28
Original Import

The original Import utility (`imp`) imports dump files that were created using the original Export utility (`exp`).

The original Export utility is desupported.

- What Is the Import Utility?
  The original Import utility (`imp`) read object definitions and table data from dump files created by the original Export utility (`exp`).
- Table Objects: Order of Import
  Table objects are imported as they are read from the export dump file.
- Before Using Import
  Learn what you should do before using the original import tool.
- Importing into Existing Tables
  These sections describe factors to consider when you import data into existing tables.
- Effect of Schema and Database Triggers on Import Operations
  Triggers that are defined to trigger on DDL events for a specific schema or on DDL-related events for the database, are system triggers.
- Invoking Import
  To start the original Import utility and specify parameters, use one of three different methods.
- Import Modes
  The Import utility supports four modes of operation.
- Import Parameters
  These sections contain descriptions of the Import command-line parameters.
- Example Import Sessions
  These sections give some examples of import sessions that show you how to use the parameter file and command-line methods.
- Exit Codes for Inspection and Display
  Import provides the results of an operation immediately upon completion. Depending on the platform, the outcome may be reported in a process exit code and the results recorded in the log file.
- Error Handling During an Import
  These sections describe errors that can occur when you import database objects.
- Table-Level and Partition-Level Import
  You can import tables, partitions, and subpartitions.
- Controlling Index Creation and Maintenance
  These sections describe the behavior of Import with respect to index creation and maintenance.
- Network Considerations for Using Oracle Net with Original Import
  To perform imports over a network, you can use the Oracle Data Pump original Import utility (`imp`) with Oracle Net.

- Character Set and Globalization Support Considerations
  These sections describe the globalization support behavior of Import with respect to character set conversion of user data and data definition language (DDL).
- Using Instance Affinity
  You can use instance affinity to associate jobs with instances in databases you plan to export and import.
- Considerations When Importing Database Objects
  These sections describe restrictions and points you should consider when you import particular database objects.
- Support for Fine-Grained Access Control
  To restore the fine-grained access control policies, the user who imports from an export file containing such tables must have the `EXECUTE` privilege on the `DBMS_RLS` package, so that the security policies on the tables can be reinstated.
- Snapshots and Snapshot Logs
  In certain situations, particularly those involving data warehousing, snapshots may be referred to as *materialized views*. These sections retain the term snapshot.
- Transportable Tablespaces
  The transportable tablespace feature enables you to move a set of tablespaces from one Oracle database to another.
- Storage Parameters
  By default, a table is imported into its original tablespace.
- Read-Only Tablespaces
  Read-only tablespaces can be exported. On import, if the tablespace does not already exist in the target database, then the tablespace is created as a read/write tablespace.
- Dropping a Tablespace
  You can drop a tablespace by redefining the objects to use different tablespaces before the import. You can then issue the `imp` command and specify `IGNORE=y`.
- Reorganizing Tablespaces
  If a user's quota allows it, the user's tables are imported into the same tablespace from which they were exported.
- Importing Statistics
  If statistics are requested at export time and analyzer statistics are available for a table, then Export will include the `ANALYZE` statement used to recalculate the statistics for the table into the dump file.
- Using Export and Import to Partition a Database Migration
  When you use the Export and Import utilities to migrate a large database, it may be more efficient to partition the migration into multiple export and import jobs.
- Tuning Considerations for Import Operations
  These sections discuss some ways to improve the performance of an import operation.
- Using Different Releases of Export and Import
  These sections describe compatibility issues that relate to using different releases of Export and the Oracle database.

28.1 What Is the Import Utility?

The original Import utility (`imp`) read object definitions and table data from dump files created by the original Export utility (`exp`).

> **Note:**
>
> Original Export is desupported for general use as of Oracle Database 11g. The only supported use of original Export in Oracle Database 11g and later releases is backward migration of `XMLType` data to Oracle Database 10g Release 2 (10.2) or earlier. Oracle strongly recommends that you use the new Oracle Data Pump Export and Import utilities. The only exception to this guidelines is in the following situations, which require original Export and Import:
>
> - You want to import files that were created using the original Export utility (`exp`).
> - You want to export files that must be imported using the original Import utility (`imp`). An example of this would be exporting data from Oracle Database 10g and then importing it into an earlier database release.

