18
External Tables Examples for Oracle Database

Learn from these examples how to use the `ORACLE_LOADER`, `ORACLE_DATAPUMP`,`ORACLE_HDFS`, `ORACLE_HIVE` access drivers, and to use external tables with vector data.

- Using the ORACLE_LOADER Access Driver to Create Partitioned External Tables
  This topic describes using the `ORACLE_LOADER` access driver to create partitioned external tables.
- Using the ORACLE_LOADER Access Driver to Create Partitioned Hybrid Tables
  This topic describes using the `ORACLE_LOADER` access driver to create partitioned hybrid tables.
- Using the ORACLE_DATAPUMP Access Driver to Create Partitioned External Tables
  See how you can use ORACLE_DATAPUMP access driver to create a subpartitioned external table, and partition tables with virtual columns..
- Using the ORACLE_BIGDATA Access Driver to Create Partitioned External Tables
  The example in this section shows how to create a subpartitioned external table.
- Using the ORA_PARTITION_VALIDATION Function to Validate Partitioned External Tables
  To confirm if a row in a partitioned external table is in the correct partition, use the `ORA_PARTITION_VALIDATION` function.
- Using SQL*Loader for External Tables with Partition Values in File Paths
  To enhance management of large numbers of data files in object stores, you can use external table partitioning with folder names as part of the filepaths. External table columns also can return the filename of the source file for each row.
- Loading LOBs with External Tables
  External tables are particularly useful for loading large numbers of records from a single file, so that each record appears in its own row in the table.
- Loading CSV Files From External Tables
  This topic provides examples of how to load CSV files from external tables under various conditions.
- Using Vector Data Types in External Tables
  See examples of how you can use Vector data types in external tables for vector similarity searches.

18.1 Using the ORACLE_LOADER Access Driver to Create Partitioned External Tables

This topic describes using the `ORACLE_LOADER` access driver to create partitioned external tables.

Example 18-1 Using ORACLE_LOADER to Create a Partitioned External Table

This example assumes there are four data files with the following content:

```
p1a.dat:
1, AAAAA Plumbing,01372,
```

```
28, Sparkly Laundry,78907,
13, Andi's Doughnuts,54570,

p1b.dat:
51, DIY Supplies,61614,
87, Fast Frames,22201,
89, Friendly Pharmacy,89901,

p2.dat:
121, Pleasant Pets,33893,
130, Bailey the Bookmonger,99915,
105, Le Bistrot du Chat Noir,94114,

p3.dat:
210, The Electric Eel Diner,07101,
222, Everyt'ing General Store,80118,
231, Big Rocket Market,01754,
```

There are three fields in the data file: `CUSTOMER_NUMBER`, `CUSTOMER_NAME` and `POSTAL_CODE`. The external table uses range partitioning on `CUSTOMER_NUMBER` to create three partitions.

- Partition 1 is for `customer_number` less than 100
- Partition 2 is for `customer_number` less than 200
- Partition 3 is for `customer_number` less than 300

Note that the first partition has two data files while the other partitions only have one. The following is the output from SQLPlus for creating the file.

```
SQL> create table customer_list_xt
  2    (CUSTOMER_NUMBER number, CUSTOMER_NAME VARCHAR2(50), POSTAL_CODE
CHAR(5))
  3  organization external
  4    (type oracle_loader default directory def_dir1)
  5  partition by range(CUSTOMER_NUMBER)
  6  (
  7    partition p1 values less than (100) location('p1a.dat', 'p1b.dat'),
  8    partition p2 values less than (200) location('p2.dat'),
  9    partition p3 values less than (300) location('p3.dat')
 10  );

Table created.
SQL>
```

The following is the output from `SELECT *` for the entire table:

```
SQL> select customer_number, customer_name, postal_code
  2    from customer_list_xt
  3    order by customer_number;

CUSTOMER_NUMBER CUSTOMER_NAME                                      POSTA
--------------- -------------------------------------------------- -----
              1  AAAAA Plumbing                                     01372
             13  Andi's Doughnuts                                  54570
             28  Sparkly Laundry                                   78907
```

```
            51  DIY Supplies                                        61614
            87  Fast Frames                                         22201
            89  Friendly Pharmacy                                   89901
           105  Le Bistrot du Chat Noir                             94114
           121  Pleasant Pets                                       33893
           130  Bailey the Bookmonger                               99915
           210  The Electric Eel Diner                              07101
           222  Everyt'ing General Store                            80118
           231  Big Rocket Market                                   01754

12 rows selected.

SQL>
```

The following query should only read records from the first partition:

```
SQL> select customer_number, customer_name, postal_code
  2    from customer_list_xt
  3    where customer_number < 20
  4    order by customer_number;

CUSTOMER_NUMBER CUSTOMER_NAME                                       POSTA
--------------- --------------------------------------------------- -----
             1  AAAAA Plumbing                                      01372
            13  Andi's Doughnuts                                    54570

2 rows selected.

SQL>
```

The following query specifies the partition to read as part of the `SELECT` statement.

```
SQL> select customer_number, customer_name, postal_code
  2    from customer_list_xt partition (p1)
  3    order by customer_number;

CUSTOMER_NUMBER CUSTOMER_NAME                                       POSTA
--------------- --------------------------------------------------- -----
             1  AAAAA Plumbing                                      01372
            13  Andi's Doughnuts                                    54570
            28  Sparkly Laundry                                     78907
            51  DIY Supplies                                        61614
            87  Fast Frames                                         22201
            89  Friendly Pharmacy                                   89901

6 rows selected.

SQL>
```

18.2 Using the ORACLE_LOADER Access Driver to Create Partitioned Hybrid Tables

This topic describes using the `ORACLE_LOADER` access driver to create partitioned hybrid tables.

Hybrid Partitioned Tables is a feature that extends Oracle Partitioning by allowing some partitions to reside in database segments and some partitions in external files or sources. This significantly enhances functionality of partitioning for Big Data SQL where large portions of a table can reside in external partitions.

Example 18-2 Example

Here is an example of a statement for creating a partitioned hybrid l table:

```
CREATE TABLE hybrid_pt (time_id date, customer number)
  TABLESPACE TS1
  EXTERNAL PARTITION ATTRIBUTES (TYPE ORACLE_LOADER
                                 DEFAULT DIRECTORY data_dir0
                                 ACCESS PARAMETERS(FIELDS TERMINATED BY ',')
                                 REJECT LIMIT UNLIMITED)
PARTITION by range (time_id)
(
 PARTITION century_18 VALUES LESS THAN ('01-01-1800')
    EXTERNAL,                                                  <-- empty
external partition
  PARTITION century_19 VALUES LESS THAN ('01-01-1900')
    EXTERNAL DEFAULT DIRECTORY data_dir1 LOCATION (‘century19_data.txt'),
  PARTITION century_20 VALUES LESS THAN ('01-01-2000')
    EXTERNAL LOCATION (‘century20_data.txt'),
  PARTITION year_2000 VALUES LESS THAN ('01-01-2001') TABLESPACE TS2,
  PARTITION pmax VALUES LESS THAN (MAXVALUE)
);
```

In this example, the table contains both internal and external partitions. The default tablespace for internal partitions in the table is `TS1`. An `EXTERNAL PARTITION ATTRIBUTES` clause is added for specifying parameters that apply, at the table level, to the external partitions in the table. The clause is mandatory for hybrid partitioned tables. In this case, external partitions are accessed through the `ORACLE_LOADER` access driver, and the parameters required by the access driver are specified in the clause. At the partition level, an `EXTERNAL` clause is specified in each external partition, along with any external parameters applied to the partition.

