s of Oracle9i, you can use the external table feature to access external files as if they are tables inside the database. When you create an external table, you define its structure and location within Oracle. When you query the table, Oracle reads the external table and returns the results just as if the data had been stored within the database. But since the data is outside the database, you do not have to be concerned about the process for loading it into the database—a potentially significant benefit for data warehouses and large databases.

External tables have limits—you cannot insert, update, or delete their rows from within Oracle, and you cannot index them. Since they are part of the database application, you will have to account for them as part of your backup and recovery processes. Despite these complications, external tables can be a powerful addition to your database architecture plans.

Accessing the External Data

To access external files from within Oracle, you must first use the `create directory` command to define a directory object pointing to the external file location. Users who will access the external files must have the READ privilege on the directory.

**NOTE**
Before you start, verify that the external directory exists, and that the user who will be issuing the `create directory` command has the CREATE ANY DIRECTORY system privilege.

The following example creates a directory named `BOOK_DIR` and grants READ and WRITE access to the Practice schema:

```
create directory BOOK_DIR as 'e:\oracle\external';
grant read on directory BOOK_DIR to practice;
grant write on directory BOOK_DIR to practice;
```

The Practice user can now read files in the `e:\oracle\external` directory as if they were inside the database. Because Practice has also been granted WRITE privilege on that directory, the Practice user can create, log, discard, and bad files within the directory—just as if that user were executing the SQL*Loader utility (see Chapter 21).

The following listing generates two files for sample data, one from `BOOKSHELF` and one from `BOOKSHELF_AUTHOR`. Note that the `spool` command cannot use the directory name created via `create directory`; you need to specify the full operating system directory name.

```
connect practice/practice

set pagesize 0
newpage 0
feedback off

select Title||'-'||Publisher||'-'||CategoryName||'-'||Rating||'-'
from BOOKSHELF
order by Title

spool e:\oracle\external\bookshelf_dump.lst
```
select Title || '-' || AuthorName || '-'
from BOOKSHELF_AUTHOR
order by Title
spool e:\oracle\external\book_auth_dump.lst
/
spool off

In addition to the data, the output files will contain a single line at the top with a "/' and a final line that reads "SQL> spool off". To simplify data management, you should manually edit the file at the operating system level to delete these extra lines.

If another user is to access the data in the bookshelf_dump.lst and book_auth_dump.lst files, you must grant that user READ privilege on the BOOK_DIR directory:

grant read on directory BOOK_DIR to another_user;
and the files themselves must be readable by the Oracle user at the operating system level.

Creating an External Table

Now that the external data is available and accessible, you can create a table structure that accesses it. To do so, you need to use the organization external clause of the create table command. Within that clause, you can specify the data structure much as you would for a SQL*Loader control file. The following listing shows the creation of the BOOKSHELF_EXT table, based on the data in the bookshelf.lst spool file created in the prior section:

set feedback on heading on newpage 1 pagesize 60

create table BOOKSHELF_EXT
(Title VARCHAR2(100),
Publisher VARCHAR2(20),
CategoryName VARCHAR2(20),
Rating VARCHAR2(2)
) organization external
(type ORACLE_LOADER
default directory BOOK_DIR
access parameters (records delimited by newline
fields terminated by "-

 {Title CHAR(100),
 Publisher CHAR(20),
 CategoryName CHAR(20),
 Rating CHAR(2)
})
location ('bookshelf_dump.lst')
);
Oracle will respond with:

```
table created.
```

although no data will have been created inside the Oracle database.

Similarly, you can create a table based on the book_auth_dump.lst spool file:

```
create table BOOKSHELF_AUTHOR_EXT
  (Title    VARCHAR2(100),
   AuthorName    VARCHAR2(50)
  )
organization external
  (type ORACLE_LOADER
   default directory BOOK_DIR
   access parameters (records delimited by newline
                     fields terminated by " ",
                       (Title   CHAR(100),
                        AuthorName   CHAR(50)
                      )))
location ('book_auth_dump.lst')
```

**NOTE**

Oracle will perform only cursory validation when the external table is created. You will not see most errors until you attempt to query the table. The syntax for the access parameters is very specific, and minor errors in the access definition may prevent all of the rows from being accessed.

