CHAPTER 9

Using Oracle8i's Import Utility

In This Chapter

Using the Import utility

Importing an entire database

Importing users

Importing a table

Using Import options

Oracle's Import utility reads data and object definitions from a file and loads them into an Oracle database. Import is the complement to Oracle's Export utility and can read files produced only by the Export utility. Import allows you to do the following:

- Load data that was exported from another database.
- Load object definitions that were exported from another database.
- Restore objects that you accidentally deleted. If you drop a table, for example, you can import it from the most recent export file.
- Import tables that you have exported as part of a reorganization process.

Much of the Import utility's functionality parallels that of the Export utility. Export is described in Chapter 8, "Using Oracle8i's Export Utility." It is recommended that you read Chapter 8 first, or at least be familiar with the Export utility, before you attempt to use the Import utility.

Using Oracle8i's Import Utility

To use the Import utility to import data into a database, you need to know how to do several operations, including:

- Starting the Import utility
- Getting help when you need it
- Passing parameters to it
- Running it interactively
- Using its prerequisites

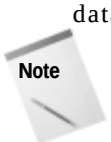
Starting the Import utility

You start the Import utility the same way that you start the Export utility, except that the command is `imp` instead of `exp`. Prior to the release of Oracle8i, the Windows versions of Oracle embedded the first two digits of the release number as part of the executable name. If you are running one of those versions, the command will be `imp80`, `imp73`, or so on, depending on the specific release of Oracle that you are using.

Like Export, Import is also a command-driven utility. On Windows NT, this means opening a Command Prompt window. The following example shows how you might import just one specific table from an export file:

```
E:\Jonathan\Oracle_Bible\ch9> imp system/manager file=bibdb log=bibdb_import
fromuser=seapark tables=caretaker

Import: Release 8.1.5.0.0 - Production on Mon Aug 9 10:52:12 1999

(c) Copyright 1999 Oracle Corporation.  All rights reserved.


Connected to: Oracle8i Release 8.1.5.0.0 - Production
With the Partitioning and Java options
PL/SQL Release 8.1.5.0.0 - Production

Export file created by EXPORT:V08.01.05 via direct path
import done in WE8ISO8859P1 character set and WE8ISO8859P1 NCHAR character set
. importing SEAPARK's objects into SEAPARK
. . importing table                    "CARETAKER"          3 rows imported
About to enable constraints...
Import terminated successfully without warnings.
```

You might import one table like this after accidentally dropping the table. Restoring only one table from a file-system backup of the Oracle datafiles is a difficult and cumbersome process. If you can afford the time to make a database export regularly, that provides you with a convenient method for restoring just one database object.



When using the Import utility to restore an object, you do have to think about the volatility of the object. After the restore, the data will be only as current as the most recent export. You can't roll forward like you can when you do a file-based recovery. The result is that you may lose data. You may also run into referential integrity problems if the data that you are importing is dependent on rows in other tables that no longer exist.

You can invoke the Import utility both in an interactive mode and a command-line mode. The interactive mode exists only for purposes of backwards compatibility with earlier releases. When you run the Import utility interactively, you respond to a number of prompts to tell the Import utility what to do. Only a small subset

of Import's functionality is available this way. The command-line interface, where you pass all information as parameters on the command line, gives you full access to all the import options. The earlier example used the command-line interface.

Getting help

The Import utility has an online help facility similar to that provided by Export. Just run Import using the HELP=Y parameter, and you will get a brief display showing all possible parameters. Listing 9-1 provides an example.

Listing 9-1: **Import's online help**

```
E:\Jonathan\Oracle_Bible\ch9>imp help=y

Import: Release 8.1.5.0.0 - Production on Mon Aug 9 09:30:59 1999

(c) Copyright 1999 Oracle Corporation.  All rights reserved.



You can let Import prompt you for parameters by entering the IMP
command followed by your username/password:

     Example: IMP SCOTT/TIGER

Or, you can control how Import runs by entering the IMP command followed
by various arguments. To specify parameters, you use keywords:

     Format:  IMP KEYWORD=value or KEYWORD=(value1,value2,...,valueN)
     Example: IMP SCOTT/TIGER IGNORE=Y TABLES=(EMP,DEPT) FULL=N
               or TABLES=(T1:P1,T1:P2), if T1 is partitioned table

USERID must be the first parameter on the command line.

Keyword  Description (Default)       Keyword      Description (Default)
------------------------------------------------------------------------------
USERID   username/password           FULL         import entire file (N)
BUFFER   size of data buffer         FROMUSER     list of owner usernames
FILE     input files (EXPDAT.DMP)    TOUSER       list of usernames
SHOW     just list file contents (N) TABLES       list of table names
IGNORE   ignore create errors (N)    RECORDLENGTH length of IO record
GRANTS   import grants (Y)           INCTYPE      incremental import type
INDEXES  import indexes (Y)          COMMIT       commit array insert (N)
ROWS     import data rows (Y)        PARFILE      parameter filename
LOG      log file of screen output   CONSTRAINTS  import constraints (Y)
DESTROY  overwrite tablespace data file (N)
INDEXFILE write table/index info to specified file
```

