314 Logical Change Record TYPEs

This chapter describes the logical change record (LCR) types.

This chapter contains these topics:

Logical Change Record TYPEs Overview

In Replication (Oracle GoldenGate and XStream), logical change records (LCRs) are message payloads that contain information about changes to a database. These changes can include changes to the data, which are data manipulation language (DML) changes, and changes to database objects, which are data definition language (DDL) changes.

When you use Replication (Oracle GoldenGate and XStream), the capture process captures changes in the form of LCRs and enqueues them into a queue. Finally, the apply process can apply LCRs at a destination database. You also have the option of creating, enqueuing, and dequeuing LCRs manually.

Logical Change Record Types Security Model

`PUBLIC` is granted `EXECUTE` privilege on the types described in this chapter.

Summary of Logical Change Record Types

This table lists the Logical Change Record TYPEs and briefly describes them.

Table 314-1 Logical Change Record (LCR) Types

Type	Description
LCR$_DDL_RECORD Type	Represents a data definition language (DDL) change to a database object
LCR$_ROW_RECORD Type	Represents a data manipulation language (DML) change to a database object
LCR$_ROW_LIST Type	Identifies a list of column values for a row in a table
LCR$_ROW_UNIT Type	Identifies the value for a column in a row

These logical change record (LCR) types can be used with the following Oracle-supplied PL/SQL packages:

- `DBMS_APPLY_ADM`
- `DBMS_AQ`

- `DBMS_AQADM`
- `DBMS_CAPTURE_ADM`
- `DBMS_PROPAGATION_ADM`
- `DBMS_RULE`
- `DBMS_RULE_ADM`
- `DBMS_STREAMS`
- `DBMS_STREAMS_ADM`
- `DBMS_TRANSFORM`

LCR$_DDL_RECORD Type

This type represents a data definition language (DDL) change to a database object.

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.

If you create or modify a DDL logical change record (DDL LCR), then make sure the `ddl_text` is consistent with the `base_table_name`, `base_table_owner`, `object_type`, `object_owner`, `object_name`, and `command_type` attributes.

This topic contains information about the constructor for row LCRs and information about the member subprograms for this type.

Note:

- When passing a name as a parameter to an LCR constructor, you can enclose the name in double quotes to handle names that use mixed case or lower case for database objects. For example, if a name contains any lower case characters, then you must enclose it in double quotes.
- The application does not need to specify a transaction identifier or SCN when it creates an LCR because the apply process generates these values and stores them in memory. If a transaction identifier or SCN is specified in the LCR, then the apply process ignores it and assigns a new value.

LCR$_DDL_RECORD Constructor

Creates a `SYS.LCR$_DDL_RECORD` object with the specified information.

```
STATIC FUNCTION CONSTRUCT(
   source_database_name  IN  VARCHAR2,
   command_type          IN  VARCHAR2,
   object_owner          IN  VARCHAR2,
   object_name           IN  VARCHAR2,
   object_type           IN  VARCHAR2,
   ddl_text              IN  CLOB,
   logon_user            IN  VARCHAR2,
   current_schema        IN  VARCHAR2,
   base_table_owner      IN  VARCHAR2,
   base_table_name       IN  VARCHAR2,
   tag                   IN  RAW       DEFAULT NULL,
   transaction_id        IN  VARCHAR2  DEFAULT NULL,
   scn                   IN  NUMBER    DEFAULT NULL,
   position              IN  RAW       DEFAULT NULL,
   edition_name          IN  VARCHAR2  DEFAULT NULL,
   root_name             IN  VARCHAR2  DEFAULT NULL)
RETURN SYS.LCR$_DDL_RECORD;
```

LCR$_DDL_RECORD Constructor Function Parameters

Table 314-2 Constructor Function Parameters for LCR$_DDL_RECORD

Parameter	Description
`source_database_name`	The database where the DDL statement occurred If the LCRs originated in a multitenant container database (CDB), then this field specifies the global name of the container where the DDL change occurred. If you do not include the domain name, then the function appends the local domain to the database name automatically. For example, if you specify `DBS1` and the local domain is `EXAMPLE.COM`, then the function specifies `DBS1.EXAMPLE.COM` automatically. Set this parameter to a non-`NULL` value.

`command_type`	The type of command executed in the DDL statement
	Set this parameter to a non-`NULL` value.
	See Also: The "SQL Command Codes" table in the *Oracle Call Interface Programmer's Guide* for a complete list of command types
	The following command types *are not supported* in DDL LCRs:
	`ALTER MATERIALIZED VIEW`
	`ALTER MATERIALIZED VIEW LOG`
	`ALTER SUMMARY`
	`CREATE SCHEMA`
	`CREATE MATERIALIZED VIEW`
	`CREATE MATERIALIZED VIEW LOG`
	`CREATE SUMMARY`
	`DROP MATERIALIZED VIEW`
	`DROP MATERIALIZED VIEW LOG`
	`DROP SUMMARY`
	`RENAME`
	The snapshot equivalents of the materialized view command types are also not supported.

`object_owner`	The user who owns the object on which the DDL statement was executed
`object_name`	The database object on which the DDL statement was executed

Table 314-2 (Cont.) Constructor Function Parameters for LCR$_DDL_RECORD

Parameter	Description

`object_type`	The type of object on which the DDL statement was executed
	The following are valid object types:
	`CLUSTER`
	`FUNCTION`
	`INDEX`
	`LINK`
	`OUTLINE`
	`PACKAGE`
	`PACKAGE BODY`
	`PROCEDURE`
	`SEQUENCE`
	`SYNONYM`
	`TABLE`
	`TRIGGER`
	`TYPE`
	`USER`
	`VIEW`
	`LINK` represents a database link.
	`NULL` is also a valid object type. Specify `NULL` for all object types not listed. The `GET_OBJECT_TYPE` member procedure returns `NULL` for object types not listed.

`ddl_text`	The text of the DDL statement Set this parameter to a non-`NULL` value.
`logon_user`	The user whose session executed the DDL statement
`current_schema`	The schema that is used if no schema is specified explicitly for the modified database objects in `ddl_text` If a schema is specified in `ddl_text` that differs from the one specified for `current_schema`, then the function uses the schema specified in `ddl_text`. Set this parameter to a non-`NULL` value.
`base_table_owner`	If the DDL statement is a table-related DDL (such as `CREATE TABLE` and `ALTER TABLE`), or if the DDL statement involves a table (such as creating a trigger on a table), then `base_table_owner` specifies the owner of the table involved. Otherwise, `base_table_owner` is `NULL`.
`base_table_name`	If the DDL statement is a table-related DDL (such as `CREATE TABLE` and `ALTER TABLE`), or if the DDL statement involves a table (such as creating a trigger on a table), then `base_table_name` specifies the name of the table involved. Otherwise, `base_table_name` is `NULL`.
`tag`	A binary tag that enables tracking of the LCR For example, this tag can be used to determine the original source database of the DDL statement if apply forwarding is used.
`transaction_id`	The identifier of the transaction
`scn`	The SCN at the time when the change record for a captured LCR was written to the redo log The SCN value is meaningless for a user-created LCR.

Table 314-2 (Cont.) Constructor Function Parameters for LCR$_DDL_RECORD

Parameter	Description
`position`	The position of the LCR LCR position is commonly used in XStream configurations. Using XStream requires purchasing a license for the Oracle GoldenGate product. **See Also:** *Oracle Database XStream Guide*
`edition_name`	The name of the edition in which the DDL statement was executed
`root_name`	If the LCRs is associated with a CDB, then this field specifies the global name of the root in the CDB. If the LCR is associated with a non-CDB, then this field is `NULL`.

Summary of LCR$_DDL_RECORD Subprograms

Table 314-3 LCR$_DDL_RECORD Type Subprograms

Subprogram	Description
EXECUTE Member Procedure	Executes the LCR under the security domain of the current user
GET_BASE_TABLE_NAME Member Function	Gets the base (dependent) table name
GET_BASE_TABLE_OWNER Member Function	Gets the base (dependent) table owner
GET_CURRENT_SCHEMA Member Function	Gets the default schema (user) name
GET_DDL_TEXT Member Procedure	Gets the DDL text in a `CLOB`
GET_EDITION_NAME Member Function	Gets the name of the edition in which the DDL statement was executed
GET_LOGON_USER Member Function	Gets the logon user name
GET_OBJECT_TYPE Member Function	Gets the type of the object involved for the DDL
SET_BASE_TABLE_NAME Member Procedure	Sets the base (dependent) table name
SET_BASE_TABLE_OWNER Member Procedure	Sets the base (dependent) table owner
SET_CURRENT_SCHEMA Member Procedure	Sets the default schema (user) name
SET_DDL_TEXT Member Procedure	Sets the DDL text
SET_EDITION_NAME Member Procedure	Sets the name of the edition in which the DDL statement was executed
SET_LOGON_USER Member Procedure	Sets the logon user name
SET_OBJECT_TYPE Member Procedure	Sets the object type
Common Subprograms	See "Common Subprograms for LCR$_DDL_RECORD and LCR$_ROW_RECORD" for a list of subprograms common to the `SYS.LCR$_ROW_RECORD` and `SYS.LCR$_DDL_RECORD` types

EXECUTE Member Procedure

Executes the DDL LCR under the security domain of the current user. Apply handlers are not run when the LCR is applied using this procedure.

Syntax

```
MEMBER PROCEDURE EXECUTE;
```

GET_BASE_TABLE_NAME Member Function

Gets the base (dependent) table name.

Syntax

```
MEMBER FUNCTION GET_BASE_TABLE_NAME()
RETURN VARCHAR2;
```

GET_BASE_TABLE_OWNER Member Function

Gets the base (dependent) table owner.