If you use original Import, then the following conditions must be true:

- The dump file is in an Oracle binary-format that can be read only by original Import.
- The version of the Import utility cannot be earlier than the version of the Export utility used to create the dump file.

28.2 Table Objects: Order of Import

Table objects are imported as they are read from the export dump file.

The dump file contains objects in the following order:

1. Type definitions
2. Table definitions
3. Table data
4. Table indexes
5. Integrity constraints, views, procedures, and triggers
6. Bitmap, function-based, and domain indexes

The order of import is as follows: new tables are created, data is imported and indexes are built, triggers are imported, integrity constraints are enabled on the new tables, and any bitmap, function-based, and/or domain indexes are built. This sequence prevents data from being rejected due to the order in which tables are imported. This sequence also prevents redundant triggers from firing twice on the same data (once when it is originally inserted and again during the import).

28.3 Before Using Import

Learn what you should do before using the original import tool.

- Overview of Import Preparation
  To prepare for the import, check to make sure you have run scripts as required, and have access privileges
- Running catexp.sql or catalog.sql
  To use Import, you must run the script `catexp.sql` or `catalog.sql` (which runs `catexp.sql`) after the database has been created or migrated to a newer version.

- Verifying Access Privileges for Import Operations
  To use Import, you must have the `CREATE SESSION` privilege on an Oracle database. This privilege belongs to the `CONNECT` role established during database creation.
- Processing Restrictions
  Restrictions apply when you process data with the Import utility.

28.3.1 Overview of Import Preparation

To prepare for the import, check to make sure you have run scripts as required, and have access privileges

Before you begin using Import, be sure you take care of the following items

- If you created your database manually, ensure that the `catexp.sql` or `catalog.sql` script has been run. If you created your database using the Database Configuration Assistant (DBCA), it is not necessary to run these scripts.
- Verify that you have the required access privileges.

28.3.2 Running catexp.sql or catalog.sql

To use Import, you must run the script `catexp.sql` or `catalog.sql` (which runs `catexp.sql`) after the database has been created or migrated to a newer version.

The `catexp.sql` or `catalog.sql` script needs to be run only once on a database. The script performs the following tasks to prepare the database for export and import operations:

- Creates the necessary import views in the data dictionary
- Creates the `EXP_FULL_DATABASE` and `IMP_FULL_DATABASE` roles
- Assigns all necessary privileges to the `EXP_FULL_DATABASE` and `IMP_FULL_DATABASE` roles
- Assigns `EXP_FULL_DATABASE` and `IMP_FULL_DATABASE` to the `DBA` role
- Records the version of `catexp.sql` that has been installed

28.3.3 Verifying Access Privileges for Import Operations

To use Import, you must have the `CREATE SESSION` privilege on an Oracle database. This privilege belongs to the `CONNECT` role established during database creation.

You can perform an import operation even if you did not create the export file. However, keep in mind that if the export file was created by a user with the `EXP_FULL_DATABASE` role, then you must have the `IMP_FULL_DATABASE` role to import it. Both of these roles are typically assigned to database administrators (DBAs).

- Importing Objects Into Your Own Schema
  To import objects into your own schema, check the privileges required for each object.
- Importing Grants
  To import the privileges that a user has granted to others, the user initiating the import must either own the objects, or have object privileges with the option `WITH GRANT OPTION`.
- Importing Objects Into Other Schemas
  To import objects into another user's schema, you must have the `IMP_FULL_DATABASE` role enabled.

- Importing System Objects
  To import system objects from a full database export file, the `IMP_FULL_DATABASE` role must be enabled.

28.3.3.1 Importing Objects Into Your Own Schema

To import objects into your own schema, check the privileges required for each object.

The following table lists the privileges required to import objects into your own schema. All of these privileges initially belong to the `RESOURCE` role.