Continued

Listing 9-1 *(continued)*

```
SKIP_UNUSABLE_INDEXES  skip maintenance of unusable indexes (N)
ANALYZE  execute ANALYZE statements in dump file (Y)
FEEDBACK display progress every x rows(0)
TOID_NOVALIDATE  skip validation of specified type ids
FILESIZE maximum size of each dump file
RECALCULATE_STATISTICS recalculate statistics (N)

The following keywords only apply to transportable tablespaces
TRANSPORT_TABLESPACE import transportable tablespace metadata (N)
TABLESPACES tablespaces to be transported into database
DATAFILES datafiles to be transported into database
TTS_OWNERS users that own data in the transportable tablespace set

Import terminated successfully without warnings.
```

The parameter explanations that you get using `HELP=Y` are very concise, but they are helpful if you know their functionality but need a brief reminder of syntax.

Using Import parameters

The general form for invoking the Import utility looks like this:

```
imp [username[/password[@service]]] [param=value
[param=value]...]
```

Replace *username* and *password* with your username and password. If you omit either of these, Import will prompt you for them. You can use a Net8 service name to allow you to import data into a remote database. The parameters, shown as *param* in the syntax, are the ones listed in Table 9-1. You may place as many parameters on the command line as you need.

Table 9-1
Export Parameters

Parameter	*Description*
`ANALYZE`	Controls whether any `ANALYZE` commands that may be in the export file are executed. Also controls whether any statistics that were exported are loaded.

Parameter	*Description*
BUFFER	Specifies the size of the buffer used to send rows to the database. This value is in bytes. The buffer must be large enough to hold the largest row in the table.
COMMIT	Specifies whether a commit should occur after each array insert. The default setting is COMMIT=N, causing a commit to occur after loading each table. The COMMIT=Y parameter causes commits to occur more frequently, thus lessening the pressure on rollback segments. However, using COMMIT=Y slows the import down quite a bit. Another problem with COMMIT=Y is that if an import fails while a table is being loaded, the changes already made to that table cannot be rolled back. This could lead to duplicate rows after you retry the import. Use of COMMIT=Y is safest when you have a primary key or a unique key defined for the table.
CONSTRAINTS	Controls whether you want table constraints to be imported. The default is CONSTRAINTS=Y.
DATAFILES	Supplies a list of the datafile names for that tablespace if you are importing a transportable tablespace.
DESTROY	Specifies having the REUSE option automatically added to all CREATE TABLESPACE commands. The default is DESTROY=N, causing the import to fail if an attempt is made to overwrite any existing datafiles.
FEEDBACK	Specifies that Import indicate progress by displaying periods on the screen. The default is FEEDBACK=0, resulting in no progress display. The FEEDBACK=10 parameter causes a period to be written for every ten rows that are inserted, FEEDBACK=20 causes a period to be written for every 20 rows, and so forth.
FILE	Specifies the name of the export file. The default is FILE=expdat.dmp. The default file extension is .dmp.
FILESIZE	Specifies the file size that was used for a multifile export. This should match the FILESIZE setting used when you exported the data.
FROMUSER	Allows you to import specific schemas, ignoring anything else in the export file. You can list one schema or several, as shown in these two examples: FROMUSER=AMY FROMUSER=(AMY,SEAPARK) These two clauses limit an import to objects owned by AMY, or by AMY and SEAPARK, respectively. Objects in the export file owned by users that you do not list will be ignored.

Continued

Table 9-1 *(continued)*

Parameter	*Description*
FULL	Specifies that everything in the export file is to be imported when the FULL=Y parameter is used. The default is FULL=N.
GRANTS	Specifies that GRANT statements are to be imported with their tables. The GRANTS=Y parameter is the default. The GRANTS=N parameter causes tables to be imported without their associated grants.
HELP	Controls the display of the help screen shown earlier in the chapter. The parameter is HELP=Y; there is no HELP=N option.
IGNORE	Controls Import's behavior when it fails to create a table that is being imported. When you use the IGNORE=N parameter, which is the default, a table creation error will cause Import to skip to the next table. When you use the IGNORE=Y parameter, even though a table could not be created, Import will attempt to import that table's data. Use IGNORE=Y if you have manually created the tables that you are trying to import. The IGNORE=Y parameter is frequently used when reorganizing storage.
INCTYPE	Specifies the incremental import options. You can use the following option values: SYSTEM. Imports system objects RESTORE. Imports all user objects There is no default value for INCTYPE.
INDEXES	Specifies whether indexes should be imported with their tables. The default is INDEXES=Y. Use INDEXES=N if you don't want index definitions to be imported, or if you want to create the indexes manually after the import is finished.
INDEXFILE	Specifies having Import generate a text file of CREATE INDEX commands for all indexes found in the export file. The following example results in a file named BIBDB_INDEXES.SQL: INDEXFILE=BIBDB_INDEXES.SQL When the INDEXFILE parameter is used, no objects are imported. Import will create only the requested text file. There is no default value for this parameter.
LOG	Specifies the name of a log file that will accumulate information about the import, including any error messages. The default file extension is .log.
PARFILE	Allows you to read export parameters from a file. Read more on this option later in the chapter.