Syntax

```
MEMBER FUNCTION GET_BASE_TABLE_OWNER()
RETURN VARCHAR2;
```

GET_CURRENT_SCHEMA Member Function

Gets the current schema name.

Syntax

```
MEMBER FUNCTION GET_CURRENT_SCHEMA()
RETURN VARCHAR2;
```

GET_DDL_TEXT Member Procedure

Gets the DDL text in a `CLOB`.

For example, the following PL/SQL code uses this procedure to get the DDL text in a DDL LCR:

```
CREATE OR REPLACE PROCEDURE ddl_in_lcr (ddl_lcr in SYS.LCR$_DDL_RECORD)
IS
  ddl_text   CLOB;
BEGIN
  DBMS_OUTPUT.PUT_LINE( '  -----------------------------------------' );
  DBMS_OUTPUT.PUT_LINE( '  Displaying DDL text in a DDL LCR: ' );
  DBMS_OUTPUT.PUT_LINE( '  -----------------------------------------' );
  DBMS_LOB.CREATETEMPORARY(ddl_text, true);
  ddl_lcr.GET_DDL_TEXT(ddl_text);
  DBMS_OUTPUT.PUT_LINE('DDL text:' || ddl_text);
  DBMS_LOB.FREETEMPORARY(ddl_text);
END;
/
```

Note:

GET_DDL_TEXT is a member procedure and not a member function to make it easier for you to manage the space used by the CLOB. Notice that the previous example creates temporary space for the CLOB and then frees the temporary space when it is no longer needed.

Syntax

```
MEMBER FUNCTION GET_DDL_TEXT(
  ddl_text   IN/OUT   CLOB);
```

Parameter

Table 314-4   GET_DDL_TEXT Procedure Parameter

Parameter	Description
ddl_text	The DDL text in the DDL LCR

GET_EDITION_NAME Member Function

Gets the name of the edition in which the DDL statement was executed.

See Also:

Oracle Database Development Guide

Syntax

```
MEMBER FUNCTION GET_EDITION_NAME()
RETURN VARCHAR2;
```

GET_LOGON_USER Member Function

Gets the logon user name.

Syntax

```
MEMBER FUNCTION GET_LOGON_USER()
RETURN VARCHAR2;
```

GET_OBJECT_TYPE Member Function

Gets the type of the object involved for the DDL.

Syntax

```
MEMBER FUNCTION GET_OBJECT_TYPE()
RETURN VARCHAR2;
```

SET_BASE_TABLE_NAME Member Procedure

Sets the base (dependent) table name.

Syntax

```
MEMBER PROCEDURE SET_BASE_TABLE_NAME(
   base_table_name  IN  VARCHAR2);
```

Parameter

Table 314-5   SET_BASE_TABLE_NAME Procedure Parameter

Parameter	Description
`base_table_name`	The name of the base table

SET_BASE_TABLE_OWNER Member Procedure

Sets the base (dependent) table owner.

Syntax

```
MEMBER PROCEDURE SET_BASE_TABLE_OWNER(
   base_table_owner  IN  VARCHAR2);
```

Parameter

Table 314-6   SET_BASE_TABLE_OWNER Procedure Parameter

Parameter	Description
`base_table_owner`	The name of the base table owner

SET_CURRENT_SCHEMA Member Procedure

Sets the default schema (user) name.

Syntax

```
MEMBER PROCEDURE SET_CURRENT_SCHEMA(
   current_schema  IN  VARCHAR2);
```

Parameter

Table 314-7   SET_CURRENT_SCHEMA Procedure Parameter

Parameter	Description
`current_schema`	The name of the schema to set as the current schema Set this parameter to a non-`NULL` value.

SET_DDL_TEXT Member Procedure

Sets the DDL text.

Syntax

```
MEMBER PROCEDURE SET_DDL_TEXT(
   ddl_text  IN  CLOB);
```

Parameter

Table 314-8 SET_DDL_TEXT Procedure Parameter

Parameter	Description
`ddl_text`	The DDL text Set this parameter to a non-`NULL` value.

SET_EDITION_NAME Member Procedure

Sets the name of the edition in which the DDL statement was executed.

> **See Also:**
>
> *Oracle Database Development Guide*

Syntax

```
MEMBER PROCEDURE SET_EDITION_NAME(
   edition_name  IN  VARCHAR2);
```

Parameter

Table 314-9 SET_EDITION_NAME Procedure Parameter

Parameter	Description
`edition_name`	Name of the edition

SET_LOGON_USER Member Procedure

Sets the logon user name.

Syntax

```
MEMBER PROCEDURE SET_LOGON_USER(
   logon_user  IN  VARCHAR2);
```

Parameter

Table 314-10 SET_LOGON_USER Procedure Parameter

Parameter	Description
`logon_user`	The name of the schema to set as the logon user

SET_OBJECT_TYPE Member Procedure

Sets the object type.

Syntax

```
MEMBER PROCEDURE SET_OBJECT_TYPE(
   object_type  IN  VARCHAR2);
```

Parameter

Table 314-11 SET_OBJECT_TYPE Procedure Parameter

Parameter	Description

`object_type`	The object type
	The following are valid object types:
	`CLUSTER`
	`FUNCTION`
	`INDEX`
	`LINK`
	`OUTLINE`
	`PACKAGE`
	`PACKAGE BODY`
	`PROCEDURE`
	`SEQUENCE`
	`SYNONYM`
	`TABLE`
	`TRIGGER`
	`TYPE`
	`USER`
	`VIEW`
	`LINK` represents a database link.
	`NULL` is also a valid object type. Specify `NULL` for all object types not listed. The `GET_OBJECT_TYPE` member procedure returns `NULL` for object types not listed.

LCR$_ROW_RECORD Type

This type represents a data manipulation language (DML) change to a row in a table. This type uses the `LCR$_ROW_LIST` type.

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.

If you create or modify a row logical change record (row LCR), then make sure the `command_type` attribute is consistent with the presence or absence of old column values and the presence or absence of new column values.

This topic contains information about the constructor for DDL LCRs and information about the member subprograms for this type.

Note:

- When passing a name as a parameter to an LCR constructor, you can enclose the name in double quotes to handle names that use mixed case or lower case for database objects. For example, if a name contains any lower case characters, then you must enclose it in double quotes.
- The application does not need to specify a transaction identifier or SCN when it creates an LCR because the apply process generates these values and stores them in memory. If a transaction identifier or SCN is specified in the LCR, then the apply process ignores it and assigns a new value.

See Also:

LCR$_ROW_LIST Type

LCR$_ROW_RECORD Constructor

Creates a `SYS.LCR$_ROW_RECORD` object with the specified information.

```
STATIC FUNCTION CONSTRUCT(
   source_database_name  IN  VARCHAR2,
   command_type          IN  VARCHAR2,
   object_owner          IN  VARCHAR2,
   object_name           IN  VARCHAR2,
   tag                   IN  RAW                DEFAULT NULL,
   transaction_id        IN  VARCHAR2           DEFAULT NULL,
   scn                   IN  NUMBER             DEFAULT NULL,
   old_values            IN  SYS.LCR$_ROW_LIST  DEFAULT NULL,
   new_values            IN  SYS.LCR$_ROW_LIST  DEFAULT NULL,
   position              IN  RAW                DEFAULT NULL,

   statement             IN  VARCHAR2           DEFAULT NULL,

   bind_variables        IN  SYS.LCR$_ROW_LIST  DEFAULT NULL,

   bind_by_position      IN  VARCHAR2           DEFAULT 'N',

   root_name             IN  VARCHAR2           DEFAULT NULL)
RETURN SYS.LCR$_ROW_RECORD;
```

LCR$_ROW_RECORD Constructor Function Parameters

Table 314-12 Constructor Function Parameters for LCR$_ROW_RECORD

Parameter	Description
`source_database_name`	The database where the row change occurred If the LCRs originated in a CDB, then this field specifies the global name of the container where the row change occurred. If you do not include the domain name, then the function appends the local domain to the database name automatically. For example, if you specify `DBS1` and the local domain is `EXAMPLE.COM`, then the function specifies `DBS1.EXAMPLE.COM` automatically. Set this parameter to a non-`NULL` value.

`command_type`	The type of command executed in the DML statement
	Set this parameter to a non-`NULL` value.
	Valid values are the following:
	`INSERT`
	`UPDATE`
	`DELETE`
	`LOB ERASE`
	`LOB WRITE`
	`LOB TRIM`
	If `INSERT`, then ensure that the LCR has a `new_values` collection that is not empty and an empty or `NULL` `old_values` collection.
	If `UPDATE`, then ensure that the LCR has a `new_values` collection that is not empty and an `old_values` collection that is not empty.
	If `DELETE`, then ensure that the LCR has a `NULL` or empty `new_values` collection and an `old_values` collection that is not empty.
	If `LOB ERASE`, `LOB WRITE`, or `LOB TRIM`, then ensure that the LCR has a `new_values` collection that is not empty and an empty or `NULL` `old_values` collection.

`object_owner`	The user who owns the table on which the row change occurred Set this parameter to a non-`NULL` value.
`object_name`	The table on which the DML statement was executed Set this parameter to a non-`NULL` value.
`tag`	A binary tag that enables tracking of the LCR For example, this tag can be used to determine the original source database of the DML change when apply forwarding is used.
`transaction_id`	The identifier of the transaction
`scn`	The SCN at the time when the change record was written to the redo log The SCN value is meaningless for a user-created LCR.
`old_values`	The column values for the row before the DML change If the DML statement is an `UPDATE` or a `DELETE` statement, then this parameter contains the values of columns in the row before the DML statement. If the DML statement is an `INSERT` statement, then there are no old values.