Table 28-1 Privileges Required to Import Objects into Your Own Schema

Object	Required Privilege (Privilege Type, If Applicable)
Clusters	`CREATE CLUSTER` (System) or `UNLIMITED TABLESPACE` (System). The user must also be assigned a tablespace quota.
Database links	`CREATE DATABASE LINK` (System) and `CREATE SESSION` (System) on remote database
Triggers on tables	`CREATE TRIGGER` (System)
Triggers on schemas	`CREATE ANY TRIGGER` (System)
Indexes	`CREATE INDEX` (System) or `UNLIMITED TABLESPACE` (System). The user must also be assigned a tablespace quota.
Integrity constraints	`ALTER TABLE` (Object)
Libraries	`CREATE ANY LIBRARY` (System)
Packages	`CREATE PROCEDURE` (System)
Private synonyms	`CREATE SYNONYM` (System)
Sequences	`CREATE SEQUENCE` (System)
Snapshots	`CREATE SNAPSHOT` (System)
Stored functions	`CREATE PROCEDURE` (System)
Stored procedures	`CREATE PROCEDURE` (System)
Table data	`INSERT TABLE` (Object)
Table definitions (including comments and audit options)	`CREATE TABLE` (System) or `UNLIMITED TABLESPACE` (System). The user must also be assigned a tablespace quota.
Views	`CREATE VIEW` (System) and `SELECT` (Object) on the base table, or `SELECT ANY TABLE` (System)
Object types	`CREATE TYPE` (System)
Foreign function libraries	`CREATE LIBRARY` (System)
Dimensions	`CREATE DIMENSION` (System)
Operators	`CREATE OPERATOR` (System)
Indextypes	`CREATE INDEXTYPE` (System)

28.3.3.2 Importing Grants

To import the privileges that a user has granted to others, the user initiating the import must either own the objects, or have object privileges with the option `WITH GRANT OPTION`.

The following table shows the required conditions for the authorizations to be valid on the target system.

Table 28-2 Privileges Required to Import Grants

Grant	Conditions
Object privileges	Either the object must exist in the user's schema, *or* the user must have the object privileges with the `WITH GRANT OPTION` *or*, the user must have the `IMP_FULL_DATABASE` role enabled.
System privileges	Users must have the `SYSTEM` privilege and also the `WITH ADMIN OPTION`.

28.3.3.3 Importing Objects Into Other Schemas

To import objects into another user's schema, you must have the `IMP_FULL_DATABASE` role enabled.

28.3.3.4 Importing System Objects

To import system objects from a full database export file, the `IMP_FULL_DATABASE` role must be enabled.

The parameter `FULL` specifies that the following system objects are included in the import:

- Profiles
- Public database links
- Public synonyms
- Roles
- Rollback segment definitions
- Resource costs
- Foreign function libraries
- Context objects
- System procedural objects
- System audit options
- System privileges
- Tablespace definitions
- Tablespace quotas
- User definitions
- Directory aliases
- System event triggers

28.3.4 Processing Restrictions

Restrictions apply when you process data with the Import utility.

Specifically:

- When a type definition has evolved and data referencing that evolved type is exported, the type definition on the import system must have evolved in the same manner.
- The table compression attribute of tables and partitions is preserved during export and import. However, the import process does not use the direct path API, hence the data will not be stored in the compressed format when imported.

28.4 Importing into Existing Tables

These sections describe factors to consider when you import data into existing tables.

- Manually Creating Tables Before Importing Data
  You can manually create tables before importing data.
- Disabling Referential Constraints
  Describes how to disable referential constraints.
- Manually Ordering the Import
  Describes manually ordering the import.

28.4.1 Manually Creating Tables Before Importing Data

You can manually create tables before importing data.