Parameter	*Description*
RECALCULATE_STATISTICS	Specifies that Import may execute ANALYZE commands for all tables and indexes that are imported, or it may import precalculated statistics that were written by the Export utility, when ANALYZE=Y is used. The Import utility makes the choice. Setting RECALCULATE_STATISTICS=Y forces Import to ignore any precalculated statistics and to execute ANALYZE commands instead. The default is RECALCULATE_STATISTICS=N.
RECORDLENGTH	Specifies the record length used in the export file. You may not need it, but the RECORDLENGTH parameter can be useful when transferring export files between operating systems. The value specified for RECORDLENGTH when importing should match that used when exporting.
ROWS	Controls whether table data is imported. The default is ROWS=Y, which causes data to be imported. Use ROWS=N if you just want to import table definitions. This results in empty tables.
SHOW	Specifies Import to display all the SQL statements that would be executed if it were to actually import the file being read. This is a good way to find out what's in an export file. When using SHOW=Y, no data or objects are imported—you get only the display. The default is SHOW=N.
SKIP_UNUSABLE_INDEXES	Specifies that Import not update or build indexes where the state has been set to *index unusable,* when using SKIP_UNUSABLE_INDEXES=Y. The default is SKIP_UNUSABLE_INDEXES=N.
TABLES	Allows you to import a specific table or a list of tables.
TABLESPACES	Allows you to export a specific tablespace or a list of tablespaces. The tablespaces must be locally managed, and you must use this parameter in conjunction with TRANSPORT_TABLESPACE=Y.
TOID_NOVALIDATE	Allows you to import a table based on object types without validating that the unique identifier for those types in the export file matches the unique identifier for the same types in the target database. This parameter expects a list of type names, and it should be formatted like this: `TOID_NOVALIDATE = ([schema.]type [,[schema.]type...])` There is no default value for this parameter.
TOUSER	Specifies a target list of schemas when used in conjunction with FROMUSER. This option allows you to copy objects from one schema to another. Objects owned by the users listed with the FROMUSER parameter are loaded into schemas for the users listed with the TOUSER parameter.

Continued

Table 9-1 *(continued)*

Parameter	*Description*
TRANSPORT_TABLESPACE	Allows you to import the metadata for transportable tablespaces. The default is TRANSPORT_TABLESPACE=N.
TTS_OWNERS	Lists the owners of the data in the transportable tablespaces being imported and can be used with TRANSPORT_TABLESPACE=Y. If this list of owners doesn't match the export file, Import will return an error.
USERID	Specifies the username and password of the user invoking the import.
VOLSIZE	Specifies the maximum number of bytes in export files when those files are stored on tape volumes.

The examples in this chapter show the more common uses of the Import utility. Many of these examples demonstrate the use of the parameters shown in Table 9-1.

Using interactive mode vs. command-line mode

The Import utility supports a limited interactive mode. To invoke Import interactively, simply start it without passing any parameters. The format to use is:

```
imp [username[/password[@service]]]
```

You can leave out the password, and Import will prompt you for it. If you aren't connecting to a remote database, you can leave out both the username and the password, and Import will prompt you for both. Listing 9-2 shows Import being used interactively to import the CARETAKER table owned by the user SEAPARK.

Listing 9-2: **Performing an interactive import**

```
E:\Jonathan\Oracle_Bible\ch9> imp system@bibdb

Import: Release 8.1.5.0.0 - Production on Mon Aug 9 10:55:54 1999

(c) Copyright 1999 Oracle Corporation.  All rights reserved.

Password:

Connected to: Oracle8i Release 8.1.5.0.0 - Production
With the Partitioning and Java options
PL/SQL Release 8.1.5.0.0 - Production

Import file: EXPDAT.DMP > bibdb
```

```
Enter insert buffer size (minimum is 8192) 30720>

Export file created by EXPORT:V08.01.05 via direct path
import done in WE8ISO8859P1 character set and WE8ISO8859P1 NCHAR character set
List contents of import file only (yes/no): no >

Ignore create error due to object existence (yes/no): no >

Import grants (yes/no): yes >

Import table data (yes/no): yes >

Import entire export file (yes/no): no >
Username: seapark

Enter table(T) or partition(T:P) names. Null list means all tables for user
Enter table(T) or partition(T:P) name or . if done: caretaker

Enter table(T) or partition(T:P) name or . if done:

. importing SEAPARK's objects into SEAPARK
. . importing table                    "CARETAKER"          3 rows imported
About to enable constraints...
Import terminated successfully without warnings.
```

When you use Import interactively like this, you don't have access to the utility's full functionality like you do when you pass parameters on the command line.