Table 314-12 (Cont.) Constructor Function Parameters for LCR$_ROW_RECORD

Parameter	Description
`new_values`	The column values for the row after the DML change If the DML statement is an `UPDATE` or an `INSERT` statement, then this parameter contains the values of columns in the row after the DML statement. If the DML statement is a `DELETE` statement, then there are no new values. If the LCR reflects a LOB operation, then this parameter contains the supplementally logged columns and any relevant LOB information.
`position`	The position of the LCR LCR position is commonly used in XStream configurations. Using XStream requires purchasing a license for the Oracle GoldenGate product. **See Also:** *Oracle Database XStream Guide*
`statement`	This parameter is reserved for internal use only.
`bind_variables`	This parameter is reserved for internal use only.
`bind_by_position`	This parameter is reserved for internal use only.
`root_name`	If the LCRs is associated with a CDB, then this field specifies the global name of the root in the CDB. If the LCR is associated with a non-CDB, then this field is `NULL`.

Summary of LCR$_ROW_RECORD Subprograms

Table 314-13 LCR$_ROW_RECORD Type Subprograms

Subprogram	Description
ADD_COLUMN Member Procedure	Adds the value as old or new, depending on the value type specified, for the column
CONVERT_LONG_TO_LOB_CHUNK Member Procedure	Converts `LONG` data in a row LCR into fixed width `CLOB`, or converts `LONG RAW` data in a row LCR into a `BLOB`
DELETE_COLUMN Member Procedure	Deletes the old value, the new value, or both, for the specified column, depending on the value type specified
EXECUTE Member Procedure	Executes the LCR under the security domain of the current user
GET_JSON_INFORMATION Member Function	Gets JSON information.
GET_LOB_INFORMATION Member Function	Gets the LOB information for the column
GET_LOB_OFFSET Member Function	Gets the LOB offset for the specified column
GET_LOB_OPERATION_SIZE Member Function	Gets the operation size for the `LOB` column
GET_LONG_INFORMATION Member Function	Gets the `LONG` information for the column
GET_ROW_TEXT Member Procedure	Gets the SQL statement for the change that is encapsulated in the LCR
GET_VALUE Member Function	Gets the old or new value for the specified column, depending on the value type specified

Table 314-13 (Cont.) LCR$_ROW_RECORD Type Subprograms

Subprogram	Description
GET_VALUES Member Function	Gets a list of old or new values, depending on the value type specified
GET_WHERE_CLAUSE Member Procedure	Gets a `WHERE` clause for the change that is encapsulated in the row LCR
GET_XML_INFORMATION Member Function	Gets the XML information for the specified column
IS_STATEMENT_LCR Member Function	Reserved for internal use only
KEEP_COLUMNS Member Procedure	Keeps a list of columns a row LCR
RENAME_COLUMN Member Procedure	Renames a column in an LCR
SET_JSON_INFORMATION Member Procedure	Sets the JSON information for a column.
SET_LOB_INFORMATION Member Procedure	Sets LOB information for the column
SET_LOB_OFFSET Member Procedure	Sets the LOB offset for the specified column
SET_LOB_OPERATION_SIZE Member Procedure	Sets the operation size for the LOB column
SET_ROW_TEXT Member Procedure	Reserved for internal use only
SET_VALUE Member Procedure	Overwrites the value of the specified column
SET_VALUES Member Procedure	Replaces the existing old or new values for the LCR, depending on the value type specified
SET_XML_INFORMATION Member Procedure	Sets the XML information for the column
Common Subprograms	See Common Subprograms for LCR$_DDL_RECORD and LCR$_ROW_RECORD for a list of subprograms common to the `SYS.LCR$_ROW_RECORD` and `SYS.LCR$_DDL_RECORD` types

ADD_COLUMN Member Procedure

Adds the value as old or new, depending on the value type specified, for the column. An error is raised if a value of the same type already exists for the column.

Note:

To set a column value that already exists, run `SET_VALUE`.

See Also:

SET_VALUE Member Procedure

Considerations for LOB Columns

When processing a row LCR with LOB columns with a procedure DML handler or error handler and the handler is using LOB assembly (the `assemble_lobs` parameter is set to `TRUE` for the handler), you use this member procedure in the handler procedure to add a LOB column to a row LCR. If assemble_lobs is set to FALSE for the handler, then you cannot use this member procedure to add a LOB column to a row LCR.

To use a DML or error handler to add a LOB column, specify the LOB locator for the column_value parameter in the member procedure. The ADD_COLUMN member procedure verifies that an ANYDATA encapsulated LOB locator is processed with a DML or error handler that is using LOB assembly. An error is raised under the following conditions:

- The handler attempts to enqueue a row LCR with an ANYDATA encapsulated LOB locator.
- An attempt is made to add an LOB column that is set incorrectly.

If an error is raised because of one of these conditions, then the transaction that includes the row LCR is moved to the error queue, and the LOB is represented by the original (nonassembled) row LCRs.

Note:

- Database compatibility must be 10.2.0 or higher to use LOB assembly.
- When you are processing a row LCR with a rule-based transformation, you cannot use this member procedure to add a LOB column.
- When you are processing a row LCR with a rule-based transformation, procedure DML handler, or error handler, you cannot use this member procedure to add a LONG or LONG RAW column.

Syntax

```
MEMBER PROCEDURE ADD_COLUMN(
   value_type     IN  VARCHAR2,
   column_name    IN  VARCHAR2,
   column_value   IN  ANYDATA);
```

Parameters

Table 314-14 ADD_COLUMN Procedure Parameters

Parameter	Description
value_type	The type of value to add for the column Specify old to add the old value of the column. Specify new to add the new value of the column.
column_name	The column name This name is not validated. An error can be raised during application of the LCRs if an invalid name is specified.
column_value	The value of the column If NULL, then this procedure raises an error. If the member procedure is used in a procedure DML handler or error handler that uses LOB assembly, then a LOB locator can be specified. A NULL column value can be specified by encapsulating the NULL value in an ANYDATA wrapper.

CONVERT_LONG_TO_LOB_CHUNK Member Procedure

Converts `LONG` data in a row LCR into a `CLOB`, or converts `LONG RAW` data in a row LCR into a `BLOB`.

This procedure can change the operation code from `LONG WRITE` to `LOB WRITE` for the row LCR.

This member procedure can be used in rule-based transformations.

The following restrictions apply to this member procedure:

- This member procedure cannot be used in apply handlers.
- `LONG` data can be sent as a part of a row LCR with one of the following operation codes: `INSERT`, `UPDATE`, or `LONG_WRITE`. Because `LONG` data can be sent in multiple pieces, make sure that this method is invoked on either none or all `LONG` pieces.
- LOB to `LONG` conversion is not supported.
- A row LCR on which this procedure is executed must have been created by a capture process. That is, this procedure does not support persistent row LCRs.

Syntax

```
MEMBER PROCEDURE CONVERT_LONG_TO_LOB_CHUNK;
```

DELETE_COLUMN Member Procedure

Deletes the old value, the new value, or both, for the specified column, depending on the value type specified.

Syntax

```
MEMBER PROCEDURE DELETE_COLUMN(
   column_name   IN  VARCHAR2,
   value_type    IN  VARCHAR2  DEFAULT '*');
```

Parameters

Table 314-15 DELETE_COLUMN Procedure Parameters

Parameter	Description
`column_name`	The column name An error is raised if the column does not exist in the LCR.
`value_type`	The type of value to delete for the column Specify `old` to delete the old value of the column. Specify `new` to delete the new value of the column. If `*` is specified, then the procedure deletes both the old and new values.

EXECUTE Member Procedure

Executes the row LCR under the security domain of the current user. Any apply handlers that would be run for an LCR are not run when the LCR is applied using this procedure.

This member procedure can be run on a row LCR under any of the following conditions:

- The LCR is being processed by an apply handler.

- The LCR has been constructed using the `LCR$_ROW_RECORD` constructor function but has not been enqueued.
- The LCR is in the error queue.

Note:

Do not run this member procedure in a custom rule-based transformation on a row LCR. Doing so could execute the row LCR outside of its transactional context.

Considerations for LOB Columns

When processing a row LCR with LOB columns with a procedure DML handler or error handler, and the handler is using LOB assembly (the `assemble_lobs` parameter is set to `TRUE` for the handler), this member procedure executes the assembled row LCR. An assembled row LCR represents a LOB value with a LOB locator or `NULL`.

If `assemble_lobs` is set to `FALSE` for the handler, then this member procedure executes the nonassembled row LCRs. Nonassembled row LCRs represent LOB values with `VARCHAR2` and `RAW` data types. These nonassembled row LCRs might have been modified by the handler.

An error is raised under the following conditions:

- A DML or error handler configured with `assemble_lobs` set to `FALSE` attempts to execute a row LCR that contains a LOB locator.
- A DML or error handler configured with `assemble_lobs` set to `TRUE` attempts to execute a row LCR that contains one or more LOB values represented with `VARCHAR2` or `RAW` data types.

If an error is raised because of one of these conditions, then the transaction that includes the row LCR is moved to the error queue, and the LOB is represented by the original (nonassembled) row LCRs.

Syntax

```
MEMBER PROCEDURE EXECUTE(
   conflict_resolution  IN  BOOLEAN);
```

Parameters

Table 314-16   EXECUTE Procedure Parameters

Parameter	Description
`conflict_resolution`	If `TRUE`, then any conflict resolution defined for the table using the `SET_UPDATE_CONFLICT_HANDLER` procedure in the `DBMS_APPLY_ADM` package is used to resolve conflicts resulting from the execution of the LCR. If `FALSE`, then conflict resolution is not used. An error is raised if this parameter is not specified or is set to `NULL`.

GET_JSON_INFORMATION Member Function

Gets JSON information.