When you choose to create tables manually before importing data into them from an export file, you should use either the same table definition previously used or a compatible format. For example, although you can increase the width of columns and change their order, you cannot do the following:

- Add `NOT NULL` columns
- Change the data type of a column to an incompatible data type (`LONG` to `NUMBER`, for example)
- Change the definition of object types used in a table
- Change `DEFAULT` column values

> **Note:**
>
> When tables are manually created before data is imported, the `CREATE TABLE` statement in the export dump file will fail because the table already exists. To avoid this failure and continue loading data into the table, set the Import parameter `IGNORE=y`. Otherwise, no data will be loaded into the table because of the table creation error.

28.4.2 Disabling Referential Constraints

Describes how to disable referential constraints.

In the normal import order, referential constraints are imported only after all tables are imported. This sequence prevents errors that could occur if a referential integrity constraint exists for data that has not yet been imported.

These errors can still occur when data is loaded into existing tables. For example, if table `emp` has a referential integrity constraint on the `mgr` column that verifies that the manager number exists in `emp`, then a legitimate employee row might fail the referential integrity constraint if the manager's row has not yet been imported.

When such an error occurs, Import generates an error message, bypasses the failed row, and continues importing other rows in the table. You can disable constraints manually to avoid this.

Referential constraints between tables can also cause problems. For example, if the `emp` table appears before the `dept` table in the export dump file, but a referential check exists from the `emp` table into the `dept` table, then some of the rows from the `emp` table may not be imported due to a referential constraint violation.

To prevent errors like these, you should disable referential integrity constraints when importing data into existing tables.

28.4.3 Manually Ordering the Import

Describes manually ordering the import.

When the constraints are reenabled after importing, the entire table is checked, which may take a long time for a large table. If the time required for that check is too long, then it may be beneficial to order the import manually.

To do so, perform several imports from an export file instead of one. First, import tables that are the targets of referential checks. Then, import the tables that reference them. This option works if tables do not reference each other in a circular fashion, and if a table does not reference itself.

28.5 Effect of Schema and Database Triggers on Import Operations

Triggers that are defined to trigger on DDL events for a specific schema or on DDL-related events for the database, are system triggers.

These triggers can have detrimental effects on certain import operations. For example, they can prevent successful re-creation of database objects, such as tables. This causes errors to be returned that give no indication that a trigger caused the problem.

Database administrators and anyone creating system triggers should verify that such triggers do not prevent users from performing database operations for which they are authorized. To test a system trigger, take the following steps:

1. Define the trigger.
2. Create some database objects.
3. Export the objects in table or user mode.
4. Delete the objects.
5. Import the objects.
6. Verify that the objects have been successfully re-created.

> **Note:**
>
> A full export does not export triggers owned by schema `SYS`. You must manually re-create `SYS` triggers either before or after the full import. Oracle recommends that you re-create them after the import in case they define actions that would impede progress of the import.

28.6 Invoking Import

To start the original Import utility and specify parameters, use one of three different methods.

The three methods you have to start the original Import utility are:

- Command-line entries
- Parameter files
- Interactive mode

Before you use one of these methods, be sure to read the descriptions of the available parameters.

- Command-Line Entries
  You can specify all valid parameters and their values from the command line.
- Parameter Files
  You can specify all valid parameters and their values in a parameter file.
- Interactive Mode
  If you prefer to be prompted for the value of each parameter, then you can simply specify `imp` at the command line.
- Invoking Import As SYSDBA
  Starting the original Import utility as `SYSDBA` is a specialized procedure, which should only be done under specific scenarios.
- Getting Online Help
  Import provides online help. Enter `imp help=y` to display Import help.

Related Topics

- Import Parameters
  These sections contain descriptions of the Import command-line parameters.

28.6.1 Command-Line Entries

You can specify all valid parameters and their values from the command line.

Use the following syntax (you will then be prompted for a username and password):

```
imp PARAMETER=value
```

or

```
imp PARAMETER=(value1,value2,...,valuen)
```

The number of parameters cannot exceed the maximum length of a command line on the system.

28.6.2 Parameter Files

You can specify all valid parameters and their values in a parameter file.

Storing the parameters in a file allows them to be easily modified or reused. If you use different parameters for different databases, then you can have multiple parameter files.