Using Import prerequisites

To use the Import utility, you must have the following:

- The `CREATE SESSION` privilege, allowing you to log on to the target database.
- The `CREATE` privileges for any objects that you are importing. For example, to import new tables, you need the `CREATE TABLE` privilege.
- The `IMP_FULL_DATABASE` role if you are importing from an export file that you did not create, or if you are importing objects owned by another user.
- Execute privileges on the `DBMS_RLS` package if you are importing tables with fine-grained access policies attached.

Before using Import against a database, you must run the `CATEXP.SQL` script once to create views and tables that the Import utility requires. The `IMP_FULL_DATABASE` role is one of the items that `CATEXP.SQL` creates. The `CATEXP.SQL` script is run by `CATALOG.SQL`, so if you ran `CATALOG.SQL` when you first created the database, you are all set. If you need them, you will find these scripts in the `$ORACLE_HOME/RDBMS/ADMIN` directory.

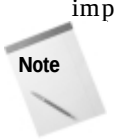
Importing an Entire Database

You can use the FULL=Y option to import the entire contents of an export file. If the export was a full database export, then importing that file results in a full database import. Listing 9-3 shows the results of a full database import:

Listing 9-3: **Performing a full database import**

```
E:\Jonathan\Oracle_Bible\ch9> imp system@bibdb full=y ignore=y file=bibdb
log=bib
db_import

Import: Release 8.1.5.0.0 - Production on Mon Aug 9 11:17:15 1999

(c) Copyright 1999 Oracle Corporation.  All rights reserved.

Password:

Connected to: Oracle8i Release 8.1.5.0.0 - Production
With the Partitioning and Java options
PL/SQL Release 8.1.5.0.0 - Production

Export file created by EXPORT:V08.01.05 via direct path
import done in WE8ISO8859P1 character set and WE8ISO8859P1 NCHAR character set
. importing SEAPARK's objects into SEAPARK
. importing AMY's objects into AMY
. importing SYS's objects into SYS
. importing SYSTEM's objects into SYSTEM
. importing SYS's objects into SYS
. importing SYSTEM's objects into SYSTEM
. . importing table                  "DEF$_AQCALL"          0 rows imported
. . importing table                 "DEF$_AQERROR"          0 rows imported

...

About to enable constraints...
. importing SCOTT's objects into SCOTT
. importing SEAPARK's objects into SEAPARK
. importing SYSTEM's objects into SYSTEM
. importing SCOTT's objects into SCOTT
. importing ORDSYS's objects into ORDSYS
. importing MDSYS's objects into MDSYS
. importing SEAPARK's objects into SEAPARK
Import terminated successfully with warnings.
```

When you do a full import like this, you may want to set the IGNORE=Y parameter. Otherwise, you will see a veritable plethora of error messages regarding system objects that couldn't be created because they already exist. Remember, though, that the IGNORE=Y parameter can cause duplicate rows in your tables if you aren't importing into an empty database.



The issue of attempting to create system objects that already exist, and having to deal with the resulting error messages, is one reason why you might try to avoid doing full database imports if you can. Instead, consider importing at either the user or the table level, where you'll have a way to know for sure whether an error message is something to be concerned about. Importing an entire database can be too broad a stroke to paint.

While Import gives you the capability to import specific tables, or to import specific users, the only way to import other database objects, such as profiles, public synonyms, and tablespace definitions, is to do a full export and import using the FULL=Y option.

Importing Users

You can use the FROMUSER and TOUSER options to import specific schemas from an export file. The FROMUSER option specifies a list of schemas to import. The TOUSER option allows you to specify a list of target schemas that differ from the original, effectively giving you a way to rename users.

Importing a specific list of users

You can import one user or a specific list of users, using the FROMUSER parameter. The format to use for FROMUSER is:

```
FROMUSER=username
FROMUSER=(username, username,...)
```

When you use the FROMUSER option, you must create the specified users in the target database before you attempt to import them. You don't have to create any of their objects before you import, but Import will expect the users to exist. Be sure, too, that you have granted the necessary CREATE privileges to the users based on the objects that you are importing.

Listing 9-4 shows FROMUSER being used to import only objects owned by AMY and SEAPARK.