The return value can be one of the following:

```
DBMS_LCR.NOT_OSON CONSTANT NUMBER := 1;
DBMS_LCR.OSON_DOC CONSTANT NUMBER := 2;
```

Note:

OSON is Oracle binary format for JSON column.

They represent either a column not containing OSON data or full OSON document.

Syntax

```
MEMBER FUNCTION GET_JSON_INFORMATION(
   column_name   IN   VARCHAR2)
RETURN NUMBER;
```

Parameters

Table 314-17   GET_JSON_INFORMATION Member Function Parameter

Parameter	Description
column_name	Name of column to obtain JSON information.

GET_LOB_INFORMATION Member Function

Gets the LOB information for the column.

The return value can be one of the following:

```
DBMS_LCR.NOT_A_LOB        CONSTANT NUMBER := 1;
DBMS_LCR.NULL_LOB         CONSTANT NUMBER := 2;
DBMS_LCR.INLINE_LOB       CONSTANT NUMBER := 3;
DBMS_LCR.EMPTY_LOB        CONSTANT NUMBER := 4;
DBMS_LCR.LOB_CHUNK        CONSTANT NUMBER := 5;
DBMS_LCR.LAST_LOB_CHUNK   CONSTANT NUMBER := 6;
```

Returns NULL if the specified column does not exist.

If the command type of the row LCR is UPDATE, then specifying 'Y' for the use_old parameter is a convenient way to get the value of the columns.

Syntax

```
MEMBER FUNCTION GET_LOB_INFORMATION(
  value_type    IN  VARCHAR2,
  column_name   IN  VARCHAR2,
  use_old       IN  VARCHAR2  DEFAULT 'Y')
RETURN NUMBER;
```

Parameters

Table 314-18 GET_LOB_INFORMATION Function Parameters

Parameter	Description
`value_type`	The type of value to return for the column, either `old` or `new`
`column_name`	The name of the column
`use_old`	If `Y` and `value_type` is `new`, and no new value exists, then the function returns the corresponding old value. If `N` and `value_type` is `new`, then the function does not return the old value if no new value exists. If `value_type` is `old` or if the `command_type` of the row LCR is not `UPDATE`, then the function ignores the value of the `use_old` parameter. `NULL` is not a valid specification for the `use_old` parameter.

GET_LOB_OFFSET Member Function

Gets the LOB offset for the specified column in the number of characters for `CLOB` columns and the number of bytes for `BLOB` columns. Returns a non-`NULL` value only if all of the following conditions are met:

- The value exists for the column
- The column value is an out-of-line LOB. That is, the information is `DBMS_LCR.LAST_LOB_CHUNK` or `DBMS_LCR.LOB_CHUNK`
- The command type is `LOB ERASE` or `LOB WRITE`

Otherwise, returns `NULL`.

Syntax

```
GET_LOB_OFFSET(
   value_type   IN  VARCHAR2,
   column_name  IN  VARCHAR2)
RETURN NUMBER;
```

Parameters

Table 314-19 GET_LOB_OFFSET Function Parameters

Parameter	Description
`value_type`	The type of value to return for the column Currently, only `new` can be specified.
`column_name`	The name of the LOB column

GET_LOB_OPERATION_SIZE Member Function

Gets the operation size for the LOB column in the number of characters for `CLOB` columns and the number of bytes for `BLOB` columns. Returns a non-`NULL` value only if all of the following conditions are met:

- The value exists for the column
- The column value is an out-of-line LOB

- The command type is `LOB ERASE` or `LOB TRIM`
- The information is `DBMS_LCR.LAST_LOB_CHUNK`

Otherwise, returns `NULL`.

Syntax

```
MEMBER FUNCTION GET_LOB_OPERATION_SIZE(
  value_type    IN  VARCHAR2,
  column_name   IN  VARCHAR2)
RETURN NUMBER,
```

Parameters

Table 314-20 GET_LOB_OPERATION_SIZE Function Parameters

Parameter	Description
`value_type`	The type of value to return for the column Currently, only `new` can be specified.
`column_name`	The name of the LOB column

GET_LONG_INFORMATION Member Function

Gets the `LONG` information for the column.

The return value can be one of the following:

```
DBMS_LCR.NOT_A_LONG         CONSTANT NUMBER := 1;
DBMS_LCR.NULL_LONG          CONSTANT NUMBER := 2;
DBMS_LCR.INLINE_LONG        CONSTANT NUMBER := 3;
DBMS_LCR.LONG_CHUNK         CONSTANT NUMBER := 4;
DBMS_LCR.LAST_LONG_CHUNK    CONSTANT NUMBER := 5;
```

Returns `NULL` if the specified column does not exist.

If the command type of the row LCR is `UPDATE`, then specifying `'Y'` for the `use_old` parameter is a convenient way to get the value of the columns.

Syntax

```
MEMBER FUNCTION GET_LONG_INFORMATION(
  value_type    IN  VARCHAR2,
  column_name   IN  VARCHAR2,
  use_old       IN  VARCHAR2  DEFAULT 'Y')
RETURN NUMBER;
```

Parameters

Table 314-21 GET_LONG_INFORMATION Function Parameters

Parameter	Description
`value_type`	The type of value to return for the column, either `old` or `new`
`column_name`	The name of the column

Table 314-21 (Cont.) GET_LONG_INFORMATION Function Parameters

Parameter	Description
`use_old`	If `Y` and `value_type` is `new`, and no new value exists, then the function returns the corresponding old value. If `N` and `value_type` is `new`, then the function does not return the old value if no new value exists. If `value_type` is `old` or if the `command_type` of the row LCR is not `UPDATE`, then the function ignores the value of the `use_old` parameter. `NULL` is not a valid specification for the `use_old` parameter.

GET_ROW_TEXT Member Procedure

Gets the SQL statement for the change that is encapsulated in the row LCR. This method performs SQL generation in PL/SQL.

This method is overloaded. The different functionality of each form of syntax is presented along with the definitions.

Syntax

The following procedure returns the SQL statement in a `CLOB` datatype.

```
MEMBER PROCEDURE GET_ROW_TEXT(
   row_text  IN/OUT  CLOB);
```

The following procedure returns the SQL statement with bind variables in a `CLOB` datatype.

```
MEMBER PROCEDURE GET_ROW_TEXT(
   row_text         IN/OUT  CLOB,
   variable_list    IN/OUT  LCR$_ROW_LIST,
   bind_var_syntax  IN      VARCHAR2  DEFAULT ':');
```

See Also:

"LCR$_ROW_LIST Type"

Parameters

Table 314-22 GET_ROW_TEXT Procedure Parameters

Parameter	Description
`row_text`	The SQL statement for the change that is encapsulated in the LCR
`variable_list`	The values for the bind variables in the order of the bind variables
`bind_var_syntax`	The syntax for the bind variables One of the following values is valid: • Specify `:`, the default, for bind values to be in the form `:1`, `:2`, and so on. • Specify `?` for bind values to be in the form `?`.

GET_VALUE Member Function

Gets the old or new value for the specified column, depending on the value type specified.

If the command type of the row LCR is `UPDATE`, then specifying `'Y'` for the `use_old` parameter is a convenient way to get the value of a column.

Syntax

```
MEMBER FUNCTION GET_VALUE(
   value_type   IN  VARCHAR2,
   column_name  IN  VARCHAR2,
   use_old      IN  VARCHAR2  DEFAULT 'Y')
RETURN ANYDATA;
```

Parameters

Table 314-23 GET_VALUE Function Parameters

Parameter	Description
`value_type`	The type of value to return for the column Specify `old` to get the old value for the column. Specify `new` to get the new value for the column.
`column_name`	The column name If the column is present and has a `NULL` value, then the function returns an `ANYDATA` instance containing a `NULL` value. If the column value is absent, then the function returns a `NULL`.
`use_old`	If `Y` and `value_type` is `new`, and no new value exists, then the function returns the corresponding old value. If `N` and `value_type` is `new`, then the function returns `NULL` if no new value exists. If `value_type` is `old` or if the `command_type` of the row LCR is not `UPDATE`, then the function ignores the value of the `use_old` parameter. `NULL` is not a valid specification for the `use_old` parameter.

GET_VALUES Member Function

Gets a list of old or new values, depending on the value type specified.

If the command type of the row LCR is `UPDATE`, then specifying `'Y'` for the `use_old` parameter is a convenient way to get the values of all columns.

Syntax

```
MEMBER FUNCTION GET_VALUES(
   value_type  IN  VARCHAR2,
   use_old     IN  VARCHAR2  DEFAULT 'Y')
RETURN SYS.LCR$_ROW_LIST;
```

Parameters

Table 314-24 GET_VALUES Function Parameters

Parameter	Description
`value_type`	The type of values to return Specify `old` to return a list of old values. Specify `new` to return a list of new values.
`use_old`	If `Y` and `value_type` is `new`, then the function returns a list of all new values in the LCR. If a new value does not exist in the list, then the function returns the corresponding old value. Therefore, the returned list contains all existing new values and the old values where there are no new values. If `N` and `value_type` is `new`, then the function returns a list of all new values in the LCR without returning any old values. If `value_type` is `old` or if the `command_type` of the row LCR is not `UPDATE`, then the function ignores the value of the `use_old` parameter. `NULL` is not a valid specification for the `use_old` parameter.

GET_WHERE_CLAUSE Member Procedure

Gets a `WHERE` clause for the change that is encapsulated in the row LCR.

Use the `WHERE` clause returned by `GET_WHERE_CLAUSE` instead of using the `ROWID`, because the `ROWID` is not ANSI compatible. The generated `WHERE` clause might not match the `WHERE` clause in the original DML operation.