In this example, `century_18`, `century_19`, and `century_20` are external partitions. `century_18` is an empty partition since it does not contain a location. The default directory for partition `century_19` is`data_dir1`, overriding the table level default directory. The partition has a location `data_dir1:century19_data.txt`. Partition`century_20` has location `data_dir0:century20_data.txt`, since the table level default directory is applied to a location when a default directory is not specified in a partition. Partitions `year_2000` and `pmax` are internal partitions. Partition `year_2000`has a `tablespace TS2`. When a partition has no`EXTERNAL` clause or external parameters specified in it, it is assumed to be an internal partition by default.

18.3 Using the ORACLE_DATAPUMP Access Driver to Create Partitioned External Tables

See how you can use ORACLE_DATAPUMP access driver to create a subpartitioned external table, and partition tables with virtual columns..

> **Note:**
>
> Starting with Oracle Database 23ai, the `ORACLE_DATAPUMP` access driver provides interval, auto-list, and composite partitioning options for hybrid partitioned tables (HyPT) support. For more information, see *Oracle Database VLDB and Partitioning Guide*

Example 18-3 Using the ORACLE_DATAPUMP Access Driver to Create Partitioned External Tables

In this example, the dump files used are the same as those created in the previous example using the `ORACLE_LOADER` access driver. However, in this example, in addition to partitioning the data using `customer_number`, the data is subpartitioned using `postal_code`. For every partition, there is a subpartition where the `postal_code` is less than 50000 and another subpartition for all other values of `postal_code`. With three partitions, each containing two subpartitions, a total of six files is required. To create the files, use the SQL `CREATE TABLE AS SELECT` statement to select the correct rows for the partition and then write those rows into the file for the `ORACLE_DATAPUMP` driver.

The following statement creates a file with data for the first subpartition (`postal_code` less than 50000) of partition `p1` (`customer_number` less than 100).

```
SQL> create table customer_list_dp_p1_sp1_xt
  2  organization external
  3    (type oracle_datapump default directory def_dir1
location('p1_sp1.dmp'))
  4  as
  5    select customer_number, customer_name, postal_code
  6      from customer_list_xt partition (p1)
  7      where to_number(postal_code) < 50000;

Table created.

SQL>
```

This statement creates a file with data for the second subpartition (all other values for `postal_code`) of partition `p1` (`customer_number` less than 100).

```
SQL> create table customer_list_dp_p1_sp2_xt
  2  organization external
  3    (type oracle_datapump default directory def_dir1
location('p1_sp2.dmp'))
  4  as
  5    select customer_number, customer_name, postal_code
```

```
  6      from customer_list_xt partition (p1)
  7      where to_number(postal_code) >= 50000;

Table created.
```

The files for other partitions are created in a similar fashion, as follows:

```
SQL> create table customer_list_dp_p2_sp1_xt
  2  organization external
  3    (type oracle_datapump default directory def_dir1
location('p2_sp1.dmp'))
  4  as
  5    select customer_number, customer_name, postal_code
  6      from customer_list_xt partition (p2)
  7      where to_number(postal_code) < 50000;

Table created.

SQL>
SQL> create table customer_list_dp_p2_sp2_xt
  2  organization external
  3    (type oracle_datapump default directory def_dir1
location('p2_sp2.dmp'))
  4  as
  5    select customer_number, customer_name, postal_code
  6      from customer_list_xt partition (p2)
  7      where to_number(postal_code) >= 50000;

Table created.

SQL>
SQL> create table customer_list_dp_p3_sp1_xt
  2  organization external
  3    (type oracle_datapump default directory def_dir1
location('p3_sp1.dmp'))
  4  as
  5    select customer_number, customer_name, postal_code
  6      from customer_list_xt partition (p3)
  7      where to_number(postal_code) < 50000;

Table created.

SQL>
SQL> create table customer_list_dp_p3_sp2_xt
  2  organization external
  3    (type oracle_datapump default directory def_dir1
location('p3_sp2.dmp'))
  4  as
  5    select customer_number, customer_name, postal_code
  6      from customer_list_xt partition (p3)
  7      where to_number(postal_code) >= 50000;

Table created.

SQL>
```

You can select from each of these external tables to verify that it has the data you intended to write out. After you have run the SQL statement `CREATE TABLE AS SELECT`, you can drop these external tables.

To use a virtual column to partition the table, create the partitioned `ORACLE_DATAPUMP` table. Again, the table is partitioned on the `customer_number` column, and subpartitioned on the `postal_code` column. The `postal_code` column is a character field that contains numbers, but this example partitions it based on the numeric value, not a character string. In order to do this, create a virtual column, `postal_code_num`, whose value is the `postal_code` field converted to a `NUMBER` data type. The `SUBPARTITION` clause uses the virtual column to determine the subpartition for the row.

```
SQL> create table customer_list_dp_xt
  2  (customer_number    number,
  3   CUSTOMER_NAME      VARCHAR2(50),
  4   postal_code        CHAR(5),
  5   postal_code_NUM    as (to_number(postal_code)))
  6  organization external
  7    (type oracle_datapump default directory def_dir1)
  8  partition by range(customer_number)
  9  subpartition by range(postal_code_NUM)
 10  (
 11    partition p1 values less than (100)
 12     (subpartition p1_sp1 values less than (50000) location('p1_sp1.dmp'),
 13      subpartition p1_sp2 values less than (MAXVALUE)
location('p1_sp2.dmp')),
 14    partition p2 values less than (200)
 15     (subpartition p2_sp1 values less than (50000) location('p2_sp1.dmp'),
 16      subpartition p2_sp2 values less than (MAXVALUE)
location('p2_sp2.dmp')),
 17    partition p3 values less than (300)
 18     (subpartition p3_sp1 values less than (50000) location('p3_sp1.dmp'),
 19      subpartition p3_sp2 values less than (MAXVALUE)
location('p3_sp2.dmp'))
 20  );

Table created.

SQL>
```

If you select all rows, then the data returned is the same as was returned in the previous example using the `ORACLE_LOADER` access driver.

```
SQL> select customer_number, customer_name, postal_code
  2    from customer_list_dp_xt
  3    order by customer_number;

customer_number CUSTOMER_NAME                                      POSTA
--------------- -------------------------------------------------- -----
              1  AAAAA Plumbing                                     01372
             13  Andi's Doughnuts                                  54570
             28  Sparkly Laundry                                   78907
             51  DIY Supplies                                      61614
             87  Fast Frames                                       22201
             89  Friendly Pharmacy                                 89901
```

```
            105  Le Bistrot du Chat Noir                          94114
            121  Pleasant Pets                                    33893
            130  Bailey the Bookmonger                            99915
            210  The Electric Eel Diner                           07101
            222  Everyt'ing General Store                         80118
            231  Big Rocket Market                                01754

12 rows selected.

SQL>
```