Create the parameter file using any flat file text editor. The command-line option `PARFILE=filename` tells Import to read the parameters from the specified file rather than from the command line. For example:

The syntax for parameter file specifications can be any of the following:

```
PARAMETER=value
PARAMETER=(value)
PARAMETER=(value1, value2, ...)
```

The following example shows a partial parameter file listing:

```
FULL=y
FILE=dba.dmp
GRANTS=y
INDEXES=y
CONSISTENT=y
```

> **Note:**
>
> The maximum size of the parameter file may be limited by the operating system. The name of the parameter file is subject to the file-naming conventions of the operating system.

You can add comments to the parameter file by preceding them with the pound (#) sign. Import ignores all characters to the right of the pound (#) sign.

You can specify a parameter file at the same time that you are entering parameters on the command line. In fact, you can specify the same parameter in both places. The position of the `PARFILE` parameter and other parameters on the command line determines which parameters take precedence. For example, assume the parameter file `params.dat` contains the parameter `INDEXES=y` and Import is started with the following line:

```
imp PARFILE=params.dat INDEXES=n
```

In this case, because `INDEXES=n` occurs *after* `PARFILE=params.dat`, `INDEXES=n` overrides the value of the `INDEXES` parameter in the parameter file.

> **See Also:**
>
> - Import Parameters
> - Network Considerations for information about how to specify an export from a remote database

28.6.3 Interactive Mode

If you prefer to be prompted for the value of each parameter, then you can simply specify `imp` at the command line.

You will be prompted for a username and password.

Commonly used parameters are then displayed. You can accept the default value, if one is provided, or enter a different value. The command-line interactive method does not provide prompts for all functionality and is provided only for backward compatibility.

28.6.4 Invoking Import As SYSDBA

Starting the original Import utility as `SYSDBA` is a specialized procedure, which should only be done under specific scenarios.

`SYSDBA` is used internally, and has specialized functions; its behavior is not the same as for generalized users. For this reason, you should not typically need to start Import as `SYSDBA`, except in the following situations:

- At the request of Oracle technical support
- When importing a transportable tablespace set

28.6.5 Getting Online Help

Import provides online help. Enter `imp help=y` to display Import help.

28.7 Import Modes

The Import utility supports four modes of operation.

Specifically:

- Full: Imports a full database. Only users with the `IMP_FULL_DATABASE` role can use this mode. Use the `FULL` parameter to specify this mode.
- Tablespace: Enables a privileged user to move a set of tablespaces from one Oracle database to another. Use the `TRANSPORT_TABLESPACE` parameter to specify this mode.
- User: Enables you to import all objects that belong to you (such as tables, grants, indexes, and procedures). A privileged user importing in user mode can import all objects in the schemas of a specified set of users. Use the `FROMUSER` parameter to specify this mode.
- Table: Enables you to import specific tables and partitions. A privileged user can qualify the tables by specifying the schema that contains them. Use the `TABLES` parameter to specify this mode.

> **Note:**
>
> When you use table mode to import tables that have columns of type ANYDATA, you may receive the following error:
>
> ORA-22370: Incorrect usage of method. Nonexistent type.
>
> This indicates that the ANYDATA column depends on other types that are not present in the database. You must manually create dependent types in the target database before you use table mode to import tables that use the ANYDATA type.

A user with the IMP_FULL_DATABASE role must specify one of these modes. Otherwise, an error results. If a user without the IMP_FULL_DATABASE role fails to specify one of these modes, then a user-level Import is performed.