The `ROWID` of an `INSERT` statement is the `ROWID` of the new row created by the `INSERT`. The `WHERE` clause generated for an `INSERT` operation identifies the new row. Therefore, the generated `WHERE` clause includes all of the new values inserted.

For example, consider the following insert into the `hr.departments` table:

```
INSERT INTO hr.departments (
   department_id, department_name, manager_id, location_id)
   VALUES (10, 'HR', 20, 40);
```

The generated `WHERE` clause represents the row with the values `10`, `'HR'`, `20`, and `40`. Hence, the generated `WHERE` clause is the following:

```
WHERE "DEPARTMENT_ID" = 10 AND "DEPARTMENT_NAME" = 'HR' AND
      "MANAGER_ID" = 20 AND "LOCATION_ID" = 40
```

The `ROWID` of an `UPDATE` statement is the `ROWID` of the row that was updated. The `WHERE` clause generated for an `UPDATE` operation identifies the row after the `UPDATE` executes. The generated `WHERE` clause is based on the old and new values of the `UPDATE`.

For example, consider the following update to the `hr.departments` table:

```
UPDATE hr.departments SET department_name='Management'
 WHERE department_name='Administration' AND location_id = 20 AND
       manager_id = 30 AND department_id = 10;
```

The values of the row after the `UPDATE` are `10`, `'Management'`, `30`, and `20`. Hence, the generated `WHERE` clause to identify the row is the following:

```
WHERE "DEPARTMENT_ID" = 10 AND "DEPARTMENT_NAME" = 'MANAGEMENT' AND
      "MANAGER_ID" = 30 AND "LOCATION_ID" = 20
```

Notice that the new value is used for `"DEPARTMENT_NAME"`, because the new value is the value of the column after the `UPDATE`. For the rest of the columns, the old values are used.

The `ROWID` of a `DELETE` operation is the row that existed before it was deleted. The generated `WHERE` clause consists of all the old column values present in the `DELETE` operation.

LOB columns do not appear in generated `WHERE` clauses. The generated `WHERE` clause is not affected by the presence of LOB columns in the LCR.

This method is overloaded. The different functionality of each form of syntax is presented along with the definitions.

Syntax

The following procedure returns the `WHERE` clause of a SQL statement in a `CLOB` datatype.

```
MEMBER PROCEDURE GET_WHERE_CLAUSE(
   where_clause  IN/OUT  CLOB);
```

The following procedure returns the `WHERE` clause of a SQL statement with bind variables in a `CLOB` datatype.

```
MEMBER PROCEDURE GET_WHERE_CLAUSE(
   where_clause     IN/OUT  CLOB,
   variable_list    IN/OUT  LCR$_ROW_LIST,
   bind_var_syntax  IN      VARCHAR2  DEFAULT ':');
```

See Also:

LCR$_ROW_LIST Type

Parameters

Table 314-25 GET_WHERE_CLAUSE Procedure Parameters

Parameter	Description
`where_clause`	The `WHERE` clause of the SQL statement for the change that is encapsulated in the LCR
`variable_list`	The values for the bind variables in the order of the bind variables
`bind_var_syntax`	The syntax for the bind variables One of the following values is valid: • Specify `:`, the default, for bind values to be in the form `:1`, `:2`, and so on. • Specify `?` for bind values to be in the form `?`.

GET_XML_INFORMATION Member Function

Gets the XML information for the specified column.

The return value can be one of the following:

```
DBMS_LCR.NOT_XML   CONSTANT NUMBER := 1;
DBMS_LCR.XML_DOC   CONSTANT NUMBER := 2;
DBMS_LCR.XML_DIFF  CONSTANT NUMBER := 3;

DBMS_LCR.XML_DIFF  CONSTANT NUMBER := 3;
```

`DBMS_LCR.NOT_XML` indicates that the column is not an `XMLType` column.

`DBMS_LCR.XML_DOC` indicates that the column contains an XML document.

`DBMS_LCR.XML_DIFF` indicates that the column contains an XML document that describes a change made by an update operation. This XML document describes the differences in the column's XML document. The entire XML document is not replaced.

`DBMS_LCR.XML_DIFF` indicates that the column contains differences between old and new XML documents for an update operation.

Returns `NULL` if the specified column does not exist.

Syntax

```
MEMBER FUNCTION GET_XML_INFORMATION(
   column_name   IN   VARCHAR2)
RETURN NUMBER;
```

Parameter

Table 314-26 GET_XML_INFORMATION Function Parameter

Parameter	Description
`column_name`	The column name

IS_STATEMENT_LCR Member Function

This function is reserved for internal use only.

KEEP_COLUMNS Member Procedure

This procedure keeps a list of columns in a row LCR. The procedure deletes columns that are not in the list from the row LCR.

Syntax

```
MEMBER PROCEDURE KEEP_COLUMNS(
   column_list   IN   VARCHAR2,
   value_type    IN   VARCHAR2   DEFAULT '*');
```

Parameters

Table 314-27 KEEP_COLUMNS Procedure Parameters

Parameter	Description
`column_list`	The names of the columns kept for the row LCR Specify a comma-delimited list of type `VARCHAR2`. This procedure removes columns that are not in the list from the current row LCR.

Table 314-27 (Cont.) KEEP_COLUMNS Procedure Parameters

Parameter	Description
`value_type`	The type of value for which to keep the columns Specify `old` to keep the old values of the columns. An error is raised if the old values do not exist in the LCR. Specify `new` to keep the new values of the columns. An error is raised if the new values do not exist in the LCR. If `*` is specified, then the procedure keeps both the old and the new columns.

RENAME_COLUMN Member Procedure

Renames a column in a row LCR.

Syntax

```
MEMBER PROCEDURE RENAME_COLUMN(
   from_column_name  IN  VARCHAR2,
   to_column_name    IN  VARCHAR2,
   value_type        IN  VARCHAR2  DEFAULT '*');
```

Parameters

Table 314-28 RENAME_COLUMN Procedure Parameters

Parameter	Description
`from_column_name`	The existing column name
`to_column_name`	The new column name An error is raised if a column with the specified name already exists.
`value_type`	The type of value for which to rename the column Specify `old` to rename the old value of the column. An error is raised if the old value does not exist in the LCR. Specify `new` to rename the new value of the column. An error is raised if the new value does not exist in the LCR. If `*` is specified, then the procedure renames the column names for both old and new value. The procedure raises an error if either column value does not exist in the LCR.

SET_JSON_INFORMATION Member Procedure

Sets the JSON information for a column.

Syntax

```
MEMBER PROCEDURE SET_JSON_INFORMATION(
  self             IN OUT NOCOPY LCR$_ROW_RECORD,
  value_type       IN VARCHAR2,
  column_name      IN  VARCHAR2,
  json_information IN NUMBER);
```

Parameters

Table 314-29 SET_JSON_INFORMATION Procedure Parameters

Parameter	Description
`self`	
`value_type`	The type of value to set for the column. The supported values are: • `NEW` • `OLD`
`column_name`	The name of the column. An exception is raised if the column does not exists in the LCR.
`json_information`	The supported values are: • `DBMS_LCR.NOT_OSON` • `DBMS_LCR.OSON_DOC`

SET_LOB_INFORMATION Member Procedure

Sets LOB information for the column.

Note:

When you are processing a row LCR with a rule-based transformation, procedure DML handler, or error handler, you cannot use this member procedure.

Syntax

```
MEMBER PROCEDURE SET_LOB_INFORMATION(
  value_type       IN  VARCHAR2,
  column_name      IN  VARCHAR2,
  lob_information  IN  NUMBER);
```

Parameters

Table 314-30 SET_LOB_INFORMATION Procedure Parameters

Parameter	Description
`value_type`	The type of value to set for the column, either `old` or `new` Specify `old` only if `lob_information` is set to `DBMS_LCR.NOT_A_LOB`.
`column_name`	The name of the column. An exception is raised if the column value does not exist. You might need to set this parameter for non-LOB columns.

Table 314-30 (Cont.) SET_LOB_INFORMATION Procedure Parameters

Parameter	Description

`lob_information`	Specify one of the following values:
	`DBMS_LCR.NOT_A_LOB        CONSTANT NUMBER := 1;`
	`DBMS_LCR.NULL_LOB          CONSTANT NUMBER := 2;`
	`DBMS_LCR.INLINE_LOB        CONSTANT NUMBER := 3;`
	`DBMS_LCR.EMPTY_LOB         CONSTANT NUMBER := 4;`
	`DBMS_LCR.LOB_CHUNK         CONSTANT NUMBER := 5;`
	`DBMS_LCR.LAST_LOB_CHUNK    CONSTANT NUMBER := 6;`

SET_LOB_OFFSET Member Procedure

Sets the LOB offset for the specified column in the number of characters for `CLOB` columns and the number of bytes for `BLOB` columns.

Note:

When you are processing a row LCR with a rule-based transformation, procedure DML handler, or error handler, you cannot use this member procedure.

Syntax

```
MEMBER PROCEDURE SET_LOB_OFFSET(
   value_type    IN  VARCHAR2,
   column_name   IN  VARCHAR2,
   lob_offset    IN  NUMBER);
```

Parameters

Table 314-31 SET_LOB_OFFSET Procedure Parameters

Parameter	Description
`value_type`	The type of value to set for the column Currently, only `new` can be specified.
`column_name`	The column name An error is raised if the column value does not exist in the LCR.
`lob_offset`	The LOB offset number Valid values are `NULL` or a positive integer less than or equal to `DBMS_LOB.LOBMAXSIZE`.

SET_LOB_OPERATION_SIZE Member Procedure

Sets the operation size for the LOB column in the number of characters for `CLOB` columns and bytes for `BLOB` columns.