The `WHERE` clause can limit the rows read to a subpartition. The following query should only read the first subpartition of the first partition.

```
SQL> select customer_number, customer_name, postal_code
  2    from customer_list_dp_xt
  3    where customer_number < 20 and postal_code_NUM < 39998
  4    order by customer_number;

customer_number CUSTOMER_NAME                                      POSTA
--------------- -------------------------------------------------- -----
              1  AAAAA Plumbing                                   01372

1 row selected.

SQL>
```

You could also specify a specific subpartition in the query, as follows:

```
SQL> select customer_number, customer_name, postal_code
  2    from customer_list_dp_xt subpartition (p2_sp2) order by
customer_number;

customer_number CUSTOMER_NAME                                      POSTA
--------------- -------------------------------------------------- -----
            105  Le Bistrot du Chat Noir                          94114
            130  Bailey the Bookmonger                            99915

2 rows selected.

SQL>
```

Related Topics

- Managing Hybrid Partitioned Tables

18.4 Using the ORACLE_BIGDATA Access Driver to Create Partitioned External Tables

The example in this section shows how to create a subpartitioned external table.

In the following example, we create a table called `SALES_EXTENDED_EXT` that has access to the table `sales_extended.parquet`, and the table `t.dat` that has access to the object store `tab_from_csv_oss`.

Example 18-4 Using the ORACLE_BIGDATA Access Driver to create

```
CREATE TABLE "SALES_EXTENDED_EXT"
        ("PROD_ID" NUMBER(10,0),
     "CUST_ID" NUMBER(10,0),
     "TIME_ID" VARCHAR2(4000 BYTE),
     "CHANNEL_ID" NUMBER(10,0),
     "PROMO_ID" NUMBER(10,0),
     "QUANTITY_SOLD" NUMBER(10,0),
     "AMOUNT_SOLD" NUMBER(10,2),
     "GENDER" VARCHAR2(4000 BYTE),
     "CITY" VARCHAR2(4000 BYTE),
     "STATE_PROVINCE" VARCHAR2(4000 BYTE),
     "INCOME_LEVEL" VARCHAR2(4000 BYTE)
        )
        ORGANIZATION EXTERNAL
         ( TYPE ORACLE_BIGDATA
           DEFAULT DIRECTORY "DATA_PUMP_DIR"
           ACCESS PARAMETERS
           ( com.oracle.bigdata.credential.name=oss
             com.oracle.bigdata.fileformat=PARQUET
       )
           LOCATION
            ( 'https://objectstorage.eu-frankfurt-1.oraclecloud.com/n/
adwc4pm/b/parquetfiles/o//sales_extended.parquet'
            )
         )
        REJECT LIMIT UNLIMITED
       PARALLEL ;

CREATE TABLE tab_from_csv_oss
     (
      c0 number,
      c1 varchar2(20)
     )
     ORGANIZATION external
     (
      TYPE oracle_bigdata
      DEFAULT DIRECTORY data_pump_dir
      ACCESS PARAMETERS
      (
       com.oracle.bigdata.fileformat=csv
       com.oracle.bigdata.credential.name=oci_swift
      )
      location
      (
       'https://objectstorage.us-sanjose-1.oraclecloud.com/n/axffbtla8jep/b/
misc/o/t.dat'
      )
     )REJECT LIMIT 1
     ;
```

18.5 Using the ORA_PARTITION_VALIDATION Function to Validate Partitioned External Tables

To confirm if a row in a partitioned external table is in the correct partition, use the `ORA_PARTITION_VALIDATION` function.

When you use partitioned external tables, Oracle Database cannot enforce data placement in a partition with the correct partition key definition. Using `ORA_PARTITION_VALIDATION` can help you to correct data placement errors.

Example 18-5 Using ORA_PARTITION_VALIDATION for Partition Testing

When you use the `ORA_PARTITION_VALIDATION` function, you can obtain a list of external table partition rows that are placed in the wrong partition. To demonstrate this feature, this example shows a partition created with the wrong department set followed by an example using the `ORA_PARTITION_VALIDATION` function to identify data in the incorrect partition:

```
create or replace directory def_dir1 as '/tmp';

REM create the exact same data in files locally
REM
set feedback 1
spool /tmp/xp1_15.txt
select '12#dept_12#xp1_15#' from dual;
spool off

spool /tmp/xp2_30.txt
select '29#dept_29#xp2_30#' from dual;
spool off

spool /tmp/xp2_wrong.txt
select '99#dept_99#xp2_wrong#' from dual;
spool off

drop table ept purge;
create table ept(deptno number,dname char(14),loc char(13))
organization external
( type oracle_loader
  default directory def_dir1
  access parameters(
    records delimited by newline
    fields terminated by '#')
)
reject limit unlimited
partition by range (deptno)
(
  partition ep1 values less than (10),
  partition ep2 values less than (20) location ('xp1_15.txt'),
  partition epwrong values less than (30) location ('xp2_wrong.txt')
)
;

select pt.*, ora_partition_validation(rowid) from pt;
```

18.6 Using SQL*Loader for External Tables with Partition Values in File Paths

To enhance management of large numbers of data files in object stores, you can use external table partitioning with folder names as part of the filepaths. External table columns also can return the filename of the source file for each row.

Starting in Oracle Database 23ai, External table partitioning where the partition key and partition value together (for example, `/state=CA`) or only the only the partition value (for example, `/state/CA/)` comprise a folder name in the file path. Also, an external table column can return the filename of the source file for each row.

External tables pointing to data in the object store can consist of a large number of files. These files can be organized across multiple directories, and even multiple directory trees. The partition values can be in the directory name or file name. For example, you can have files for different months or different states in separate directories. This can be a requirement for Hive-generated tables in the object store.

18.7 Loading LOBs with External Tables

External tables are particularly useful for loading large numbers of records from a single file, so that each record appears in its own row in the table.

- Overview of LOBs and External Tables
  Learn the benefits of using external tables with your database to read and write data, and to understand how to create them.
- Loading LOBs From External Tables with ORACLE_LOADER Access Driver
  You can load LOB columns from the primary data files, from `LOBfile`s, or from LOB Location Specifiers (LLS).
- Loading LOBs with ORACLE_DATAPUMP Access Driver
  Use this example to see how you can load LOBs `ORACLE_LOADER` access driver.

18.7.1 Overview of LOBs and External Tables

Learn the benefits of using external tables with your database to read and write data, and to understand how to create them.

External tables enable you to treat the contents of external files as if they are rows in a table in your Oracle Database. After you create an external table, you can then use SQL statements to read rows from the external table, and insert them into another table.