Listing 9-4: Importing objects owned by a specific set of users

```
E:\Jonathan\Oracle_Bible\ch9> imp system@bibdb fromuser=(amy,seapark) file=bibdb
log=bibdb_log

Import: Release 8.1.5.0.0 - Production on Mon Aug 9 12:12:15 1999

(c) Copyright 1999 Oracle Corporation.  All rights reserved.

Password:

Connected to: Oracle8i Release 8.1.5.0.0 - Production
With the Partitioning and Java options
PL/SQL Release 8.1.5.0.0 - Production

Export file created by EXPORT:V08.01.05 via direct path
import done in WE8ISO8859P1 character set and WE8ISO8859P1 NCHAR character set
. importing SEAPARK's objects into SEAPARK
. importing AMY's objects into AMY
. importing SEAPARK's objects into SEAPARK
. . importing table               "AQUATIC_ANIMAL"         10 rows imported
. . importing table                    "CARETAKER"          3 rows imported
. . importing table                      "CHECKUP"          3 rows imported

...

. importing AMY's objects into AMY
. . importing table                       "ARTIST"         10 rows imported
. . importing table                        "BOOKS"          3 rows imported
. . importing table                "BOOKS_LOANED"          1 rows imported

...

About to enable constraints...
Import terminated successfully without warnings.
```

If the users in question own no objects before the import starts, the import should run to completion without generating any error message.

Importing one user into another

You can use the TOUSER parameter to import data owned by one user into another user's schema. If you need to make a copy of a user for testing purposes, the TOUSER parameter allows you to do that.

The syntax for TOUSER is the same as for FROMUSER. You supply either one schema name or a list of schema names separated by commas. The schema names listed in TOUSER correspond to the schema names listed with FROMUSER on a positional basis. Take a look at this example:

```
FROMUSER=(AMY,SEAPARK,HAROLD)
TOUSER=(SEAPARK, AMY)
```

In this rather extreme example, AMY's objects would be imported into the SEAPARK schema, SEAPARK's objects would be imported into AMY's schema, and you would have one mixed-up database. HAROLD's objects, since no corresponding TOUSER entry existed for HAROLD, would be imported into the HAROLD schema.

Schemas in the FROMUSER list without corresponding entries in TOUSER are imported unchanged.

TOUSER is commonly used to make a duplicate copy of a schema for testing purposes. Say that you want to make a copy of the SEAPARK schema and call it SEAPARK_COPY. You could do that by following these steps:

1. Export the current SEAPARK user. You could use a command like this:

   ```
   exp system@bibdb file=seapark log=seapark owner=seapark
   ```

2. Create a new user named SEAPARK_COPY. For example:

   ```
   CREATE USER seapark_copy identified by seapark
   DEFAULT TABLESPACE users
   TEMPORARY TABLESPACE temp
   QUOTA UNLIMITED ON users
   QUOTA UNLIMITED ON temp;
   GRANT CONNECT, RESOURCE TO seapark_copy;
   ```

3. Import the SEAPARK data into SEAPARK_COPY's schema. The following import command will do this:

   ```
   imp system@bibdb fromuser=seapark touser=seapark_copy
       file=seapark log=seapark_import
   ```

When copying a schema, you can use any export file containing that schema's objects. One approach, the one shown in Step 1, is to create a brand new export for just the one schema. Another approach is to simply use a recent full database export, if you have one. Whichever method you choose, the FROMUSER parameter in Step 3 controls which schema's data gets imported. The TOUSER parameter then determines the destination schema.

Importing a Table

You can use the TABLES parameter to limit an import to just one table or to a specific list of tables. The syntax for TABLES takes either of these forms:

```
TABLES=table_name
TABLES=(table_name, table_name...)
```

Be aware that while the Export utility allows table names listed after the TABLES parameter to be qualified with schema names, the Import utility does not. If you are trying to restore the SEAPARK user's CARETAKER table, you can't use the following parameter to do that:

```
TABLES=seapark.caretaker
```

Instead, to restore a table for a user other than yourself, you need to use both FROMUSER and TABLES together. For example:

```
FROMUSER=seapark TABLES=caretaker
```

The FROMUSER parameter gets you to the specific schema, in this case, the SEAPARK schema, while TABLES gets you to a specific list of tables within that schema, in this case, the CARETAKER table.

If an export was done by a user who was not a DBA, then another user may import that data into his or her own schema without using FROMUSER/TOUSER. If an export was done by a DBA, then it must be imported by a DBA.

Using Import Options

A lot of parameters affect how the Import utility operates. Some of the more significant parameters allow you to do the following:

- ✦ Ignore create errors
- ✦ Generate CREATE INDEX statements
- ✦ Import from multiple-file exports
- ✦ Use parameter files

This section shows you how and why to use these features.

Ignoring create errors

You will often import data into tables that you have already created. Maybe you are just using Export and Import to copy data from one database to another. Or maybe you want to export a nonpartitioned table and import it back as a partitioned table. Whatever the reason, this conflicts with Import's standard practice of attempting to create objects that it imports. When Import loads a table, it goes through the following steps:

1. It creates the table.
2. It inserts the data.
3. It creates indexes on the table.
4. It creates integrity constraints on the table.

Step 1 presents a problem if the table already exists. Import's normal behavior is to skip an object if any type of error occurs. If a table exists, Import will try to create it, the create will fail, and Import will move to the next table. The result is that no data gets loaded.