Note:

When you are processing a row LCR with a rule-based transformation, procedure DML handler, or error handler, you cannot use this member procedure.

Syntax

```
MEMBER PROCEDURE SET_LOB_OPERATION_SIZE(
  value_type          IN  VARCHAR2,
  column_name         IN  VARCHAR2,
  lob_operation_size  IN  NUMBER);
```

Parameters

Table 314-32 SET_LOB_OPERATION_SIZE Procedure Parameters

Parameter	Description
`value_type`	The type of value to set for the column Currently, only `new` can be specified.
`column_name`	The name of the LOB column An exception is raised if the column value does not exist in the LCR.
`lob_operation_size`	If `lob_information` for the LOB is or will be `DBMS_LCR.LAST_LOB_CHUNK`, then this parameter can be set to either a valid `LOB ERASE` value or a valid `LOB TRIM` value. A `LOB ERASE` value must be a positive integer less than or equal to `DBMS_LOB.LOBMAXSIZE`. A `LOB TRIM` value must be a nonnegative integer less than or equal to `DBMS_LOB.LOBMAXSIZE`. Otherwise, set to `NULL`.

SET_ROW_TEXT Member Procedure

This procedure is reserved for internal use only.

SET_VALUE Member Procedure

Overwrites the old or new value of the specified column.

One reason to overwrite an old value for a column is to resolve an error that resulted from a conflict.

Note:

To add a column to a row LCR, run `ADD_COLUMN`.

See Also:

ADD_COLUMN Member Procedure

Considerations for LOB Columns

When processing a row LCR with LOB columns with a procedure DML handler or error handler, and the handler is using LOB assembly (the `assemble_lobs` parameter is set to `TRUE` for the handler), you can use this member procedure in the handler procedure on a LOB column in a row LCR. If `assemble_lobs` is set to `FALSE` for the handler, then you cannot use this member procedure on a LOB column.

To use a DML or error handler to set the value of a LOB column, specify the LOB locator for the `column_value` parameter in the member procedure. The `SET_VALUE` member procedure verifies that an `ANYDATA` encapsulated LOB locator is processed with a DML or error handler that is using LOB assembly. An error is raised under the following conditions:

- The handler attempts to enqueue a row LCR with an `ANYDATA` encapsulated LOB locator.
- An attempt is made to set a LOB column incorrectly.

If an error is raised because of one of these conditions, then the transaction that includes the row LCR is moved to the error queue, and the LOB is represented by the original (nonassembled) row LCRs.

Note:

- Database compatibility must be `10.2.0` or higher to use LOB assembly.
- When you are processing a row LCR with a rule-based transformation, you cannot use this member procedure on a LOB column.
- When you are processing a row LCR with a rule-based transformation, procedure DML handler, or error handler, you cannot use this member procedure on a `LONG` or `LONG RAW` column.

Considerations for XMLType Columns

When processing a row LCR with `XMLType` columns with a procedure DML handler or error handler, any `XMLType` columns and LOB columns in the LCR are always assembled using LOB assembly. You can use this member procedure in the handler procedure on a row LCR that contains one or more `XMLType` columns.

To use a DML or error handler to set the value an `XMLType` column, specify the `XMLType` for the `column_value` parameter. The `SET_VALUE` member procedure verifies that an `ANYDATA` encapsulated `XMLType` is processed with a DML or error handler. An error is raised under the following conditions:

- The handler attempts to enqueue a row LCR with an `ANYDATA` encapsulated `XMLType`.
- An attempt is made to set a `XMLType` column incorrectly.

If an error is raised because of one of these conditions, then the transaction that includes the row LCR is moved to the error queue, and the `XMLType` column is represented by the original (nonassembled) row LCRs.

Note:

- Database compatibility must be `11.1.0` or higher to process row LCRs with `XMLType` columns.
- When you are processing a row LCR with a rule-based transformation, you cannot use this member procedure on `XMLType` columns.

Syntax

```
MEMBER PROCEDURE SET_VALUE(
   value_type     IN  VARCHAR2,
   column_name    IN  VARCHAR2,
   column_value   IN  ANYDATA);
```

Parameters

Table 314-33 SET_VALUE Procedure Parameters

Parameter	Description
`value_type`	The type of value to set Specify `old` to set the old value of the column. Specify `new` to set the new value of the column.
`column_name`	The column name An error is raised if the specified `column_value` does not exist in the LCR for the specified `column_type`.
`column_value`	The new value of the column If `NULL` is specified, then this procedure raises an error. To set the value to `NULL`, encapsulate the `NULL` in an `ANYDATA` instance. If the member procedure is used in a procedure DML handler or error handler that uses LOB assembly, then specify a LOB locator for LOB columns.

SET_VALUES Member Procedure

Replaces all old values or all new values for the LCR, depending on the value type specified.

Considerations for LOB Columns

You can use this procedure when processing a row LCR with LOB columns with a procedure DML handler or error handler. If the handler is using LOB assembly (the `assemble_lobs` parameter is set to `TRUE` for the handler), then you can use this member procedure in the handler procedure. If `assemble_lobs` is set to `FALSE` for the handler, then you cannot use this member procedure on a row LCR.

To use a DML or error handler to set the value of one or more LOB columns in a row LCR, specify a LOB locator for each LOB column in the `value_list` parameter. The `SET_VALUES` member procedure verifies that an `ANYDATA` encapsulated LOB locator is processed with a DML or error handler that is using LOB assembly. An error is raised under the following conditions:

- The handler attempts to enqueue a row LCR with an `ANYDATA` encapsulated LOB locator.
- An attempt is made to set a LOB column incorrectly.

If an error is raised because of one of these conditions, then the transaction that includes the row LCR is moved to the error queue, and the LOB columns are represented by the original (nonassembled) row LCRs.

Note:

- Database compatibility must be `10.2.0` or higher to use LOB assembly.
- When you are processing a row LCR with a rule-based transformation, you cannot use this member procedure on LOB columns.
- When you are processing a row LCR with a rule-based transformation, procedure DML handler, or error handler, you cannot use this member procedure on `LONG` or `LONG RAW` columns.

Considerations for XMLType Columns

When processing a row LCR with `XMLType` columns with a procedure DML handler or error handler, any `XMLType` and LOB columns in the LCR are always assembled using LOB assembly. You can use this member procedure in the handler procedure on a row LCR that contains one or more `XMLType` columns.

To use a DML or error handler to set the value of one or more `XMLType` columns in a row LCR, specify an `XMLType` for each `XMLType` column in the `value_list` parameter. The `SET_VALUES` member procedure verifies that an `ANYDATA` encapsulated `XMLType` is processed with a DML or error handler. An error is raised under the following conditions:

- The handler attempts to enqueue a row LCR with an `ANYDATA` encapsulated `XMLType`.
- An attempt is made to set a `XMLType` incorrectly.

If an error is raised because of one of these conditions, then the transaction that includes the row LCR is moved to the error queue, and the `XMLType` columns are represented by the original (nonassembled) row LCRs.

Note:

- Database compatibility must be `11.1.0` or higher to process row LCRs with `XMLType` columns.
- When you are processing a row LCR with a rule-based transformation, you cannot use this member procedure on `XMLType` columns.

Syntax

```
MEMBER PROCEDURE SET_VALUES(
   value_type   IN   VARCHAR2,
   value_list   IN   SYS.LCR$_ROW_LIST);
```

Parameters

Table 314-34 SET_VALUES Procedure Parameters

Parameter	Description
`value_type`	The type of values to replace Specify `old` to replace the old values. Specify `new` to replace the new values.
`value_list`	List of values to replace the existing list Use a `NULL` or an empty list to remove all values. If the member procedure is used in a procedure DML handler or error handler that uses LOB assembly, then specify one or more LOB locators for LOB columns.

SET_XML_INFORMATION Member Procedure

Sets the XML information for the column.

Syntax

```
MEMBER PROCEDURE SET_XML_INFORMATION(
   column_name      IN  VARCHAR2,
   xml_information  IN  NUMBER);
```

Parameters

Table 314-35 SET_XML_INFORMATION Procedure Parameters

Parameter	Description
`column_name`	The name of the column An exception is raised if the column value does not exist in the LCR.

`xml_information`	Specify one of the following values:
	`DBMS_LCR.NOT_XML   CONSTANT NUMBER := 1;`
	`DBMS_LCR.XML_DOC   CONSTANT NUMBER := 2;`
	`DBMS_LCR.XML_DIFF  CONSTANT NUMBER := 3;`
	
	`DBMS_LCR.XML_DIFF  CONSTANT NUMBER := 3;`
	`DBMS_LCR.NOT_XML` indicates that the column is not an `XMLType` column.
	`DBMS_LCR.XML_DOC` indicates that the column contains an XML document.
	`DBMS_LCR.XML_DIFF` indicates that the column contains differences between old and new XML documents for an update operation.
	`DBMS_LCR.XML_DIFF` indicates that the column contains an XML document that describes a change made by an update operation. This XML document describes the differences in the column's XML document. The entire XML document is not replaced.

Common Subprograms for LCR$_DDL_RECORD and LCR$_ROW_RECORD

These functions and procedures are common to both the `LCR$_DDL_RECORD` and `LCR$_ROW_RECORD` type.