To perform these operations, Oracle Database uses one of the following access drivers:

- The `ORACLE_LOADER` access driver reads text files and other file formats, similar to SQL Loader.
- The `ORACLE_DATAPUMP` access driver creates binary files that store data returned by a query. It also returns rows from files in binary format.

When you create an external table, you specify column and data types for the external table. The access driver has a list of columns in the data file, and maps the contents of the field in the data file to the column with the same name in the external table. The access driver takes care of finding the fields in the data source, and converting these fields to the appropriate data type

for the corresponding column in the external table. After you create an external table, you can load the target table by using an `INSERT AS SELECT` statement.

One of the advantages of using external tables to load data over SQL Loader is that external tables can load data in parallel. The easiest way to do this is to specify the `PARALLEL` clause as part of `CREATE TABLE` for both the external table and the target table.

Example 18-6

This example creates a table, `CANDIDATE`, that can be loaded by an external table. When it is loaded, it then creates an external table, `CANDIDATE_XT`. Next, it executes an `INSERT` statement to load the table. The `INSERT` statement includes the `+APPEND` hint, which uses direct load to insert the rows into the table `CANDIDATES`. The `PARALLEL` parameter tells SQL that the tables can be accessed in parallel.

The `PARALLEL` parameter setting specifies that there can be four (`4`) parallel query processes reading from `CANDIDATE_XT`, and four parallel processes inserting into `CANDIDATE`. Note that LOBs that are stored as `BASICFILE` cannot be loaded in parallel. You can only load `SECUREFILE LOBS` in parallel. The variable *`additional-external-table-info`* indicates where additional external table information can be inserted.

```
CREATE TABLE CANDIDATES

  (candidate_id       NUMBER,

   first_name         VARCHAR2(15),

   last_name          VARCHAR2(20),

   resume             CLOB,

   picture            BLOB

  ) PARALLEL 4;


CREATE TABLE CANDIDATE_XT

  (candidate_id       NUMBER,

   first_name         VARCHAR2(15),

   last_name          VARCHAR2(20),

   resume             CLOB,

   picture            BLOB

  ) PARALLEL 4;

ORGANIZATION EXTERNAL additional-external-table-info PARALLEL 4;

INSERT /*+APPEND*/ INTO CANDIDATE SELECT * FROM CANDIDATE_XT;
```

File Locations for External Tables Created By Access Drivers

All files created or read by `ORACLE_LOADER` and `ORACLE_DATAPUMP` reside in directories pointed to by directory objects. Either the DBA or a user with the `CREATE DIRECTORY` privilege can create a directory object that maps a new to a path on the file system. These users can grant `READ`, `WRITE` or `EXECUTE` privileges on the created directory object to other users. A user granted `READ` privilege on a directory object can use external tables to read files from directory for the directory object. Similarly, a user with `WRITE` privilege on a directory object can use external tables to write files to the directory for the directory object.

Example 18-7 Creating Directory Object

The following example shows how to create a directory object and grant `READ` and `WRITE` access to user `HR`:

```
create directory HR_DIR as /usr/hr/files/exttab;

grant read, write on directory HR_DIR to HR;
```

> **Note:**
>
> When using external tables in an Oracle Real Application Clusters (Oracle RAC) environment, you must make sure that the directory pointed to by the directory object maps to a directory that is accessible from all nodes.

18.7.2 Loading LOBs From External Tables with ORACLE_LOADER Access Driver

You can load LOB columns from the primary data files, from `LOBfile`s, or from LOB Location Specifiers (LLS).

- Loading LOBs from Primary Data Files
  Use this example to see how you can use the `ORACLE_LOADER` access driver to load LOB columns from the primary data datatype files.
- Loading LOBs from LOBFILE Files
  Use this example to see how you can use the `ORACLE_LOADER` access driver to load LOB columns from `LOBFILE` data type files.
- Loading LOBs from LOB Location Specifiers
  Use this example to see how you can use the `ORACLE_LOADER` access driver to load LOBs from LOB location specifiers.

18.7.2.1 Loading LOBs from Primary Data Files

Use this example to see how you can use the `ORACLE_LOADER` access driver to load LOB columns from the primary data datatype files.

If the LOB data is in the primary data file, then it is just another field defined for the record format of the data file. It doesn't matter how you define the field in the access driver. You can use fixed positions to define the field, or you can use `CHAR`, `VARCHAR` or `VARCHARC`. Remember that the data types for `ORACLE_LOADER` are not the same as data types for SQL.

> **Note:**
>
> With Oracle Database 18c and later releases, symbolic links are not allowed in directory object path names used with `ORACLE_LOADER` access driver.

Example 18-8 Loading LOBs from primary data file

In this example, the `COMMENTS` field in each record is up to 10000 bytes. When you use `SELECT` to select the `COMMENT` column from table `INTERVIEW_XT`, the data for the `COMMENTS` field is converted into a character large object (CLOB), and presented to the Oracle SQL engine.

```
CREATE TABLE INTERVIEW_XT

(candidate_id         NUMBER,

 interviewer_id       NUMBER,

 comments             CLOB

)

ORGANIZATION EXTERNAL

(type ORACLE_LOADER

 default directory hr_dir

 access parameters

 (records delimited by newline

  fields terminated by '|'

  (candidate_id    CHAR(10),

   employee_id     CHAR(10),

   comments        CHAR(10000))

  )

  location ('interviews.dat')

);
```

18.7.2.2 Loading LOBs from LOBFILE Files

Use this example to see how you can use the `ORACLE_LOADER` access driver to load LOB columns from `LOBFILE` data type files.

Using LOB files can be preferable to reading LOBs from the from the primary data file, if your primary data file has any of the following characteristics:

- Record delimiters.

  The data for the LOB field cannot contain record deliminators in the data. In primary data files, record deliminators such as `NEWLINE` can be present in the data. But when the `ORACLE_LOADER` access driver accesses the next record, it looks for the next occurrence of the record delimiter. If the record delimiter is also part of the data, then it will not read the correct data for the LOB column.

- Field terminators.

  The data for the LOB column cannot contain field terminators. With primary data files, the data can contain field terminators, such as `|`. But just as with record deliminators, if field terminators are part of the data, then `ORACLE_LOADER` will not read the correct data for the LOB column.

- Record size that exceeds size limits.

  The data for a LOB column must fit within the size limits for a record. The `ORACLE_LOADER` access driver requires that a record not be any larger than the size of the read buffer. The default value is `1MB`, but you can change that with the READSIZE parameter.

- Binary data

  Reading binary data from the primary file requires extra care in creating the file. Unless you can guarantee that the record delimiter or field delimiter cannot occur inside the data for a `BLOB`, you need to use `VAR` record formats, and use `VARRAW` or `VARRAWC` data types for the binary fields. Files such as this typically must be generated programatically.

If your primary data file has any of these characteristics, then using `LOBFILE` data types to load LOB columns can be the better option for you to use.

> **Note:**
>
> With Oracle Database 18c and later releases, symbolic links are not allowed in directory object path names used with `ORACLE_LOADER` access driver.

