified database objects must have the same shape at each database. Specifically, the database objects must have the same number of columns at each database, and the datatypes of corresponding columns must match.

If a database object being compared contains columns that do not exist in the other database object, then you can compare the database objects by excluding the extra columns during comparison creation. Use the `column_list` parameter in the `CREATE_COMPARISON` procedure to list only the columns that exist in both database objects.

> **See Also:**
>
> CREATE_COMPARISON Procedure

Index Column Requirements for the DBMS_COMPARISON Package

This section discusses number, timestamp, and interval columns. These include the following datatypes:

- Number columns are of the following datatypes: `NUMBER`, `FLOAT`, `BINARY_FLOAT`, and `BINARY_DOUBLE`.
- Timestamp columns are of the following datatypes: `TIMESTAMP`, `TIMESTAMP WITH TIME ZONE`, and `TIMESTAMP WITH LOCAL TIME ZONE`
- Interval columns are of the following datatypes: `INTERVAL YEAR TO MONTH` and `INTERVAL DAY TO SECOND`.

For all scan modes to be supported by the DBMS_COMPARISON package, the database objects must have one of the following types of indexes:

- A single-column index on a number, timestamp, interval, or DATE datatype column
- A composite index that only includes number, timestamp, interval, or DATE datatype columns. Each column in the composite index must either have a NOT NULL constraint or be part of the primary key.

For the scan modes CMP_SCAN_MODE_FULL and CMP_SCAN_MODE_CUSTOM to be supported, the database objects must have one of the following types of indexes:

- A single-column index on a number, timestamp, interval, DATE, VARCHAR2, or CHAR datatype column
- A composite index that only includes number, timestamp, interval, DATE, VARCHAR2, or CHAR columns. Each column in the composite index must either have a NOT NULL constraint or be part of the primary key.

If the database objects do not have one of these types of indexes, then the DBMS_COMPARISON package does not support the database objects. For example, if the database objects only have a single index on an NVARCHAR2 column, then the DBMS_COMPARISON package does not support them. Or, if the database objects have only one index, and it is a composite index that includes a NUMBER column and an NCHAR column, then the DBMS_COMPARISON package does not support them.

You can specify an index when you create a comparison using the index_schema_name and index_name parameters in the CREATE_COMPARISON procedure. If you specify an index, then make sure the columns in the index meet the requirements of the scan mode used for the comparison.

The index columns in a comparison must uniquely identify every row involved in a comparison. The following constraints satisfy this requirement:

- A primary key constraint
- A unique constraint on one or more non-NULL columns

If these constraints are not present on a table, then use the index_schema_name and index_name parameters in the CREATE_COMPARISON procedure to specify an index whose columns satisfy this requirement.

When a single index value identifies both a local row and a remote row, the two rows must be copies of the same row in the replicated tables. In addition, each pair of copies of the same row must always have the same index value.

The DBMS_COMPARISON package can use an index only if all of the columns in the index are included in the column_list parameter when the comparison is created with the CREATE_COMPARISON procedure.

After a comparison is created, you can determine the index column or columns for it by running the following query:

```
SELECT COLUMN_NAME, COLUMN_POSITION FROM DBA_COMPARISON_COLUMNS
  WHERE COMPARISON_NAME = 'COMPARE_CUSTOM' AND
        INDEX_COLUMN    = 'Y';
```

If there are multiple index columns, then the index column with 1 for the COLUMN_POSITION is the lead index column in the composite index.

See Also:

- "Constants" for information about scan modes
- CREATE_COMPARISON Procedure for information about specifying an index for a comparison

Datatype Requirements for the DBMS_COMPARISON Package

The `DBMS_COMPARISON` package can compare data in columns of the following datatypes:

- `VARCHAR2`
- `NVARCHAR2`
- `NUMBER`
- `FLOAT`
- `DATE`
- `BINARY_FLOAT`
- `BINARY_DOUBLE`
- `TIMESTAMP`
- `TIMESTAMP WITH TIME ZONE`
- `TIMESTAMP WITH LOCAL TIME ZONE`
- `INTERVAL YEAR TO MONTH`
- `INTERVAL DAY TO SECOND`
- `RAW`
- `CHAR`
- `NCHAR`

If a column with datatype `TIMESTAMP WITH LOCAL TIME ZONE` is compared, then the two databases must use the same time zone. Also, if a column with datatype `NVARCHAR2` or `NCHAR` is compared, then the two databases must use the same national character set.