You can use `IGNORE=Y` to change Import's behavior in response to creation errors. When the `IGNORE=Y` setting is used, Import will still try to create each table that it imports. The difference is that if the table already exists, Import will ignore the creation error and proceed to insert the table's data into the preexisting table. Listing 9-5 shows a simple example that demonstrates Imports normal response to a creation error. The `AQUATIC_ANIMAL` table has already been created, but it contains no data. The import attempt doesn't specify `IGNORE=Y`, so look what happens.

Listing 9-5: **Import skips a table because of a creation error**

```
E:\Jonathan\Oracle_Bible\ch9> imp system/manager@bibdb fromuser=seapark
tables=aquatic_animal file=bibdb log=bibdb_import

Import: Release 8.1.5.0.0 - Production on Mon Aug 9 14:36:31 1999

(c) Copyright 1999 Oracle Corporation.  All rights reserved.


Connected to: Oracle8i Release 8.1.5.0.0 - Production
With the Partitioning and Java options
PL/SQL Release 8.1.5.0.0 - Production

Export file created by EXPORT:V08.01.05 via direct path
import done in WE8ISO8859P1 character set and WE8ISO8859P1 NCHAR character set
```

Continued

Listing 9-5 *(continued)*

```
importing SEAPARK's objects into SEAPARK
IMP-00015: following statement failed because the object already exists:
 "CREATE TABLE "AQUATIC_ANIMAL" ("ID_NO" NUMBER(10, 0), "TANK_NO" NUMBER(10, "
 "0), "ANIMAL_NAME" VARCHAR2(30), "MARKINGS_DESCRIPTION" VARCHAR2(30), "BIRTH"
 "_DATE" DATE, "DEATH_DATE" DATE)  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS "
 "255 LOGGING STORAGE(INITIAL 10240 NEXT 10240 MINEXTENTS 1 MAXEXTENTS 121 PC"
 "TINCREASE 50 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT) TABLESPACE "
 ""USERS""
Import terminated successfully with warnings.
```

What happened was that no data was loaded. Import detected an error when it tried to create the `AQUATIC_ANIMAL` table and did not do anything more. Listing 9-6 shows the same import again, but this time with the `IGNORE=Y` parameter setting.

Listing 9-6: **Using IGNORE=Y to ignore creation errors**

```
E:\Jonathan\Oracle_Bible\ch9> imp system/manager@bibdb fromuser=seapark
tables=aquatic_animal file=bibdb log=bibdb_import ignore=y

Import: Release 8.1.5.0.0 - Production on Mon Aug 9 14:38:56 1999

(c) Copyright 1999 Oracle Corporation.  All rights reserved.


Connected to: Oracle8i Release 8.1.5.0.0 - Production
With the Partitioning and Java options
PL/SQL Release 8.1.5.0.0 - Production

Export file created by EXPORT:V08.01.05 via direct path
import done in WE8ISO8859P1 character set and WE8ISO8859P1 NCHAR character set
. importing SEAPARK's objects into SEAPARK
. . importing table               "AQUATIC_ANIMAL"         10 rows imported
About to enable constraints...
Import terminated successfully without warnings.
```

This time, because of the `IGNORE=Y` setting, the creation error was ignored. In fact, you don't even see the error message. Import goes on to load the data into the existing table.

When you create a table before importing it, you can make almost any change that you want involving the way that the table is stored. You can change the tablespace, partition the table, change the storage parameters, and even add new columns to the table. You do, however, need to keep the following points in mind:

- ✦ Column names in the table must match those in the export file.
- ✦ Column types must be compatible. You can import a `NUMBER(10)` into a `FLOAT`, but you cannot import a `NUMBER` into a `VARCHAR2`.
- ✦ Column lengths must not shrink. Don't try to import a `VARCHAR2(30)` into a `VARCHAR2(10)`.
- ✦ You can't add `NOT NULL` columns. Since the old rows will not contain values for these columns, the `NOT NULL` constraint will always fail.

Another issue to consider, unrelated to compatibility between column types, is the effect of referential integrity constraints, especially if you are loading multiple tables. Referential integrity constraints define dependencies between tables. Normally, Import doesn't apply any referential integrity constraints until after all data has been imported. If you have first created tables with referential integrity constraints and you are importing data into these tables, you may find rows being rejected because the required parent rows don't exist. You can work around this problem by disabling all involved referential integrity constraints before you load your data. After all data in all related tables have been loaded, you can enable the constraints again.

Generating CREATE INDEX statements

If you are doing an export followed by an import to reorganize your database, you may not want the Import utility to create your indexes for you. The reason for this is that by default, Import will create the indexes in the same tablespace that they were in when they were exported, and with the same storage parameters. You may want the opportunity to change things around. You can get that opportunity by following this process:

1. Run Import with the `INDEXFILE` parameter, which generates a SQL file containing `CREATE INDEX` commands.
2. Edit the index creation commands so that they contain the desired storage and tablespace options.
3. Create the tables that you are going to import before importing.
4. Run Import using `IGNORE=Y` and `INDEXES=N` to load the data into your created tables without creating the indexes.
5. Run the index creation script.