> **Note:**
>
> As of Oracle Database 12*c* release 2 (12.2) the import utility (imp), for security reasons, will no longer import objects as user SYS. If a dump file contains objects that need to be re-created as user SYS, then the imp utility tries to re-create them as user SYSTEM instead. If the object cannot be re-created by user SYSTEM, then you must manually re-create the object yourself after the import is completed.
>
> If the import job is run by a user with the imp_full_database role, then you receive a IMP-403 warning, and an empty file ("xx_sys.sql") is generated.
>
> If the import job is run by a user with the DBA role, and not all objects can be re-created by user SYSTEM, then the following warning message is written to the log file:
>
> ```
> IMP-00403: Warning: This import generated a separate SQL file
> "logfilename_sys" which contains DDL that failed due to a privilege
> issue.
> ```
>
> The SQL file that is generated contains the failed DDL of objects that could not be re-created by user SYSTEM. To re-create those objects, you must manually execute the failed DDL after the import finishes.
>
> The SQL file is automatically named by appending '_sys.sql' to the file name specified for the LOG parameter. For example, if the log file name was JulyImport, then the SQL file name would be JulyImport_sys.sql.
>
> If no log file was specified, then the default name of the SQL file is import_sys.sql.
>
> Note: Not all import jobs generate a SQL file; only those jobs run as user DBA.

The following table lists the objects that are imported in each mode.

Table 28-3 Objects Imported in Each Mode

Object	Table Mode	User Mode	Full Database Mode	Tablespace Mode
Analyze cluster	No	Yes	Yes	No
Analyze tables/statistics	Yes	Yes	Yes	Yes

Table 28-3 (Cont.) Objects Imported in Each Mode

Object	Table Mode	User Mode	Full Database Mode	Tablespace Mode
Application contexts	No	No	Yes	No
Auditing information	Yes	Yes	Yes	No
B-tree, bitmap, domain function-based indexes	Yes[1]	Yes	Yes	Yes
Cluster definitions	No	Yes	Yes	Yes
Column and table comments	Yes	Yes	Yes	Yes
Database links	No	Yes	Yes	No
Default roles	No	No	Yes	No
Dimensions	No	Yes	Yes	No
Directory aliases	No	No	Yes	No
External tables (without data)	Yes	Yes	Yes	No
Foreign function libraries	No	Yes	Yes	No
Indexes owned by users other than table owner	Yes (Privileged users only)	Yes	Yes	Yes
Index types	No	Yes	Yes	No
Java resources and classes	No	Yes	Yes	No
Job queues	No	Yes	Yes	No
Nested table data	Yes	Yes	Yes	Yes
Object grants	Yes (Only for tables and indexes)	Yes	Yes	Yes
Object type definitions used by table	Yes	Yes	Yes	Yes
Object types	No	Yes	Yes	No
Operators	No	Yes	Yes	No
Password history	No	No	Yes	No
Postinstance actions and objects	No	No	Yes	No
Postschema procedural actions and objects	No	Yes	Yes	No
Posttable actions	Yes	Yes	Yes	Yes
Posttable procedural actions and objects	Yes	Yes	Yes	Yes
Preschema procedural objects and actions	No	Yes	Yes	No
Pretable actions	Yes	Yes	Yes	Yes
Pretable procedural actions	Yes	Yes	Yes	Yes
Private synonyms	No	Yes	Yes	No
Procedural objects	No	Yes	Yes	No

Table 28-3 (Cont.) Objects Imported in Each Mode

Object	Table Mode	User Mode	Full Database Mode	Tablespace Mode
Profiles	No	No	Yes	No
Public synonyms	No	No	Yes	No
Referential integrity constraints	Yes	Yes	Yes	No
Refresh groups	No	Yes	Yes	No
Resource costs	No	No	Yes	No
Role grants	No	No	Yes	No
Roles	No	No	Yes	No
Rollback segment definitions	No	No	Yes	No
Security policies for table	Yes	Yes	Yes	Yes
Sequence numbers	No	Yes	Yes	No
Snapshot logs	No	Yes	Yes	No
Snapshots and materialized views	No	Yes	Yes	No
System privilege grants	No	No	Yes	No
Table constraints (primary, unique, check)	Yes	Yes	Yes	Yes
Table data	Yes	Yes	Yes	Yes
Table definitions	Yes	Yes	Yes	Yes
Tablespace definitions	No	No	Yes	No
Tablespace quotas	No	No	Yes	No
Triggers	Yes	Yes[2]	Yes[3]	Yes
Triggers owned by other users	Yes (Privileged users only)	No	No	No
User definitions	No	No	Yes	No
User proxies	No	No	Yes	No
User views	No	Yes	Yes	No
User-stored procedures, packages, and functions	No	Yes	Yes	No

1 Nonprivileged users can export and import only indexes they own on tables they own. They cannot export indexes they own that are on tables owned by other users, nor can they export indexes owned by other users on their own tables. Privileged users can export and import indexes on the specified users' tables, even if the indexes are owned by other users. Indexes owned by the specified user on other users' tables are not included, unless those other users are included in the list of users to export.