The `DBMS_COMPARISON` package cannot compare data in columns of the following datatypes:

- `LONG`
- `LONG RAW`
- `ROWID`
- `UROWID`
- `CLOB`
- `NCLOB`
- `BLOB`
- `BFILE`
- User-defined types (including object types, `REF`s, varrays, and nested tables)

- Oracle-supplied types (including any types, XML types, spatial types, and media types)

You can compare database objects that contain unsupported columns by excluding the unsupported columns during comparison creation. Use the `column_list` parameter in the `CREATE_COMPARISON` procedure to list only the supported columns in a shared database object.

See Also:

- CREATE_COMPARISON Procedure
- *Oracle Database SQL Language Reference* for more information about datatypes
- *Oracle Database Globalization Support Guide* for information about national character sets

Only Converge Rows That Are Not Being Updated

You should only converge rows that are not being updated on either database. For example, if the shared database object is updated by replication components, then only converge rows for which replication changes have been applied and make sure no new changes are in the process of being replicated for these rows. If you compare replicated database objects, then it is typically best to compare them during a time of little or no replication activity to identify persistent differences.

Note:

If a scan identifies that a row is different in the shared database object at two databases, and the row is modified after the scan, then it can result in unexpected data in the row after the `CONVERGE` procedure is run.

DBMS_COMPARISON Data Structures

The `DBMS_COMPARISON` package defines a `RECORD` type.

Contains information returned by the `COMPARE` function or `CONVERGE` procedure in the `DBMS_COMPARISON` package.

Note:

The `COMPARE` function only returns a value for the `scan_id` field.

COMPARISON_TYPE Record Type

This record type contains information returned by the `COMPARE` function or `CONVERGE` procedure in the `DBMS_COMPARISON` package.

> **Note:**
>
> The `COMPARE` function only returns a value for the `scan_id` field.

Syntax

```
TYPE COMPARISON_TYPE IS RECORD(
  scan_id             NUMBER,
  loc_rows_merged     NUMBER,
  rmt_rows_merged     NUMBER,
  loc_rows_deleted    NUMBER,
  rmt_rows_deleted    NUMBER);
```

Table 52-2 COMPARISON_TYPE Attributes

Field	Description
`scan_id`	The scan ID of the scan
`loc_rows_merged`	The number of rows in the local database object updated with information from the database object at the remote site
`rmt_rows_merged`	The number of rows in the database object updated at the remote site with information from the database object at the local site
`loc_rows_deleted`	The number of rows deleted from the local database object
`rmt_rows_deleted`	The number of rows deleted from the remote database object

Summary of DBMS_COMPARISON Subprograms

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

Table 52-3 DBMS_COMPARISON Package Subprograms

Subprogram	Description
COMPARE Function	Performs the specified comparison
CONVERGE Procedure	Executes data manipulation language (DML) changes to synchronize the portion of the database object that was compared in the specified scan
CREATE_COMPARISON Procedure	Creates a comparison
DROP_COMPARISON Procedure	Drops a comparison
PURGE_COMPARISON Procedure	Purges the comparison results, or a subset of the comparison results, for a comparison
RECHECK Function	Rechecks the differences in a specified scan for a comparison

COMPARE Function

This function performs the specified comparison.

Each time a comparison is performed, it results in at least one new scan, and each scan has a unique scan ID. You can define and name a comparison using the CREATE_COMPARISON procedure.