Example 18-9 Loading LOBs from primary data file

For each LOB column in each record, the `ORACLE_LOADER` access driver requires a directory object, and the file name for the file that contains the contents of the `LOB`. Typically, all of the file for the `LOB` columns is in one directory, and each record in the data file has the file name in the directory. For example, suppose there is this object created for LOB files as user `HR`:

```
create directory HR_LOB_DIR as /usr/hr/files/exttab/lobfile;

grant read, write on directory HR_LOB_DIR to HR;
```

Suppose the data consists of these records:

```
cristina_resume.pdf

cristina.jpg

arvind_resume.pdf

arvind.jpg
```

The data file looks like this, using field terminators, comma delimitors, character strings, and binary data:

```
4378,Cristina,Garcia,cristina_resume.pdf,cristina.jpg

673289,Arvind,Gupta,arvind_resume.pdf,arvind.jpg
```

In this scenario, the external table LOB file appears as follows:

```
CREATE TABLE CANDIDATE_XT

  (candidate_id       NUMBER,

   first_name         VARCHAR2(15),

   last_name          VARCHAR2(20),

   resume             CLOB,

   picture            BLOB

  )

ORGANIZATION EXTERNAL

(type oracle_loader

 default directory hr_dir

 access parameters

 (fields terminated by ‘,’

  (candidate_id       char(10),

   first_name         char(15),

   last_name          char(20),

   resume_file        char(40),

   picture_file       char(40)

  )

  column transforms

  (

    resume from lobfile (constant 'HR_LOB_DIR': resume_file,

    picture from lobfile (constant 'HR_LOB_DIR': picture_file

  )
```

18.7.2.3 Loading LOBs from LOB Location Specifiers

Use this example to see how you can use the `ORACLE_LOADER` access driver to load LOBs from LOB location specifiers.

LOB Location Specifiers (LLS) are used when you have data for multiple LOBs in one file. When you use LLS to load a LOB column, the data in the primary data file contains the name of the file with the LOB data, the offset of the start of the LOB, and the number of bytes for the LOB.

> **Note:**
>
> With Oracle Database 18c and later releases, symbolic links are not allowed in directory object path names used with `ORACLE_LOADER` access driver.

Example 18-10 Loading Data Using LOB Location Specifiers

In the following example, suppose we have the directory `HR_LOB_DIR`, which contains resumes and pictures. In the directory, we have concatenated the resumes into one file, and the pictures into another file:

```
resumes.dat
pictures.dat
```

The data file appears as follows:

```
4378,Cristina,Garcia,resumes.dat.1.10928/,picture.dat.1.38679/

673289,Arvind,Gupta,resumes.dat.10929.8439,picture.dat.38680,45772/
```

In this scenario, the external table LOB file appears as follows:

```
CREATE TABLE CANDIDATE_XT

  (candidate_id       NUMBER,

   first_name         VARCHAR2(15),

   last_name          VARCHAR2(20),

   resume             CLOB,

   picture            BLOB

  )

ORGANIZATION EXTERNAL

(type oracle_loader

 default directory hr_dir
```

```
 access parameters

 (fields terminated by ‘,’

  (candidate_id       char(10),

   first_name         char(15),

   last_name          char(20),

   resume_file        lls directory ‘HR_LOB_DIR’,

   picture_file       lls directory ‘HR_LOB_DIR’

  )

 )

 location ('candidates.dat')

);
```

Related Topics

- LLS Clause
  If a field in a data file is a LOB location Specifier (`LLS`) field, then you can indicate this by using the LLS clause.

18.7.3 Loading LOBs with ORACLE_DATAPUMP Access Driver

Use this example to see how you can load LOBs `ORACLE_LOADER` access driver.

The `ORACLE_DATAPUMP` access driver enables you to unload data from a `SELECT` statement by using the command `CREATE TABLE AS SELECT`. This command creates a binary file that with data for all of the rows returned by the `SELECT` statement. After you have this file, you can create an `ORACLE_DATAPUMP` external table on the target database, and use the statement `INSERT INTO target_table SELECT * FROM external_table` to load the table.

> **Note:**
>
> With Oracle Database 18c and later releases, symbolic links are not allowed in directory object path names used with `ORACLE_DATAPUMP` access driver.

Example 18-11 Creating an External Table with CREATE TABLE AS SELECT

This example uses `CREATE TABLE AS SELECT` to unload data from a table in a database. It creates a file named `candidate.dmp` in the directory for `hr_dir`. It then creates an external table (it can be in another database or another schema in the same database), and then uses

INSERT to load the target table. Note that if the target table is in a different database then the file, then the file candidates.dmp must be copied to the directory for HR_DIR in that database.

```
CREATE TABLE CANDIDATE_XT

  (candidate_id       NUMBER,

   first_name         VARCHAR2(15),

   last_name          VARCHAR2(20),

   resume             CLOB,

   picture            BLOB

  )

ORGANIZATION EXTERNAL

(type oracle_datapump

 default directory hr_dir

 location ('candidates.dmp')

)

as select * from candidates;
```

Next, in another schema or another database, create the external table using the file created above. If executing this command in another database, then you must copy the file to the directory for HR_DIR in that database.

```
CREATE TABLE CANDIDATE_XT

  (candidate_id       NUMBER,

   first_name         VARCHAR2(15),

   last_name          VARCHAR2(20),

   resume             CLOB,

   picture            BLOB

  )

ORGANIZATION EXTERNAL

(type oracle_datapump

 default directory hr_dir

 location ('candidates.dmp')
```

```
);

INSERT INTO CANDIDATES SELECT * FROM CANDIDATE_XT;
```

18.8 Loading CSV Files From External Tables

This topic provides examples of how to load CSV files from external tables under various conditions.

Some of the examples build on previous examples.

Example 18-12 Loading Data From CSV Files With No Access Parameters

This example requires the following conditions:

- The order of the columns in the table must match the order of fields in the data file.
- The records in the data file must be terminated by newline.
- The field in the records in the data file must be separated by commas (if field values are enclosed in quotation marks, then the quotation marks are *not* removed from the field).
- There cannot be any newline characters in the middle of a field.