See Also:

For descriptions of the subprograms for these types that are exclusive to each type:

- "LCR$_DDL_RECORD Type"
- "LCR$_ROW_RECORD Type"

Table 314-36 Summary of Common Subprograms for DDL and Row LCR Types

Subprogram	Description
GET_COMMAND_TYPE Member Function	Gets the command type of the logical change record (LCR)
GET_COMMIT_SCN Member Function	Gets the commit system change number (SCN) of the transaction to which the current LCR belongs
GET_COMMIT_SCN_FROM_POSITION Static Function	Gets the commit SCN of a transaction from the input position, which is generated by an XStream outbound server
GET_COMMIT_TIME	Gets the commit time of the transaction to which the current LCR belongs
GET_COMPATIBLE Member Function	Gets the minimal database compatibility required to support the LCR
GET_EXTRA_ATTRIBUTE Member Function	Gets the value for the specified extra attribute in the LCR
GET_OBJECT_NAME Member Function	Gets the name of the object that is changed by the LCR
GET_OBJECT_OWNER Member Function	Gets the owner of the object that is changed by the LCR
GET_POSITION Member Function	Gets the position of the current LCR
GET_ROOT_NAME Member Function	Gets the global name of the root for a CDB.
GET_SCN Member Function	Gets the SCN of the LCR
GET_SCN_FROM_POSITION Static Function	Gets the SCN from the input position, which is generated by an XStream outbound server
GET_SOURCE_DATABASE_NAME Member Function	Gets the source database name.
GET_SOURCE_TIME Member Function	Gets the time when the change in an LCR captured by a capture process was generated in the redo log of the source database, or the time when a persistent LCR was created
GET_TAG Member Function	Gets the tag for the LCR
GET_THREAD_NUMBER Member Function	Gets the thread number of the database instance that made the change that is encapsulated in the LCR

Table 314-36 (Cont.) Summary of Common Subprograms for DDL and Row LCR Types

Subprogram	Description
GET_TRANSACTION_ID Member Function	Gets the transaction identifier of the LCR
IS_NULL_TAG Member Function	Returns `Y` if the tag for the LCR is `NULL`, or returns `N` if the tag for the LCR is not `NULL`
SET_COMMAND_TYPE Member Procedure	Sets the command type in the LCR
SET_EXTRA_ATTRIBUTE Member Procedure	Sets the value for the specified extra attribute in the LCR
SET_OBJECT_NAME Member Procedure	Sets the name of the object that is changed by the LCR
SET_OBJECT_OWNER Member Procedure	Sets the owner of the object that is changed by the LCR
SET_ROOT_NAME Member Procedure	Sets the global name of the root in a CDB.
SET_SOURCE_DATABASE_NAME Member Procedure	Sets the source database name of the object that is changed by the LCR
SET_TAG Member Procedure	Sets the tag for the LCR

GET_COMMAND_TYPE Member Function

Gets the command type of the LCR.

See Also:

The "SQL Command Codes" table in the *Oracle Call Interface Programmer's Guide* for a complete list of command types

Syntax

```
MEMBER FUNCTION GET_COMMAND_TYPE()
RETURN VARCHAR2;
```

GET_COMMIT_SCN Member Function

Gets the commit system change number (SCN) of the transaction to which the current LCR belongs.

The commit SCN for a transaction is available only during apply or during error transaction execution. This function can be used only in a procedure DML handler, DDL handler, or error handler.

The commit SCN might not be available for an LCR that is part of an incomplete transaction. For example, persistent LCRs might not have a commit SCN. If the commit SCN is not available for an LCR, then this function returns `NULL`.

Syntax

```
MEMBER FUNCTION GET_COMMIT_SCN()
RETURN NUMBER;
```

GET_COMMIT_SCN_FROM_POSITION Static Function

Gets the commit system change number (SCN) of a transaction from the input position, which is generated by an XStream outbound server.

Syntax

```
STATIC FUNCTION GET_COMMIT_SCN_FROM_POSITION(
   position  IN  RAW)
RETURN NUMBER;
```

Parameters

Table 314-37 GET_COMMIT_SCN_FROM_POSITION Function Parameter

Parameter	Description
`position`	The position You can obtain the position by using the `GET_POSITION` member function or by querying the `DBA_XSTREAM_OUTBOUND_PROGRESS` data dictionary view.

Note:

Using XStream requires purchasing a license for the Oracle GoldenGate product. See *Oracle Database XStream Guide*.

GET_COMMIT_TIME

Gets the commit time of the transaction to which the current LCR belongs.

The commit time for a transaction is available only during apply or during error transaction execution. This function can be used only in a procedure DML handler, DDL handler, or error handler.

The commit time might not be available for an LCR that is part of an incomplete transaction. For example, persistent LCRs might not have a commit time. If the commit time is not available for an LCR, then this function returns `NULL`.

Syntax

```
MEMBER FUNCTION GET_COMMIT_TIME()
RETURN DATE;
```

GET_COMPATIBLE Member Function

Gets the minimal database compatibility required to support the LCR. You control the compatibility of an Oracle database using the `COMPATIBLE` initialization parameter.

The return value for this function can be one of the following:

Return Value	COMPATIBLE Initialization Parameter Equivalent
`DBMS_STREAMS.COMPATIBLE_9_2`	`9.2.0`
`DBMS_STREAMS.COMPATIBLE_10_1`	`10.1.0`

Return Value	COMPATIBLE Initialization Parameter Equivalent
DBMS_STREAMS.COMPATIBLE_10_2	10.2.0
DBMS_STREAMS.COMPATIBLE_11_1	11.1.0
DBMS_STREAMS.COMPATIBLE_11_2	11.2.0

DDL LCRs always return DBMS_STREAMS.COMPATIBLE_9_2.

You can use the following functions in the DBMS_STREAMS package for constant compatibility return values:

- The COMPATIBLE_9_2 function returns the DBMS_STREAMS.COMPATIBLE_9_2 constant.
- The COMPATIBLE_10_1 function returns DBMS_STREAMS.COMPATIBLE_10_1 constant.
- The COMPATIBLE_10_2 function returns DBMS_STREAMS.COMPATIBLE_10_2 constant.
- The COMPATIBLE_11_1 function returns DBMS_STREAMS.COMPATIBLE_11_1 constant.
- The COMPATIBLE_11_2 function returns DBMS_STREAMS.COMPATIBLE_11_2 constant.
- The MAX_COMPATIBLE function returns an integer that is greater than the highest possible compatibility constant for the current release of Oracle Database.

You can use these functions with the GET_COMPATIBLE member function for an LCR in rule conditions and apply handlers.

See Also:

- *Oracle Database Reference* and *Oracle Database Upgrade Guide* for more information about the COMPATIBLE initialization parameter

Syntax

```
MEMBER FUNCTION GET_COMPATIBLE()
RETURN NUMBER;
```

GET_EXTRA_ATTRIBUTE Member Function

Gets the value for the specified extra attribute in the LCR. The returned extra attribute is contained within an ANYDATA instance. You can use the INCLUDE_EXTRA_ATTRIBUTE procedure in the DBMS_CAPTURE_ADM package to instruct a capture process to capture one or more extra attributes.

See Also:

INCLUDE_EXTRA_ATTRIBUTE Procedure

Syntax

```
MEMBER FUNCTION GET_EXTRA_ATTRIBUTE(
   attribute_name   IN   VARCHAR2)
RETURN ANYDATA;
```

Parameters

Table 314-38 GET_EXTRA_ATTRIBUTE Function Parameter

Parameter	Description
`attribute_name`	The name of the extra attribute to return Valid names are: • `row_id` The rowid of the row changed in a row LCR. This attribute is not included in DDL LCRs, nor in row LCRs for index-organized tables. The type is `UROWID`. • `serial#` The serial number of the session that performed the change captured in the LCR. The type is `NUMBER`. • `session#` The identifier of the session that performed the change captured in the LCR. The type is `NUMBER`. • `thread#` The thread number of the instance in which the change captured in the LCR was performed. Typically, the thread number is relevant only in an Oracle Real Application Clusters (Oracle RAC) environment. The type is `NUMBER`. • `tx_name` The name of the transaction that includes the LCR. The type is `VARCHAR2`. • `username` The name of the current user who performed the change captured in the LCR. The type is `VARCHAR2`. An error is raised if the specified `attribute_name` is not valid. If no value exists for the specified extra attribute, then the function returns a `NULL`. **See Also:** *Oracle Database PL/SQL Language Reference* for more information about the current user

GET_OBJECT_NAME Member Function

Gets the name of the object that is changed by the LCR.

Syntax

```
MEMBER FUNCTION GET_OBJECT_NAME()
RETURN VARCHAR2;
```

GET_OBJECT_OWNER Member Function

Gets the owner of the object that is changed by the LCR.

Syntax

```
MEMBER FUNCTION GET_OBJECT_OWNER()
RETURN VARCHAR2;
```

GET_POSITION Member Function

Gets the position of the current LCR. The position uniquely identifies each LCR. The position strictly increases within each transaction and across transactions.

LCR position is commonly used in XStream configurations.

Syntax

```
MEMBER FUNCTION GET_POSITION()
RETURN RAW;
```

> **Note:**
>
> Using XStream requires purchasing a license for the Oracle GoldenGate product. See *Oracle Database XStream Guide*.

GET_ROOT_NAME Member Function

Gets the global name of the root in a CDB, which is the root name for the LCR.

Syntax

```
MEMBER FUNCTION GET_ROOT_NAME()
RETURN VARCHAR2;
```

GET_SCN Member Function

Gets the system change number (SCN) of the LCR.

Syntax

```
MEMBER FUNCTION GET_SCN()
RETURN NUMBER;
```

GET_SCN_FROM_POSITION Static Function

Gets the system change number (SCN) from the input position, which is generated by an XStream outbound server.

Syntax

```
STATIC FUNCTION GET_SCN_FROM_POSITION(
   position  IN  RAW)
RETURN NUMBER;
```

Parameters

Table 314-39 GET_SCN_FROM_POSITION Function Parameter

Parameter	Description
`position`	The position You can obtain the position by using the `GET_POSITION` member function or by querying the `DBA_XSTREAM_OUTBOUND_PROGRESS` data dictionary view.