See Also:

- "Overview"
- CREATE_COMPARISON Procedure

Syntax

```
DBMS_COMPARISON.COMPARE(
   comparison_name  IN   VARCHAR2,
   scan_info        OUT  COMPARISON_TYPE,
   min_value        IN   VARCHAR2    DEFAULT NULL,
   max_value        IN   VARCHAR2    DEFAULT NULL,
   perform_row_dif  IN   BOOLEAN     DEFAULT FALSE)
RETURN BOOLEAN;
```

Parameters

Table 52-4   COMPARE Function Parameters

Parameter	Description
comparison_name	The name of the comparison.
scan_info	Information about the compare operation returned in the COMPARISON_TYPE datatype. See COMPARISON_TYPE Record Type.
min_value	When the scan mode for the comparison is set to CMP_SCAN_MODE_CUSTOM, specify the minimum index column value for the range of rows that are being compared. To determine the index column for a comparison, query the DBA_COMPARISON_COLUMNS data dictionary view. For a composite index, specify a value for the column with column_position equal to 1 in the DBA_COMPARISON_COLUMNS view. See the index column requirements under DBMS_COMPARISON Operational Notes. If the scan mode is set to a value other than CMP_SCAN_MODE_CUSTOM, then this parameter must be set to NULL. If NULL and the scan_mode parameter is set to CMP_SCAN_MODE_CUSTOM, then an error is raised. To determine the scan mode for the comparison, query the DBA_COMPARISON data dictionary view. See DBMS_COMPARISON Constants for information about scan modes.

Table 52-4 (Cont.) COMPARE Function Parameters

Parameter	Description
`max_value`	When the scan mode for the comparison is set to `CMP_SCAN_MODE_CUSTOM`, specify the maximum index column value for the range of rows that are being compared. To determine the index column for a comparison, query the `DBA_COMPARISON_COLUMNS` data dictionary view. For a composite index, specify a value for the column with `column_position` equal to `1` in the `DBA_COMPARISON_COLUMNS` view. See the index column requirements under DBMS_COMPARISON Operational Notes. If the scan mode is set to a value other than `CMP_SCAN_MODE_CUSTOM`, then this parameter must be set to `NULL`. If `NULL` and the scan_mode parameter is set to `CMP_SCAN_MODE_CUSTOM`, then an error is raised. To determine the scan mode for the comparison, query the `DBA_COMPARISON` data dictionary view. See DBMS_COMPARISON Constants for information about scan modes.
`perform_row_dif`	If `TRUE`, then compares each row individually in the database object being compared after reaching the smallest possible bucket for the comparison. If `FALSE`, then compares buckets for differences but does not compare each row individually when differences are found in the smallest possible bucket. See DBMS_COMPARISON Overview for information about buckets.

Return Values

This function returns `TRUE` when no differences are found in the database objects being compared. This function returns `FALSE` when differences are found in the database objects being compared.

CONVERGE Procedure

This procedure executes data manipulation language (DML) changes to synchronize the portion of the database objects that was compared in the specified scan.

Syntax

```
DBMS_COMPARISON.CONVERGE(
   comparison_name      IN   VARCHAR2,
   scan_id              IN   NUMBER,
   scan_info            OUT  COMPARISON_TYPE,
   converge_options     IN   VARCHAR2  DEFAULT CMP_CONVERGE_LOCAL_WINS,
   perform_commit       IN   BOOLEAN   DEFAULT TRUE,
   local_converge_tag   IN   RAW       DEFAULT NULL,
   remote_converge_tag  IN   RAW       DEFAULT NULL);
```

Parameters

Table 52-5 CONVERGE Procedure Parameters

Parameter	Description
`comparison_name`	The name of the comparison.

Table 52-5 (Cont.) CONVERGE Procedure Parameters

Parameter	Description
`scan_id`	The identifier for the scan that contains the differences between the database objects being converged. See "Overview" for more information about specifying a scan ID in this parameter.
`scan_info`	Information about the converge operation returned in the `COMPARISON_TYPE` datatype. See COMPARISON_TYPE Record Type.
`converge_options`	Either the `CMP_CONVERGE_LOCAL_WINS` constant or the `CMP_CONVERGE_REMOTE_WINS` constant. See "Constants" for information about these constants.
`perform_commit`	If `TRUE`, then performs a `COMMIT` periodically while making the DML changes. The `CONVERGE` procedure might perform more than one `COMMIT` when this parameter is set to `TRUE`. If `FALSE`, then does not perform a `COMMIT` after making DML changes.
`local_converge_tag`	The Replication tag to set in the session on the local database before performing any changes to converge the data in the database objects being converged. If non-`NULL`, then this parameter setting takes precedence over the `local_converge_tag` parameter in the `CREATE_COMPARISON` procedure that created the comparison. If `NULL`, then this parameter is ignored, and the `local_converge_tag` parameter in the `CREATE_COMPARISON` procedure that created the comparison is used.
`remote_converge_tag`	The Replication tag to set in the session on the remote database before performing any changes to converge the data in the database objects being converged. If non-`NULL`, then this parameter setting takes precedence over the `remote_converge_tag` parameter in the `CREATE_COMPARISON` procedure that created the comparison. If `NULL`, then this parameter is ignored, and the `remote_converge_tag` parameter in the `CREATE_COMPARISON` procedure that created the comparison is used.