The following example in Listing 9-7 shows how you can use the INDEXFILE parameter to generate a file containing CREATE INDEX commands.

Listing 9-7: **Placing CREATE INDEX statements into a file**

```
E:\Jonathan\Oracle_Bible\ch9> imp system/manager@bibdb
indexfile=seapark_indexes.sql fromuser=seapark file=bibdb log=bibdb_indexes

Import: Release 8.1.5.0.0 - Production on Mon Aug 9 15:20:05 1999

(c) Copyright 1999 Oracle Corporation.  All rights reserved.


Connected to: Oracle8i Release 8.1.5.0.0 - Production
With the Partitioning and Java options
PL/SQL Release 8.1.5.0.0 - Production

Export file created by EXPORT:V08.01.05 via conventional path
import done in WE8ISO8859P1 character set and WE8ISO8859P1 NCHAR character set
. . skipping table "AQUATIC_ANIMAL"

. . skipping table "CARETAKER"

. . skipping table "CHECKUP"

. . skipping table "CHECKUP_HISTORY"

. . skipping table "ITEMS"

. . skipping table "PARK_REVENUE"

. . skipping table "TANK"

Import terminated successfully without warnings.
```

The resulting file, in this instance named seapark_indexes.sql, will look like the one shown in Listing 9-8.

Listing 9-8: **The resulting seapark_indexes.sql file**

```
REM  CREATE TABLE "SEAPARK"."AQUATIC_ANIMAL" ("ID_NO" NUMBER(10, 0),
REM  "TANK_NO" FLOAT(126), "ANIMAL_NAME" VARCHAR2(30),
REM  "MARKINGS_DESCRIPTION" VARCHAR2(30), "BIRTH_DATE" DATE, "DEATH_DATE"
REM  DATE) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 LOGGING
REM  STORAGE(INITIAL 10240 NEXT 10240 MINEXTENTS 1 MAXEXTENTS 121
REM  PCTINCREASE 50 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
REM  TABLESPACE "USERS" ;
```

```
REM  ... 10 rows
CONNECT SEAPARK;
CREATE INDEX "SEAPARK"."AA_NAME" ON "AQUATIC_ANIMAL" ("ANIMAL_NAME" )
PCTFREE 10 INITRANS 2 MAXTRANS 255 STORAGE(INITIAL 10240 NEXT 10240
MINEXTENTS 1 MAXEXTENTS 121 PCTINCREASE 50 FREELISTS 1 FREELIST GROUPS 1
BUFFER_POOL DEFAULT) TABLESPACE "USERS" LOGGING ;
REM  ALTER TABLE "SEAPARK"."AQUATIC_ANIMAL" ADD CONSTRAINT "AQ_ANIMAL_PK"
REM  PRIMARY KEY ("ID_NO") USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255
REM  STORAGE(INITIAL 10240 NEXT 10240 MINEXTENTS 1 MAXEXTENTS 121
REM  PCTINCREASE 50 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
REM  TABLESPACE "USERS" ENABLE NOVALIDATE ;
```

While the format is a bit messy, the file contains a remarked-out version of the `CREATE TABLE` statement, followed by `CREATE INDEX` statements for all indexes on the table, followed by an `ALTER TABLE` statement creating the table's primary key.

By uncommenting the `CREATE TABLE` and `ALTER TABLE` statements, you can even use this SQL script as a basis for recreating the table prior to importing the data.

Importing from multiple-file exports

In Chapter 8, "Using Oracle8i's Export Utility," you saw how to use the `FILE` and `FILESIZE` parameters to create exports that spanned multiple files. When you import from such an export, you can use the same `FILE` and `FILESIZE` parameters for the import command as you originally used when exporting the data. Doing this enables Import to perform two functions:

- ✦ Check the `FILESIZE` that you specify against that recorded in the export files
- ✦ Automatically advance through the export files

If the file size specified in the import command doesn't match the size recorded in the export file, the import terminates with an error. For example, the import command shown in Listing 9-9 specifies a file size of 20KB when, in fact, a 10KB size was used when creating the export.

Listing 9-9: **An import specifying a 20KB file size**

```
E:\Jonathan\OR98C7~1\ch9> imp system/manager@jonathan.gennick
file=(seapark_1,seapark_2,seapark_3) full=y filesize=20k

Import: Release 8.1.5.0.0 - Production on Mon Aug 9 17:12:38 1999

(c) Copyright 1999 Oracle Corporation.  All rights reserved.
Connected to: Oracle8i Release 8.1.5.0.0 - Production
```

Continued

Listing 9-9 *(continued)*

```
With the Partitioning and Java options
PL/SQL Release 8.1.5.0.0 - Production

Export file created by EXPORT:V08.01.05 via conventional path
import done in WE8ISO8859P1 character set and WE8ISO8859P1 NCHAR character set
IMP-00040: FILESIZE does not match the value used for export: 10240
IMP-00000: Import terminated unsuccessfully
```

In most cases, it's probably easiest just to leave off the `FILESIZE` parameter altogether. Oracle records the file size in the export files and will automatically advance from one file to the next as long as you list all the file names using the `FILE` parameter. See Listing 9-10.