The data for the external table is as follows:

```
events_all.csv
Winter Games,10-JAN-2010,10,
Hockey Tournament,18-MAR-2009,3,
Baseball Expo,28-APR-2009,2,
International Football Meeting,2-MAY-2009,14,
Track and Field Finale,12-MAY-2010,3,
Mid-summer Swim Meet,5-JUL-2010,4,
Rugby Kickoff,28-SEP-2009,6,
```

The definition of the external table is as follows:

```
SQL> CREATE TABLE EVENTS_XT_1
  2  (EVENT       varchar2(30),
  3   START_DATE  date,
  4   LENGTH      number)
  5  ORGANIZATION EXTERNAL
  6  (default directory def_dir1 location ('events_all.csv'));

Table created.
```

The following shows a `SELECT` operation on the external table `EVENTS_XT_1`:

```
SQL> select START_DATE, EVENT, LENGTH
  2    from EVENTS_XT_1
  3    order by START_DATE;

START_DAT EVENT                              LENGTH
--------- ------------------------------ ----------
18-MAR-09 Hockey Tournament                       3
28-APR-09 Baseball Expo                           2
02-MAY-09 International Football Meeting         14
```

```
28-SEP-09 Rugby Kickoff                            6
10-JAN-10 Winter Games                            10
12-MAY-10 Track and Field Finale                   3
05-JUL-10 Mid-summer Swim Meet                     4

7 rows selected.

SQL>
```

Example 18-13 Default Date Mask For the Session Does Not Match the Format of Data Fields in the Data File

This example is the same as the previous example, except that the default date mask for the session does not match the format of date fields in the data file. In the example below, the session format for dates is `DD-Mon-YYYY` whereas the format of dates in the data file is `MM/DD/YYYY`. If the external table definition does not have a date mask, then the `ORACLE_LOADER` access driver uses the session date mask to attempt to convert the character data in the data file to a date data type. ou specify an access parameter for the date mask to use for all fields in the data file that are used to load date columns in the external table.

The following is the contents of the data file for the external table:

```
events_all_date_fmt.csv
Winter Games,1/10/2010,10
Hockey Tournament,3/18/2009,3
Baseball Expo,4/28/2009,2
International Football Meeting,5/2/2009,14
Track and Field Finale,5/12/2009,3
Mid-summer Swim Meet,7/5/2010,4
Rugby Kickoff,9/28/2009,6
```

The definition of the external table is as follows:

```
SQL> CREATE TABLE EVENTS_XT_2
  2  (EVENT       varchar2(30),
  3   START_DATE  date,
  4   LENGTH      number)
  5  ORGANIZATION EXTERNAL
  6  (default directory def_dir1
  7   access parameters (fields date_format date mask "mm/dd/yyyy")
  8   location ('events_all_date_fmt.csv'));

Table created.

SQL>
```

The following shows a `SELECT` operation on the external table `EVENTS_XT_2`:

```
SQL> select START_DATE, EVENT, LENGTH
  2    from EVENTS_XT_2
  3    order by START_DATE;

START_DAT EVENT                          LENGTH
--------- ------------------------------ ----------
18-MAR-09 Hockey Tournament                       3
```

```
28-APR-09 Baseball Expo                          2
02-MAY-09 International Football Meeting        14
12-MAY-09 Track and Field Finale                 3
28-SEP-09 Rugby Kickoff                          6
10-JAN-10 Winter Games                          10
05-JUL-10 Mid-summer Swim Meet                   4

7 rows selected.
```

Example 18-14 Data is Split Across Two Data Files

This example is that same as the first example in this section except for the following:

- The data is split across two data files.
- Each data file has a row containing the names of the fields.
- Some fields in the data file are enclosed by quotation marks.

The `FIELD NAMES ALL FILES` tells the access driver that the first row in each file contains a row with names of the fields in the file. The access driver matches the names of the fields to the names of the columns in the table. This means the order of the fields in the file can be different than the order of the columns in the table. If a field name in the first row is not enclosed in quotation marks, then the access driver uppercases the name before trying to find the matching column name in the table. If the field name is enclosed in quotation marks, then it does not change the case of the names before looking for a matching name.

Because the fields are enclosed in quotation marks, the access parameter requires the `CSV WITHOUT EMBEDDED RECORD TERMINATORS` clause. This clause states the following:

- Fields in the data file are separated by commas.
- If the fields are enclosed in double quotation marks, then the access driver removes them from the field value.
- There are no new lines embedded in the field values (this option allows the access driver to skip some checks that can slow the performance of `SELECT` operations on the external table).

The two data files are as follows:

```
events_1.csv

"EVENT","START DATE","LENGTH",
"Winter Games", "10-JAN-2010", "10"
"Hockey Tournament", "18-MAR-2009", "3"
"Baseball Expo", "28-APR-2009", "2"
"International Football Meeting", "2-MAY-2009", "14"


events_2.csv

Event,Start date,Length,
Track and Field Finale, 12-MAY-2009, 3
Mid-summer Swim Meet, 5-JUL-2010, 4
Rugby Kickoff, 28-SEP-2009, 6
```

The external table definition is as follows:

```
SQL> CREATE TABLE EVENTS_XT_3
  2  ("START DATE"  date,
  3   EVENT         varchar2(30),
  4   LENGTH        number)
  5  ORGANIZATION EXTERNAL
  6  (default directory def_dir1
  7   access parameters (records field names all files
  8                      fields csv without embedded record terminators)
  9   location ('events_1.csv', 'events_2.csv'));

Table created.
```

The following shows the result of a `SELECT` operation on the `EVENTS_XT_3` external table:

```
SQL> select "START DATE", EVENT, LENGTH
  2    from EVENTS_XT_3
  3    order by "START DATE";

START DAT EVENT                              LENGTH
--------- ------------------------------ ----------
18-MAR-09 Hockey Tournament                       3
28-APR-09 Baseball Expo                           2
02-MAY-09 International Football Meeting         14
12-MAY-09 Track and Field Finale                  3
28-SEP-09 Rugby Kickoff                           6
10-JAN-10 Winter Games                           10
05-JUL-10 Mid-summer Swim Meet                    4

7 rows selected.
```

Example 18-15 Data Is Split Across Two Files and Only the First File Has a Row of Field Names

This example is the same as example 3 except that only the 1st file has a row of field names. The first row of the second file has real data. The RECORDS clause changes to "field names first file".

The two data files are as follows:

```
events_1.csv (same as for example 3)

"EVENT","START DATE","LENGTH",
"Winter Games", "10-JAN-2010", "10"
"Hockey Tournament", "18-MAR-2009", "3"
"Baseball Expo", "28-APR-2009", "2"
"International Football Meeting", "2-MAY-2009", "14"


events_2_no_header_row.csv

Track and Field Finale, 12-MAY-2009, 3
```

```
Mid-summer Swim Meet, 5-JUL-2010, 4
Rugby Kickoff, 28-SEP-2009, 6
```

The external table definition is as follows:

```
SQL> CREATE TABLE EVENTS_XT_4
  2  ("START DATE"  date,
  3   EVENT         varchar2(30),
  4   LENGTH        number)
  5  ORGANIZATION EXTERNAL
  6  (default directory def_dir1
  7   access parameters (records field names first file
  8                      fields csv without embedded record terminators)
  9   location ('events_1.csv', 'events_2_no_header_row.csv'));

Table created.
```

The following shows a `SELECT` operation on the `EVENTS_XT_4` external table:

```
SQL> select "START DATE", EVENT, LENGTH
  2    from EVENTS_XT_4
  3    order by "START DATE";

START DAT EVENT                              LENGTH
--------- ------------------------------ ----------
18-MAR-09 Hockey Tournament                       3
28-APR-09 Baseball Expo                           2
02-MAY-09 International Football Meeting         14
12-MAY-09 Track and Field Finale                  3
28-SEP-09 Rugby Kickoff                           6
10-JAN-10 Winter Games                           10
05-JUL-10 Mid-summer Swim Meet                    4

7 rows selected.
```

Example 18-16 The Order of the Fields in the File Match the Order of the Columns in the Table

This example has the following conditions:

- The order of the fields in the file match the order of the columns in the table.
- Fields are separated by newlines and are optionally enclosed in double quotation marks.
- There are fields that have embedded newlines in their value and those fields are enclosed in double quotation marks.