Usage Notes

If one of the database objects being converged is a read-only materialized view, then the `converge_options` parameter must be set to ensure that the read-only materialized view "wins" in the converge operation. The `CONVERGE` procedure raises an error if it tries to make changes to a read-only materialized view.

CREATE_COMPARISON Procedure

This procedure creates a comparison.

Syntax

```
DBMS_COMPARISON.CREATE_COMPARISON(
   comparison_name       IN  VARCHAR2,
   schema_name           IN  VARCHAR2,
   object_name          IN  VARCHAR2,
   dblink_name          IN  VARCHAR2,
   index_schema_name    IN  VARCHAR2  DEFAULT NULL,
   index_name           IN  VARCHAR2  DEFAULT NULL,
   remote_schema_name   IN  VARCHAR2  DEFAULT NULL,
   remote_object_name   IN  VARCHAR2  DEFAULT NULL,
   comparison_mode      IN  VARCHAR2  DEFAULT CMP_COMPARE_MODE_OBJECT,
   column_list          IN  VARCHAR2  DEFAULT '*',
   scan_mode            IN  VARCHAR2  DEFAULT CMP_SCAN_MODE_FULL,
   scan_percent         IN  NUMBER    DEFAULT NULL,
   null_value           IN  VARCHAR2  DEFAULT CMP_NULL_VALUE_DEF,
   local_converge_tag   IN  RAW       DEFAULT NULL,
   remote_converge_tag  IN  RAW       DEFAULT NULL,
   max_num_buckets      IN  NUMBER    DEFAULT CMP_MAX_NUM_BUCKETS,
   min_rows_in_bucket   IN  NUMBER    DEFAULT CMP_MIN_ROWS_IN_BUCKET);
```

Parameters

Table 52-6 CREATE_COMPARISON Procedure Parameters

Parameter	Description
`comparison_name`	The name of the comparison.
`schema_name`	The name of the schema that contains the local database object to compare.
`object_name`	The name of the local database object to compare.
`dblink_name`	Database link to the remote database. The specified database object in the remote database is compared with the database object in the local database. If `NULL`, then the comparison is configured to compare two database objects in the local database. In this case, parameters that specify the remote database object apply to the second database object in the comparison and to operations on the second database object. For example, specify the second database object in this procedure by using the `remote_schema_name` and `remote_object_name` parameters.
`index_schema_name`	The name of the schema that contains the index. If `NULL`, then the schema specified in the `schema_name` parameter is used.
`index_name`	The name of the index. If `NULL`, then the system determines the index columns for the comparison automatically. If the `index_schema_name` parameter is non-`NULL`, then the `index_name` parameter must also be non-`NULL`. Otherwise, an error is raised. **See Also:** "Usage Notes" for more information about specifying an index
`remote_schema_name`	The name of the schema that contains the database object at the remote database. Specify a non-`NULL` value if the schema names are different at the two databases. If `NULL`, then the schema specified in the `schema_name` parameter is used.