You can verify the contents of the external tables by querying from them and comparing them to the source tables, as shown in the following listing:

```
select Title from BOOKSHELF
  where CategoryName = 'CHILDRENPIC';
```

```
+------------------------------+
<table>
<thead>
<tr>
<th>TITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOOD DOG, CARL</td>
</tr>
<tr>
<td>POLAR EXPRESS</td>
</tr>
<tr>
<td>RUNAWAY BUNNY</td>
</tr>
</tbody>
</table>
+------------------------------+
3 rows selected.
```

```
select Title from BOOKSHELF_EXT
  where CategoryName = 'CHILDRENPIC';
```

```
+------------------------------+
<table>
<thead>
<tr>
<th>TITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOOD DOG, CARL</td>
</tr>
<tr>
<td>POLAR EXPRESS</td>
</tr>
<tr>
<td>RUNAWAY BUNNY</td>
</tr>
</tbody>
</table>
+------------------------------+
3 rows selected.
```
GOOD DOG, CARL
POLAR EXPRESS
RUNAWAY BUNNY

3 rows selected.

```
select COUNT(*) from BOOKSHELF_AUTHOR;
  COUNT(*)
----------
    37
```

```
select COUNT(*) from BOOKSHELF_AUTHOR_EXT;
  COUNT(*)
----------
    37
```

You can join the “internal” table BOOKSHELF_AUTHOR to its external counterpart, BOOKSHELF_AUTHOR_EXT, to verify there are no rows missing or added:

```
select * from BOOKSHELF_AUTHOR BA
where not exists
  (select 'x' from BOOKSHELF_AUTHOR_EXT BAE
    where BA.Title = BAE.Title
    and BA.AuthorName = BAE.AuthorName);
```

no rows selected

The BOOKSHELF_AUTHOR_EXT table points to the book_auth_dump.lst file. If you alter the data in the file, the data in BOOKSHELF_AUTHOR_EXT will change. As illustrated here, you can query external tables the same way you query standard tables—in joins, as part of views, and so on. You can perform functions on the external table columns during queries just as you would for standard tables.

You can query the USER_EXTERNAL_TABLES data dictionary view for information about your external tables, including the default directory and access definitions:

```
desc USER_EXTERNAL_TABLES
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_NAME</td>
<td>NOT NULL</td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>TYPE_OWNER</td>
<td>NOT NULL</td>
<td>CHAR(3)</td>
</tr>
</tbody>
</table>
For example, the BOOKSHELF_AUTHOR_EXT table uses BOOK_DIR as its default directory as shown in the following listing:

```sql
select * from USER_EXTERNAL_TABLES
where Table_Name = 'BOOKSHELF_AUTHOR_EXT';
```

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>TYP</th>
<th>TYPE_NAME</th>
<th>DEF</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_DIRECTORY_NAME</td>
<td>REJECT_LIMIT</td>
<td></td>
<td>SYS</td>
</tr>
<tr>
<td>ACCESS_PARAMETERS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BOOKSHELF_AUTHOR_EXT</td>
<td></td>
<td>SYS Oracle.Loader</td>
<td>SYS</td>
</tr>
<tr>
<td>BOOK_DIR</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

records delimited by newline
fields terminated by ";"
  (Title CHAR(100),
   AuthorName CHAR(50)
)

USER_EXTERNAL_TABLES does not show the name of the external file (or files) the table references. To see that information, query USER_EXTERNAL_LOCATIONS:

```sql
select * from USER_EXTERNAL_LOCATIONS;
```

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCATION</td>
</tr>
<tr>
<td>DIR.Directory_Name</td>
</tr>
<tr>
<td>BOOKSHELF_AUTHOR_EXT</td>
</tr>
<tr>
<td>book_auth_dump.lst</td>
</tr>
<tr>
<td>SYS BOOK_DIR</td>
</tr>
<tr>
<td>BOOKSHELF_EXT</td>
</tr>
<tr>
<td>bookshelf_dump.lst</td>
</tr>
<tr>
<td>SYS BOOK_DIR</td>
</tr>
</tbody>
</table>

**External Table Creation Options**

Within the `organization external` clause, there are four main subclauses: `type`, `default directory`, `access parameters`, and `location`. When you create an external table, you can use these data to customize the way Oracle views the external data.

**Type and Default Directory**

The syntax for the `type` component is