The contents of the data files are as follows:

```
event_contacts_1.csv

Winter Games, 10-JAN-2010, Ana Davis,
Hockey Tournament, 18-MAR-2009, "Daniel Dube
Michel Gagnon",
Baseball Expo, 28-APR-2009, "Robert Brown"
Internation Football Meeting, 2-MAY-2009,"Pete Perez
```

```
Randall Barnes
Melissa Gray",


event_contacts_2.csv

Track and Field Finale, 12-MAY-2009, John Taylor,
Mid-summer Swim Meet, 5-JUL-2010, "Louise Stewart
Cindy Sanders"
Rugby Kickoff, 28-SEP-2009, "Don Nguyen
Ray Lavoie"
```

The table definition is as follows. The `CSV WITH EMBEDDED RECORD TERMINATORS` clause tells the access driver how to handle fields enclosed by double quotation marks that also have embedded new lines.

```
SQL> CREATE TABLE EVENTS_CONTACTS_1
  2  (EVENT       varchar2(30),
  3   START_DATE  date,
  4   CONTACT     varchar2(120))
  5  ORGANIZATION EXTERNAL
  6  (default directory def_dir1
  7   access parameters (fields CSV with embedded record terminators)
  8   location ('event_contacts_1.csv', 'event_contacts_2.csv'));

Table created.
```

The following shows the result of a `SELECT` operation on the `EVENT_CONTACTS_1` external table:

```
SQL> column contact format a30
SQL> select START_DATE, EVENT, CONTACT
  2    from EVENTS_CONTACTS_1
  3    order by START_DATE;

START_DAT EVENT                          CONTACT
--------- ------------------------------ ------------------------------
18-MAR-09 Hockey Tournament              Daniel Dube
                                         Michel Gagnon

28-APR-09 Baseball Expo                  Robert Brown
02-MAY-09 Internation Football Meeting   Pete Perez
                                         Randall Barnes
                                         Melissa Gray

12-MAY-09 Track and Field Finale         John Taylor
28-SEP-09 Rugby Kickoff                  Don Nguyen
                                         Ray Lavoie

10-JAN-10 Winter Games                   Ana Davis
05-JUL-10 Mid-summer Swim Meet           Louise Stewart
                                         Cindy Sanders

7 rows selected.
```

Example 18-17 Not All Fields in the Data File Use Default Settings for the Access Parameters

This example shows what to do when most field in the data file use default settings for the access parameters but a few do not. Instead of listing the setting for all fields, this example shows how you can set attributes for just the fields that are different from the default. The differences are as follows:

- there are two date fields, one of which uses the session format, but `registration_deadline` uses a different format
- `registration_deadline` also uses a value of `NONE` to indicate a null value.

The content of the data file is as follows:

```
events_reg.csv

Winter Games,10-JAN-2010,10,12/1/2009,
Hockey Tournament,18-MAR-2009,3,3/11/2009,
Baseball Expo,28-APR-2009,2,NONE
International Football Meeting,2-MAY-2009,14,3/1/2009
Track and Field Finale,12-MAY-2010,3,5/10/010
Mid-summer Swim Meet,5-JUL-2010,4,6/20/2010
Rugby Kickoff,28-SEP-2009,6,NONE
```

The table definition is as follows. The `ALL FIELDS OVERRIDE` clause allows you to specify information for that field while using defaults for the remaining fields. The remaining fields have a data type of `CHAR(255)` and the field data is terminated by a comma with a trimming option of `LDRTRIM`.

```
SQL> CREATE TABLE EVENT_REGISTRATION_1
  2  (EVENT                 varchar2(30),
  3   START_DATE            date,
  4   LENGTH                number,
  5   REGISTRATION_DEADLINE date)
  6  ORGANIZATION EXTERNAL
  7  (default directory def_dir1
  8   access parameters
  9  (fields all fields override
 10     (REGISTRATION_DEADLINE CHAR (10) DATE_FORMAT DATE MASK "mm/dd/yyyy"
 11                           NULLIF REGISTRATION_DEADLINE = 'NONE'))
 12   location ('events_reg.csv'));

Table created.
```

The following shows the result of a `SELECT` operation on the `EVENT_REGISTRATION_1` external table:

```
SQL> select START_DATE, EVENT, LENGTH, REGISTRATION_DEADLINE
  2    from EVENT_REGISTRATION_1
  3    order by START_DATE;

START_DAT EVENT                              LENGTH REGISTRAT
--------- ------------------------------ ---------- ---------
18-MAR-09 Hockey Tournament                       3 11-MAR-09
28-APR-09 Baseball Expo                           2
```

```
02-MAY-09 International Football Meeting         14 01-MAR-09
28-SEP-09 Rugby Kickoff                           6
10-JAN-10 Winter Games                           10 01-DEC-09
12-MAY-10 Track and Field Finale                  3 10-MAY-10
05-JUL-10 Mid-summer Swim Meet                    4 20-JUN-10

7 rows selected.
```

18.9 Using Vector Data Types in External Tables

See examples of how you can use Vector data types in external tables for vector similarity searches.

- Understanding Vector Data Types in External Tables
  Especially with large data sets, external tables can be an efficient way to store data outside of the database store.
- Creating External Tables Using Oracle Loader Driver
  In this example, you can see how to create an external table vector store using the `ORACLE_LOADER` driver.
- Creating External Tables Using ORACLE_DATAPUMP Driver
  To create an external table vector store with `ORACLE_DATAPUMP`, you use SQL*Loader.
- Querying an Inline External Table
  In this example, vectors in an external table of type `ORACLE_BIGDATA` are queried as part of a vector search.
- Performing a Semantic Similarity Search Using External Table
  See a SQL example plan that illustrates how you can use external tables as the data set for semantic similarity searches

18.9.1 Understanding Vector Data Types in External Tables

Especially with large data sets, external tables can be an efficient way to store data outside of the database store.

Starting with Oracle Database 23ai (Release Update 23.7), you can use vector data in external tables. Vector embeddings often require very large data sets, especially when using unstructured data. With external tables, you can store unstructured data outside of the database, and make it available for processing using external tables. Using external tables can be an efficient way to store data outside of the database store, using the external tables for similarity searches, and for creating and maintaining vector indexes.

External data sources can be in the following locations:

- Local directories
- Oracle Cloud Infrastructure (OCI) Object Stores
- Microsoft Azure Blob Storage
- Amazon Web Service (AWS) S3 Cloud Object Storage
- GitHub storage

You can load, retrieve, and unload data from external tables using the following drivers:

- The `ORACLE_LOADER` access driver

- The Oracle Data Pump (`ORACLE_DATAPUMP`) access driver
- The Oracle Big Data (`ORACLE_BIGDATA`) access driver

18.9.2 Creating External Tables Using Oracle Loader Driver

In this example, you can see how to create an external table vector store using the `ORACLE_LOADER` driver.