Table 52-6 (Cont.) CREATE_COMPARISON Procedure Parameters

Parameter	Description
`remote_object_name`	The name of the database object at the remote database. Specify a non-`NULL` value if the database object names are different at the two databases. If `NULL`, then the database object specified in the `object_name` parameter is used.
`comparison_mode`	Specify the default value `CMP_COMPARE_MODE_OBJECT`. Additional modes might be added in future releases.
`column_list`	Specify `'*'` to include all of the columns in the database objects being compared. To compare a subset of columns in the database objects, specify a comma-delimited list of the columns to check. Any columns that are not in the list are ignored during a comparison and convergence. See "Usage Notes" for information about columns that are required in the `column_list` parameter.
`scan_mode`	Either `CMP_SCAN_MODE_FULL`, `CMP_SCAN_MODE_RANDOM`, `CMP_SCAN_MODE_CYCLIC`, or `CMP_SCAN_MODE_CUSTOM`. If you specify `CMP_SCAN_MODE_CUSTOM`, then make sure you specify an index using the `index_schema_name` and `index_name` parameters. Specifying an index ensures that you can specify the correct `min_value` and `max_value` for the lead index column when you run the `COMPARE` or `RECHECK` function. See "Constants" for information about these constants.
`scan_percent`	The percentage of the database object to scan for comparison when the `scan_mode` parameter is set to either `CMP_SCAN_MODE_RANDOM` or `CMP_SCAN_MODE_CYCLIC`. For these `scan_mode` settings, a non-`NULL` value that is greater than 0 (zero) and less than 100 is required. If `NULL` and the `scan_mode` parameter is set to `CMP_SCAN_MODE_FULL`, then the entire database object is scanned for comparison. If `NULL` and the `scan_mode` parameter is set to `CMP_SCAN_MODE_CUSTOM`, then the portion of the database object scanned for comparison is specified when the `COMPARE` function is run. If non-`NULL` and the `scan_mode` parameter is set to either `CMP_SCAN_MODE_FULL` or `CMP_SCAN_MODE_CUSTOM`, then the `scan_percent` parameter is ignored. **Note:** When the `scan_percent` parameter is non-`NULL`, and the lead index column for the comparison does not distribute the rows in the database object evenly, the portion of the database object that is compared might be smaller or larger than the specified `scan_percent` value. See DBMS_COMPARISON Operational Notes for more information about the `DBMS_COMPARISON` package index requirements for the lead index column.
`null_value`	The value to substitute for each `NULL` in the database objects being compared. Specify a value or use the `CMP_NULL_VALUE_DEF` constant. If a column being compared can contain `NULL`s, then the value specified for this parameter must be different than any non-`NULL` value in the column. Otherwise, if the value specified for this parameter can appear in the column, some row differences might not be found. See "Constants" for information about this constant.

Table 52-6 (Cont.) CREATE_COMPARISON Procedure Parameters

Parameter	Description
`local_converge_tag`	The Oracle Replication tag to set in the session on the local database before performing any changes to converge the data in the database objects being compared. If the `local_converge_tag` parameter is non-`NULL` in the `CONVERGE` procedure when comparison results for this comparison are converged, then the setting in the `CONVERGE` procedure takes precedence. See CONVERGE Procedure for more information.
`remote_converge_tag`	The Oracle Replication tag to set in the session on the remote database before performing any changes to converge the data in the database objects being compared. If the `remote_converge_tag` parameter is non-`NULL` in the `CONVERGE` procedure when comparison results for this comparison are converged, then the setting in the `CONVERGE` procedure takes precedence. See CONVERGE Procedure for more information.
`max_num_buckets`	Specify the maximum number of buckets to use. Specify a value or use the `CMP_MAX_NUM_BUCKETS` constant. See "Constants" for information about this constant. See "Overview" for information about buckets. **Note:** If an index column for a comparison is a `VARCHAR2` or `CHAR` column, then the number of buckets might exceed the value specified for the `max_num_buckets` parameter.
`min_rows_in_bucket`	Specify the minimum number of rows in each bucket. Specify a value or use the `CMP_MIN_ROWS_IN_BUCKET` constant. See "Constants" for information about this constant. See "Overview" for information about buckets.

Usage Notes

This section contains usage notes for the `CREATE_COMPARISON` procedure.

Usage Notes for the index_schema_name and index_name Parameters

When you specify an index for a comparison with the `index_schema_name` and `index_name` parameters, the specified index determines the comparison's index columns and their ordering. The order of the columns in the index determines the index column ordering for the comparison. Therefore, the column in column position 1 in the index is the lead column for the comparison.

The index columns and their ordering affect the details of each SQL statement generated and executed for a comparison. For each SQL statement, the optimizer decides whether to use indexes. If the optimizer decides to use indexes, then the optimizer decides which particular indexes to use. An index specified in `column_list` parameter might or might not be used.