```sql
(type access_driver_type) external_data_properties
[reject limit { integer | unlimited }]
```
For external tables, the access driver is the API used to transform the external data. Use type
ORACLE_LOADER for your external tables—the only type available at this time—as shown in the
examples earlier in this chapter.

NOTE
Because the access driver is part of the Oracle software, only files
accessible by the database can be accessed as external tables. Files
the Oracle user cannot access cannot be used as external tables.

Following the type declaration, you can set a “reject limit” value. By default, no rows can
be rejected—any problem with any row will cause the select statement to return an error. Let’s
generate another copy of the BOOKSHELF data to a separate file, and this time leave in the extra
lines SQL*Plus inserts during the spool operation:

```sql
set pagesize 0新股价 0 feedback off
select Title||'-'||Publisher||'-'||CategoryName||'-'||Rating||'
from BOOKSHELF
order by Title
spool e:\oracle\external\bookshelf_dump_2.lst
/
spool off
```

Now create a new table that references this spool file, telling Oracle to skip the first record
(skip 1) and to allow one other error (reject limit 1). That will account for the “/” in the first line
and the “SQL> spool off” in the last line:

```sql
set feedback on heading on新股价 1 pagesize 60

create table BOOKSHELF_EXT_2
 (Title VARCHAR2(100),
 Publisher VARCHAR2(20),
 CategoryName VARCHAR2(20),
 Rating VARCHAR2(2)
 )
organization external
 {type ORACLE_LOADER
 default directory BOOK_DIR
 access parameters (records delimited by newline
skip 1
 fields terminated by
"-"
 (Title CHAR(100),
 Publisher CHAR(20),
 CategoryName CHAR(20),
 Rating CHAR(2)
 )
location ('bookshelf_dump_2.lst')

```
reject limit 1

You can now verify the number of rows in the table:

```
set feedback on heading on newpage 1 pagesize 60
select COUNT(*) from BOOKSHELF_EXT_2;
```

The default directory clause specifies the directory object to be used for all datafiles that do not specify another directory. If you use multiple external files located in multiple directories, you can name one of them as the default directory and specify the others by directory name in the location clause. You must use directory object names (such as BOOK_DIR) in the location clause, not the full directory path name.

**Access Parameters**

The access parameters clause tells Oracle how to map the rows in the file to rows in the table. Its syntax is shown in the following illustration:
Within the **access parameters** clause, you first tell Oracle how to create a record—whether its length is fixed or variable, and how rows are delimited. In the case of the BOOKSHELLE EXT example, the records are delimited by newlines. If there were multiple rows on a single line, you could use a character string as a separator between rows. Since the external data may come from a non-Oracle database, Oracle supports multiple character sets and string sizes.

As with SQL*Loader, you can specify a **when** clause to limit which rows are selected. In the following listing, the BOOKSHELFE EXT 3 table is created, with a **when** clause (shown in bold) to limit it to only books in the CHILDRENPIC category.

```sql
create table BOOKSHELFE EXT 3
  (Title VARCHAR2(100),
   Publisher VARCHAR2(20),
   CategoryName VARCHAR2(20),
   Rating VARCHAR2(2)
  )
organization external
;type ORACLE_LOADER
default directory BOOK_DIR
access parameters (records delimited by newline
  load when CategoryName = 'CHILDRENPIC'
    skip 1
    fields terminated by '='
    (Title CHAR(100),
     Publisher CHAR(20),
     CategoryName CHAR(20),
     Rating CHAR(2)
    )
)
location ('bookshelf_dump_2.lst')
)
reject limit 1
;
```

You can see the result here:

```sql
select SUBSTR(Title, 1, 30), CategoryName
from BOOKSHELFE EXT 3;
```

<table>
<thead>
<tr>
<th>SUBSTR(TITLE,1,30)</th>
<th>CATEGORYNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOOD DOG, CARL</td>
<td>CHILDRENPIC</td>
</tr>
<tr>
<td>POLAR EXPRESS</td>
<td>CHILDRENPIC</td>
</tr>
<tr>
<td>RUNAWAY BUNNY</td>
<td>CHILDRENPIC</td>
</tr>
</tbody>
</table>

3 rows selected.

BOOKSHELFE EXT 3 accesses the same file as BOOKSHELFE EXT 2, but it only shows the records for the CHILDRENPIC category due to its **load when** clause.

As with SQL*Loader, you can create a log file, a bad file, and a discard file. Rows that fail the **load when** condition will be written to the discard file. Rows that fail the **access parameters**
conditions will be written to the bad file, and the load details will be written to the log file. For all three types of files, you can specify a directory object along with the filename so that you can write the output to a directory other than your input datafile directory. You can specify `nodiscardfile`, `nobadfile`, and `nologfile` to prevent these files from being created. Use directory object names (such as `BOOK_DIR` in this chapter’s examples) when specifying locations for discard files, bad files, and log files. If you don’t specify locations for log files, bad files, and discard files, Oracle creates them in the default directory with system-generated names.

Within the `access parameters` clause, you also specify the field definitions and delimiters, such as

```sql
fields terminated by ";"
  | Title CHAR(100),
  | Publisher CHAR(20),
  | CategoryName CHAR(20),
  | Rating CHAR(2)
```

You can use the `missing field values are null` clause to set values for `NULL` column values, but you must exercise caution when using this option. For example, the `AUTHOR` table has `NULL` values in its `Comments` column. The external table creation for `AUTHOR_EXT` is shown in the following listing:

```sql
set pagesize 0 newpage 0 feedback off
select AuthorName||"-"||Comments||"-"
  from AUTHOR
order by AuthorName
spool e:\oracle\external\author_dump.lst
/
spool off
set feedback on heading on newpage 1 pagesize 60
create table AUTHOR_EXT
  (AuthorName VARCHAR2(50),
   Comments VARCHAR2(100)
  )
organization external
[type ORACLE_LOADER
default directory BOOK_DIR
access parameters (records delimited by newline
  skip 1
  fields terminated by "-"
  missing field values are null
  (AuthorName CHAR(50),
   Comments CHAR(100)
  )
)
location ('author_dump.lst')
)
reject limit 1
;
```
But this is not correct—if you select the AuthorName values from AUTHOR_EXT, you will see that the values include:

```sql
select AuthorName from AUTHOR_EXT
where AuthorName like 'S%';
```

SOREN KIERKEGAARD
STEPHEN AMBROSE
STEPHEN JAY GOULD

SQL> spool off

4 rows selected.

Because of the **missing field values are null** clause, the "SQL> spool off" line at the end of the listing was read as an AuthorName value, with a **NULL** Comments value. This highlights the problem with coding exceptions into your loader definitions—you need to make sure you fully understand the source data and the way the loader will treat it. In most cases, your data integrity will be better served by forcing rows to fail (into bad files or discard files) and evaluating the failed rows apart from your general loads.

See the SQL*Loader entry in the Alphabetical Reference for the full syntax available for the **access parameters** clause.

**Location**

In the **location** clause, you specify the datafiles to use as the source data for the table. You can name multiple files in the **location** clause if they all exist in directory objects the user has **READ privilege** on. The following example combines two separate BOOKSHELF spool files to illustrate the ability to combine multiple files into a single external table.