In this example, the external table name is `ext_table_3`. The table is created using the `ORACLE_LOADER` driver, where records are delimited by new lines, and the fields are terminated by `:`.

```
create table ext_table_3
(
    v1 vector,
    v2 vector
)
organization external
(
    type oracle_loader
    default directory dir1
    access parameters
    (
        records delimited by newline
        fields terminated by ':'
        missing field values are null
    )
    location ('tkexvect3.csv')
)
reject limit unlimited;
```

18.9.3 Creating External Tables Using ORACLE_DATAPUMP Driver

To create an external table vector store with `ORACLE_DATAPUMP`, you use SQL*Loader.

Complete the following procedure:

1. Create an external table using SQL*Loader

   Example:

```
CREATE TABLE vector_ext_tab (
        country_code         VARCHAR2(5),
        country_name         VARCHAR2(50),
        country_language     VARCHAR2(50),
        country_vector       VECTOR(*,*)
    )
    ORGANIZATION EXTERNAL (
       TYPE ORACLE_LOADER
       DEFAULT DIRECTORY DIR1
       ACCESS PARAMETERS (
          RECORDS DELIMITED BY NEWLINE
          FIELDS TERMINATED BY ":"
          MISSING FIELD VALUES ARE NULL
          (
```

```
                country_code         CHAR(5),
                country_name         CHAR(50),
                country_language     CHAR(50),
                country_vector       CHAR(10000)
             )
          )
          LOCATION ('tklgvectorcountries.dat')
       )
       PARALLEL 5
       REJECT LIMIT UNLIMITED;
```

2. Generate a dump file (`dmp`).

   Example:

```
create table export_table
organization external
(
    type oracle_datapump
    default directory dir1
    location ('exp1.dmp')
)
as select * from vector_ext_tab;
```

18.9.4 Querying an Inline External Table

In this example, vectors in an external table of type `ORACLE_BIGDATA` are queried as part of a vector search.

```
select * from external (
    (
      COL1 vector,
      COL2 vector,
      COL3 vector,
      COL4 vector
    )
    TYPE ORACLE_BIGDATA
    DEFAULT DIRECTORY DEF_DIR1
    ACCESS PARAMETERS
    (
      com.oracle.bigdata.credential.name\=OCI_CRED
      com.oracle.bigdata.credential.schema\=PDB_ADMIN
      com.oracle.bigdata.fileformat=parquet
      com.oracle.bigdata.debug=true
    )
    location ( 'https://swiftobjectstorage.us-phoenix-1.oraclecloud.com/v1/
myvdataproject/BIGDATA_PARQUET/vector_data/basic_vector_data.parquet' )
    REJECT LIMIT UNLIMITED
  ) tkexobd_bd_vector_inline;
```

18.9.5 Performing a Semantic Similarity Search Using External Table

See a SQL example plan that illustrates how you can use external tables as the data set for semantic similarity searches

The following is an example of an explain plan for `select id`, embedding from `ext_table_3`, and using `order by vector_distance('[1,1]', embedding, cosine)` to return approximately only the first three rows of data with a target accuracy of 90 percent:

```
SQL> select * from table(dbms_xplan.display('plan_table', null, 'advanced
predicate'));

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--
Plan hash value: 1784440045

--------------------------------------------------------------------------------
--
---------------------

| Id  | Operation                     | Name        | Rows  | Bytes |TempSpc|
Co
st (%CPU)| Time     |

--------------------------------------------------------------------------------
--
---------------------


PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--
|   0 | SELECT STATEMENT              |             |     3 | 48945 |
|  1
466K  (2)| 00:00:58 |

|*  1 |  COUNT STOPKEY                |             |       |       |       |
         |          |

|   2 |   VIEW                        |             |   102K|  1588M|
|  1
466K  (2)| 00:00:58 |

|*  3 |    SORT ORDER BY STOPKEY      |             |   102K|  1589M|
798M|  1
466K  (2)| 00:00:58 |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--

|   4 |     EXTERNAL TABLE ACCESS FULL| EXT_TABLE_3 |   102K|  1589M|       |
362   (7)| 00:00:01 |
```

```
-------------------------------------------------------------------------------
--
---------------------


Query Block Name / Object Alias (identified by operation id):
-------------------------------------------------------------


PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------
--
   1 - SEL$2
   2 - SEL$1 / "from$_subquery$_002"@"SEL$2"
   3 - SEL$1
   4 - SEL$1 / "EXT_TABLE_3"@"SEL$1"

Outline Data
-------------

  /*+
      BEGIN_OUTLINE_DATA
      FULL(@"SEL$1" "EXT_TABLE_3"@"SEL$1")

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------
--
      NO_ACCESS(@"SEL$2" "from$_subquery$_002"@"SEL$2")
      OUTLINE_LEAF(@"SEL$2")
      OUTLINE_LEAF(@"SEL$1")
      ALL_ROWS
      OPT_PARAM('_fix_control' '6670551:0')
      OPT_PARAM('_optimizer_cost_model' 'fixed')
      DB_VERSION('26.1.0')
      OPTIMIZER_FEATURES_ENABLE('26.1.0')
      IGNORE_OPTIM_EMBEDDED_HINTS
      END_OUTLINE_DATA
  */

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------
--

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter(ROWNUM<=3)
   3 - filter(ROWNUM<=3)

Column Projection Information (identified by operation id):
-----------------------------------------------------------

   1 - "from$_subquery$_002"."ID"[NUMBER,22],
"from$_subquery$_002"."EMBEDDING"[

PLAN_TABLE_OUTPUT
```

```
--------------------------------------------------------------------------------
--
VECTOR,32600]

   2 - "from$_subquery$_002"."ID"[NUMBER,22],
"from$_subquery$_002"."EMBEDDING"[
VECTOR,32600]

   3 - (#keys=1) VECTOR_DISTANCE(VECTOR('[1,1]', *, *, * /*+
USEBLOBPCW_QVCGMD
*/ ),

       "EMBEDDING" /*+ LOB_BY_VALUE */ , COSINE)[BINARY_DOUBLE,8],
"ID"[NUMBER,2
2], "EMBEDDING" /*+


PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--
       LOB_BY_VALUE */ [VECTOR,32600]
   4 - "ID"[NUMBER,22], "EMBEDDING" /*+ LOB_BY_VALUE */ [VECTOR,32600],
       VECTOR_DISTANCE(VECTOR('[1,1]', *, *, * /*+  USEBLOBPCW_QVCGMD */ ),
"EMB
EDDING" /*+

       LOB_BY_VALUE */ , COSINE)[BINARY_DOUBLE,8]

Query Block Registry:
---------------------

  SEL$1 (PARSER) [FINAL]

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--
  SEL$2 (PARSER) [FINAL]
```