The columns in the specified index must meet the requirements described in DBMS_COMPARISON Operational Notes. If the index columns do not meet these requirements, then an error is raised.

Note:

If you do not specify an index when you create a comparison, then the CREATE_COMPARISON procedure selects either the primary key, if it exists, or an existing unique index. The procedure never selects a non-unique index. However, if you specify an index, then the CREATE_COMPARISON procedure does not check its uniqueness. Therefore, if you specify a non-unique index, and duplicate index keys exist, then the results might be incorrect when the CONVERGE procedure synchronizes data.

Usage Notes for the column_list Parameter

When the column_list parameter is set to a value other than '*', the following columns are required in the column_list parameter:

- Any columns that are required to meet the index column requirements for the DBMS_COMPARISON package. If the index_name parameter is non-NULL, then the columns in the specified index must be in the column list. If the index_name parameter is NULL, then see DBMS_COMPARISON Operational Notes for information about the DBMS_COMPARISON index requirements.
- If you plan to use the CONVERGE procedure to make changes to a database object based on the comparison, then any columns in this database object that have a NOT NULL constraint but no default value must be included in the column list. If these columns are not included, then the CONVERGE procedure returns an error. See CONVERGE Procedure.

DROP_COMPARISON Procedure

This procedure drops a comparison.

Syntax

```
DBMS_COMPARISON.DROP_COMPARISON(
   comparison_name   IN   VARCHAR2);
```

Parameters

Table 52-7   DROP_COMPARISON Procedure Parameters

Parameter	Description
comparison_name	The name of the comparison.

PURGE_COMPARISON Procedure

This procedure purges the comparison results, or a subset of the comparison results, for a comparison.

> **Note:**
>
> At least one of the following parameters must be set to `NULL`: `scan_id` or `purge_time`. If both the `scan_id` and `purge_time` parameters are `NULL`, then this procedure purges all comparison results for the comparison.

Syntax

```
DBMS_COMPARISON.PURGE_COMPARISON(
   comparison_name   IN  VARCHAR2,
   scan_id           IN  NUMBER      DEFAULT NULL,
   purge_time        IN  TIMESTAMP   DEFAULT NULL);
```

Parameters

Table 52-8 PURGE_COMPARISON Procedure Parameters

Parameter	Description
`comparison_name`	The name of the comparison.
`scan_id`	The scan ID of the scan for which results are purged. The scan ID must identify a root scan. If the scan ID does not identify a root scan, then an error is raised. When a root scan ID is specified, it is purged, and all direct and indirect child scans of the specified root scan are purged. If `NULL`, then no scan ID is considered when purging comparison results for the comparison. See "Overview" for information about scans.
`purge_time`	The date before which results are purged. If `NULL`, then no date is considered when purging comparison results for the comparison.

RECHECK Function

This function rechecks the differences in a specified scan for a comparison.

This function performs one of the following actions:

- If the specified scan completed successfully the last time it ran, then this function checks the previously identified differences in the scan.
- If the specified scan completed partially, then this function continues to check the database object from the point where the previous scan ended.

Note:

This function does not compare the shared database object for differences that were not recorded in the specified comparison scan. To check for those differences, run the `COMPARE` function.

See Also:

COMPARE Function

Syntax

```
DBMS_COMPARISON.RECHECK(
   comparison_name  IN  VARCHAR2,
   scan_id          IN  NUMBER,
   perform_row_dif  IN  BOOLEAN   DEFAULT FALSE)
RETURN BOOLEAN;
```

Parameters

Table 52-9 RECHECK Function Parameters

Parameter	Description
`comparison_name`	The name of the comparison.
`scan_id`	The scan ID of the scan to recheck. See "Overview" for more information about specifying a scan ID in this parameter.
`perform_row_dif`	If `TRUE`, then compares each row individually in the database objects being compared after reaching the smallest possible bucket for the comparison. If `FALSE`, then compares buckets for differences but does not compare each row individually when differences are found in the smallest possible bucket. See "Overview" for information about buckets.

Return Values

This function returns `TRUE` when no differences are found in the database objects being compared. This function returns `FALSE` when differences are found in the database objects being compared.