```sql
create table BOOKSHELF_EXT_4
(Title VARCHAR2(100),
Publisher VARCHAR2(20),
CategoryName VARCHAR2(20),
Rating VARCHAR2(2)
)
organization external
(type ORACLE_LOADER
default directory BOOK_DIR
access parameters (records delimited by newline
  skip 1
  fields terminated by ",";
(TITLE CHAR(100),
Publisher CHAR(20),
CategoryName CHAR(20),
Rating CHAR(2))
)
location ("bookshelf_dump_2.lst", "bookshelf_dump_1.lst")
)
```
reject limit 1

The order of the files is important—the skip 1 applies to the first file, not to the second file. The second file, bookshelf_dump.lst, is the file that was previously edited to eliminate the non-data rows in its first and last rows. The result, reflecting the rows in both, is shown in the following listing:

```sql
select COUNT(*) from BOOKSHELF_EXT_4;

COUNT(*)
--------
 62
```

Limitations, Benefits, and Potential Uses of External Tables

External tables have limitations that may make them inappropriate for some online transaction processing applications. You cannot perform any `insert`, `update`, or `delete` operations on external tables. The more dynamic the table is, the less appropriate external files may be. As shown in the examples earlier in this chapter, you can change the file dynamically at the operating system level. If your application generates `inserts` only, you may be able to write those inserted records into an external file instead of a database table.

You cannot index external tables. The lack of indexes on external tables does not have to be a negative factor in application performance. Queries of external tables complete very quickly, even though a full table scan is required with each access. There is I/O involved, but modern I/O systems use caching and RAID techniques to significantly reduce the performance penalty associated with repeated full scans of the same file.

You cannot specify constraints on an external table. Even creating a NOT NULL or foreign key constraint fails:

```sql
alter table BOOKSHELF_EXT add constraint CATFK
foreign key (CategoryName) references CATEGORY(CategoryName);

foreign key (CategoryName) references CATEGORY(CategoryName)
*  
ERROR at line 2:
ORA-30557: operation not supported on external organized table
```

Despite these limitations, external tables offer many useful features. You can join external tables (to each other, or to standard tables). You can use hints to force the optimizer to choose different join paths, and you can see the results in the query execution paths (see Chapter 38 for details on hints and the Oracle optimizer).

As an alternative to data loading, external tables offer DBAs and application developers the possibility of accessing data without supporting long-running load programs. Because the files can be edited at the operating system level, you can quickly replace a table's data without worrying about outstanding transactions modifying the table. For example, you could use this
capability to create multiple external tables and create a \texttt{union all} view across them, creating a partition view across multiple files. You can then manage each table's data separately at the file system level, replacing their contents as needed.

Because the external table can be queried, you can use the external table as the data source for an \texttt{insert as select} command. During that operation, Oracle will attempt to load the external files in parallel, potentially improving performance. To further improve performance of the \texttt{insert as select} operation, you should use the APPEND hint to force block-level inserts. When you specify the degree of parallelism for the \texttt{insert as select} operation, Oracle starts multiple ORACLE_LOADER access drivers to process the data in parallel. To further enhance load performance, avoid using variable-length fields, delimited fields, character set conversion, NULLIF, DEFAULTIF, and datatype conversion operations. Turning off \texttt{badfile} (with \texttt{nobadfile}) eliminates the costs associated with the file creation and the maintenance of the original row's context.

During the \texttt{insert as select}, you can perform functions on the data as it is processed. You can perform the functions either in the \texttt{insert as select} command syntax or in the external table definition. This capability highlights an important benefit of external tables—you can centralize the representation and processing requirements for your data, building translation routines into your table definitions. There is no processing data stored in SQL*Loader control files or PL/SQL routines; all of the logic is built into the table definition, accessible via \texttt{USER_EXTERNAL_TABLES}.

During queries, external tables allow you to select specific data sets (via the \texttt{load when} clause, as illustrated in this chapter). If you have multiple data sources for a data warehouse load, you can choose which data will be made available even while the data is outside the database. You can use this feature to maintain application availability during data loads. These loads can occur in parallel if the external file has a FIXED file format.

The limited access feature also allows you to enforce complex security rules concerning data access. For example, you may keep sensitive data outside the database, in a secure directory. Users with READ access to that directory would be able to use the external table and join it to other tables; users without that access would be limited to the data inserted into the publicly accessible tables. Highly secure data, or lightly accessed dynamic data, need not be inserted into the database until it is needed, if at all.

If you use external tables in your database architecture, you must make sure your backup and recovery plans account for those files as well as the rest of your database. If the external files change more rapidly than the database files, you may need to back them up more frequently in order to take advantage of Oracle's full recovery capabilities.