Listing 9-10: **A multifile import**

```
E:\Jonathan\OR98C7~1\ch9> imp system/manager@jonathan.gennick
file=(seapark_1,seapark_2,seapark_3) full=y

Import: Release 8.1.5.0.0 - Production on Mon Aug 9 17:18:29 1999

(c) Copyright 1999 Oracle Corporation.  All rights reserved.


Connected to: Oracle8i Release 8.1.5.0.0 - Production
With the Partitioning and Java options
PL/SQL Release 8.1.5.0.0 - Production

Export file created by EXPORT:V08.01.05 via conventional path
import done in WE8ISO8859P1 character set and WE8ISO8859P1 NCHAR character set
IMP-00046: using FILESIZE value from export file of 10240
. importing SYSTEM's objects into SYSTEM
. importing SEAPARK's objects into SEAPARK
. . importing table               "AQUATIC_ANIMAL"         10 rows imported
. . importing table                    "CARETAKER"          3 rows imported
. . importing table                      "CHECKUP"          3 rows imported
. . importing table             "CHECKUP_HISTORY"          0 rows imported
. . importing table                       "ITEMS"          0 rows imported
. . importing table                  "PARK_REVENUE"          4 rows imported
. . importing table                    "PLAN_TABLE"          4 rows imported
. . importing table                         "TANK"          3 rows imported
Import terminated successfully with warnings.
```

In this case, the warning occurs only if you did not explicitly use the FILESIZE parameter. This is nothing to worry about. Import used the size recorded in the export file.

If you've done a multifile export, but you forget to specify all the file names when doing the import, the Import utility will prompt you for the missing files, as shown in Listing 9-11.

Listing 9-11: **The Import utility prompting for missing files**

```
E:\Jonathan\OR98C7~1\ch9> imp system/manager@jonathan.gennick file=seapark_1
full
=y

Import: Release 8.1.5.0.0 - Production on Mon Aug 9 17:22:36 1999

(c) Copyright 1999 Oracle Corporation.  All rights reserved.


Connected to: Oracle8i Release 8.1.5.0.0 - Production
With the Partitioning and Java options
PL/SQL Release 8.1.5.0.0 - Production

Export file created by EXPORT:V08.01.05 via conventional path
import done in WE8ISO8859P1 character set and WE8ISO8859P1 NCHAR character set
IMP-00046: using FILESIZE value from export file of 10240
. importing SYSTEM's objects into SYSTEM
. importing SEAPARK's objects into SEAPARK
. . importing table               "AQUATIC_ANIMAL"         10 rows imported
. . importing table                    "CARETAKER"          3 rows imported
. . importing table                      "CHECKUP"          3 rows imported
. . importing table             "CHECKUP_HISTORY"          0 rows imported
. . importing table                       "ITEMS"          0 rows imported
. . importing table                  "PARK_REVENUE"          4 rows imported
. . importing table                    "PLAN_TABLE"
Import file: EXPDAT.DMP > seapark_2
```

When Import prompts for a file name, it won't know what the next file name should be. The default, shown in the prompt, will be EXPDAT.DMP. You'll have to type the name of the next export file to open. However, if you get the files out of sequence, Import will detect that and return an error message.

Using parameter files

You use parameter files with the Import utility for the same reason that you use them with the Export utility. Parameter files allow you to define import jobs that you can run repeatedly. For example, if you were regularly importing the SEAPARK user's objects into the SEAPARK_COPY schema, you could build a text file with the following contents:

```
FROMUSER=seapark
TOUSER=seapark_copy
FILE=seapark
LOG=seapark_import
```

If the text file containing these parameters were named copy_seapark.par, you could invoke the import using this short command:

```
imp system@bibdb PARFILE=copy_seapark.par
```

Using parameter files like this allows you to maintain consistency from one import to the next and also allows you to deal with extremely long parameter lists. If you need to list 1,000 tables after the TABLES parameter, for example, it's unlikely that your operating system will support a command that long. Parameter files provide you with a way to handle those situations.

Summary

In this chapter, you learned:

- The Import utility is the complement of the Export utility and is used to load data from an export file back into a database.
- You use the HELP=Y command-line parameter to get a brief help screen describing all the import parameters.
- The FULL=Y parameter tells Import to import everything in the file. To limit the import to a specific user, use the FROMUSER parameter. To further limit the export to a specific table, use the TABLES parameter.
- If you are reorganizing storage and you have first created the tables that you are importing, specify IGNORE=Y as a parameter so that Import will ignore any errors it receives while trying to create those tables again.
- If you don't want Import to create your indexes for you, specify INDEXES=N. Then, run the import again using the INDEXFILE parameter to get a SQL file containing CREATE INDEX statements. You can then edit those statements before executing the file.

✦ ✦ ✦