Note:

Using XStream requires purchasing a license for the Oracle GoldenGate product. See *Oracle Database XStream Guide*.

GET_SOURCE_DATABASE_NAME Member Function

Gets the global name of the source database. The source database is the database where the change occurred.

Syntax

```
MEMBER FUNCTION GET_SOURCE_DATABASE_NAME()
RETURN VARCHAR2;
```

GET_SOURCE_TIME Member Function

Gets the time when the change in an LCR captured by a capture process was generated in the redo log of the source database, or the time when a persistent LCR was created.

Syntax

```
MEMBER FUNCTION GET_SOURCE_TIME()
RETURN DATE;
```

GET_TAG Member Function

Gets the tag for the LCR. An LCR tag is a binary tag that enables tracking of the LCR. For example, this tag can be used to determine the original source database of the DML or DDL change when apply forwarding is used.

Syntax

```
MEMBER FUNCTION GET_TAG()
RETURN RAW;
```

GET_THREAD_NUMBER Member Function

Gets the thread number of the database instance that made the change that is encapsulated in the LCR. Typically, the thread number is relevant in an Oracle Real Application Clusters configuration.

See Also:

Oracle Real Application Clusters Administration and Deployment Guide

Syntax

```
MEMBER FUNCTION GET_THREAD_NUMBER()
RETURN NUMBER;
```

GET_TRANSACTION_ID Member Function

Gets the transaction identifier of the LCR.

Syntax

```
MEMBER FUNCTION GET_TRANSACTION_ID()
RETURN VARCHAR2;
```

IS_NULL_TAG Member Function

Returns `Y` if the tag for the LCR is `NULL`, or returns `N` if the tag for the LCR is not `NULL`.

Syntax

```
MEMBER FUNCTION IS_NULL_TAG()
RETURN VARCHAR2;
```

SET_COMMAND_TYPE Member Procedure

Sets the command type in the LCR. If the command type specified cannot be interpreted, then this procedure raises an error. For example, changing `INSERT` to `GRANT` would raise an error.

See Also:

- The description of the `command_type` parameter in LCR$_DDL_RECORD Constructor Function Parameters
- The description of the command_type parameter in LCR$_ROW_RECORD Type
- The "SQL Command Codes" table in the *Oracle Call Interface Programmer's Guide* for a complete list of command types

Syntax

```
MEMBER PROCEDURE SET_COMMAND_TYPE(
   command_type   IN   VARCHAR2);
```

Parameter

Table 314-40 SET_COMMAND_TYPE Procedure Parameter

Parameter	Description
`command_type`	The command type Set this parameter to a non-`NULL` value.

SET_EXTRA_ATTRIBUTE Member Procedure

Sets the value for the specified extra attribute in the LCR. You can use the `INCLUDE_EXTRA_ATTRIBUTE` procedure in the `DBMS_CAPTURE_ADM` package to instruct a capture process to capture one or more extra attributes.

See Also:

INCLUDE_EXTRA_ATTRIBUTE Procedure

Syntax

```
MEMBER PROCEDURE SET_EXTRA_ATTRIBUTE(
   attribute_name   IN  VARCHAR2,
   attribute_value  IN  ANYDATA);
```

Parameters

Table 314-41 SET_EXTRA_ATTRIBUTE Procedure Parameter

Parameter	Description
`attribute_name`	The name of the extra attribute to set Valid names are: • `row_id` The rowid of the row changed in a row LCR. This attribute is not included in DDL LCRs, nor in row LCRs for index-organized tables. The type is `VARCHAR2`. • `serial#` The serial number of the session that performed the change captured in the LCR. The type is `NUMBER`. • `session#` The identifier of the session that performed the change captured in the LCR. The type is `NUMBER`. • `thread#` The thread number of the instance in which the change captured in the LCR was performed. Typically, the thread number is relevant only in an Oracle Real Application Clusters (Oracle RAC) environment. The type is `NUMBER`. • `tx_name` The name of the transaction that includes the LCR. The type is `VARCHAR2`. • `username` The name of the current user who performed the change captured in the LCR. The type is `VARCHAR2`. An error is raised if the specified `attribute_name` is not valid. **See Also:** *Oracle Database PL/SQL Language Reference* for more information about the current user
`attribute_value`	The value to which the specified extra attribute is set If set to `NULL`, then this procedure removes the specified extra attribute from the LCR. To set to `NULL`, encapsulate the `NULL` in an `ANYDATA` instance.

SET_OBJECT_NAME Member Procedure

Sets the name of the object that is changed by the LCR.

Syntax

```
MEMBER PROCEDURE SET_OBJECT_NAME(
   object_name  IN  VARCHAR2);
```

Parameter

Table 314-42 SET_OBJECT_NAME Procedure Parameter

Parameter	Description
`object_name`	The name of the object

SET_OBJECT_OWNER Member Procedure

Sets the owner of the object that is changed by the LCR.

Syntax

```
MEMBER PROCEDURE SET_OBJECT_OWNER(
   object_owner  IN  VARCHAR2);
```

Parameter

Table 314-43 SET_OBJECT_OWNER Procedure Parameter

Parameter	Description
`object_owner`	The schema that contains the object

SET_ROOT_NAME Member Procedure

Sets the global name of the root in a CDB. The setting is the root name for the LCR.

Syntax

```
MEMBER PROCEDURE SET_ROOT_NAME(
   root_name  IN  VARCHAR2);
```

Parameter

Table 314-44 SET_ROOT_NAME Procedure Parameter

Parameter	Description
`root_name`	The global name of the root.

SET_SOURCE_DATABASE_NAME Member Procedure

Sets the source database name of the object that is changed by the LCR.

Syntax

```
MEMBER PROCEDURE SET_SOURCE_DATABASE_NAME(
   source_database_name  IN  VARCHAR2);
```

Parameter

Table 314-45   SET_SOURCE_DATABASE_NAME Procedure Parameter

Parameter	Description
`source_database_name`	The source database of the change If you do not include the domain name, then the procedure appends the local domain to the database name automatically. For example, if you specify `DBS1` and the local domain is `EXAMPLE.COM`, then the procedure specifies `DBS1.EXAMPLE.COM` automatically. Set this parameter to a non-`NULL` value.

SET_TAG Member Procedure

Sets the tag for the LCR. An LCR tag is a binary tag that enables tracking of the LCR. For example, this tag can be used to determine the original source database of the change when apply forwarding is used.

Syntax

```
MEMBER PROCEDURE SET_TAG(
   tag  IN  RAW);
```

Parameter

Table 314-46   SET_TAG Procedure Parameter

Parameter	Description
`tag`	The binary tag for the LCR The size limit for a tag value is two kilobytes.

LCR$_ROW_LIST Type

This type identifies a list of column values for a row in a table.

It uses the `LCR$_ROW_UNIT` type and is used in the `LCR$_ROW_RECORD` type.

See Also:

- LCR$_ROW_UNIT Type
- LCR$_ROW_RECORD Type

Syntax

```
CREATE TYPE SYS.LCR$_ROW_LIST AS TABLE OF SYS.LCR$_ROW_UNIT
/
```

LCR$_ROW_UNIT Type

This type identifies the value for a column in a row.

It is used in the `LCR$_ROW_LIST` type.

See Also:

LCR$_ROW_LIST Type

Syntax

```
CREATE TYPE LCR$_ROW_UNIT AS OBJECT (
  column_name         VARCHAR2(4000),
  data                ANYDATA,
  lob_information     NUMBER,
  lob_offset          NUMBER,
  lob_operation_size  NUMBER
  long_information    NUMBER);
/
```

Attributes

Table 314-47 LCR$_ROW_UNIT Attributes

Attribute	Description
`column_name`	The name of the column
`data`	The data contained in the column

`lob_information`	Contains the LOB information for the column and contains one of the following values:
	`DBMS_LCR.NOT_A_LOB        CONSTANT NUMBER := 1;`
	`DBMS_LCR.NULL_LOB         CONSTANT NUMBER := 2;`
	`DBMS_LCR.INLINE_LOB       CONSTANT NUMBER := 3;`
	`DBMS_LCR.EMPTY_LOB        CONSTANT NUMBER := 4;`
	`DBMS_LCR.LOB_CHUNK        CONSTANT NUMBER := 5;`
	`DBMS_LCR.LAST_LOB_CHUNK   CONSTANT NUMBER := 6;`

`lob_offset`	The LOB offset specified in the number of characters for `CLOB` columns and the number of bytes for `BLOB` columns Valid values are `NULL` or a positive integer less than or equal to `DBMS_LOB.LOBMAXSIZE`.
`lob_operation_size`	If `lob_information` for the LOB is `DBMS_LCR.LAST_LOB_CHUNK`, then this parameter can be set to either a valid `LOB ERASE` value or a valid `LOB TRIM` value. A `LOB ERASE` value must be a positive integer less than or equal to `DBMS_LOB.LOBMAXSIZE`. A `LOB  TRIM` value must be a nonnegative integer less than or equal to `DBMS_LOB.LOBMAXSIZE`. If `lob_information` is not `DBMS_LCR.LAST_LOB_CHUNK` and for all other operations, is `NULL`.

Table 314-47 (Cont.) LCR$_ROW_UNIT Attributes

Attribute	Description

`long_information`	Contains the `LONG` information for the column and contains one of the following values:
	`DBMS_LCR.not_a_long CONSTANT NUMBER := 1;`
	`DBMS_LCR.null_long CONSTANT NUMBER := 2;`
	`DBMS_LCR.inline_long CONSTANT NUMBER := 3;`
	`DBMS_LCR.long_chunk CONSTANT NUMBER := 4;`
	`DBMS_LCR.last_long_chunk CONSTANT NUMBER := 5;`