2 Nonprivileged and privileged users can export and import all triggers owned by the user, even if they are on tables owned by other users.

3 A full export does not export triggers owned by schema SYS. You must manually re-create SYS triggers either before or after the full import. Oracle recommends that you re-create them after the import in case they define actions that would impede progress of the import.

28.8 Import Parameters

These sections contain descriptions of the Import command-line parameters.

- BUFFER
  The `BUFFER` import parameter defines the size, in bytes, of the buffer through which data rows are transferred
- COMMIT
  The `COMMIT` import parameter specifies whether Import performs a commit after each array insert
- COMPILE
  The `COMPILE` Import parameter specifies whether Import compiles packages, procedures, and functions as they are created.
- CONSTRAINTS
  The `CONSTRAINTS` Import parameter specifies whether table constraints are imported.
- DATA_ONLY
  The `DATA_ONLY` Import parameter imports only data from a dump file.
- DATAFILES
  The `DATAFILES` Import parameter lists the data files that you want to transport into the database.
- DESTROY
  The `DESTROY` Import parameter specifies whether the existing data files making up the database should be reused.
- FEEDBACK
  The `FEEDBACK` Import utility parameter specifies that Import should display a progress meter in the form of a period for *n* number of rows imported.
- FILE
  The `FILE` Import utility parameter specifies the names of the export files to import.
- FILESIZE
  The `FILESIZE` Import utility parameter lets you specify the same maximum dump file size that you specified on export.
- FROMUSER
  The `FROMUSER` parameter of the Import utility enables you to import a subset of schemas from an export file containing multiple schemas.
- FULL
  The `FULL` Import utility parameter specifies whether to import the entire export dump file.
- GRANTS
  Specifies whether to import object grants.
- HELP
  The `HELP` parameter of Import utility displays a description of the Import parameters.
- IGNORE
  The IGNORE Import utility parameter specifies how object creation errors should be handled.
- INDEXES
  Indexes import parameter specifies whether to import indexes.
- INDEXFILE
  INDEXFILE parameter of Import utility specifies a file to receive index-creation statements.
- LOG
  Specifies a file (for example, import.log) to receive informational and error messages.

- PARFILE
  Specifies a file name for a file that contains a list of Import parameters.
- RECORDLENGTH
  Specifies the length, in bytes, of the file record.
- RESUMABLE
  The `RESUMABLE` parameter is used to enable and disable resumable space allocation.
- RESUMABLE_NAME
  The value for the `RESUMABLE_NAME` parameter identifies the statement that is resumable.
- RESUMABLE_TIMEOUT
  The value of the `RESUMABLE_TIMEOUT` parameter specifies the time period during which an error must be fixed.
- ROWS
  Specifies whether to import the rows of table data.
- SHOW
  Lists the contents of the export file before importing.
- SKIP_UNUSABLE_INDEXES
  Both Import and the Oracle database provide a `SKIP_UNUSABLE_INDEXES` parameter.
- STATISTICS
  Specifies what is done with the database optimizer statistics at import time.
- STREAMS_CONFIGURATION
  Specifies whether to import any general GoldenGate Replication metadata that may be present in the export dump file.
- STREAMS_INSTANTIATION
  Specifies whether to import Streams instantiation metadata that may be present in the export dump file.
- TABLES
- TABLESPACES
  The `TABLESPACES` parameter for the Import utility.
- TOID_NOVALIDATE
  Use the `TOID_NOVALIDATE` parameter to specify types to exclude from TOID comparison.
- TOUSER
  Specifies a list of user names whose schemas will be targets for Import.
- TRANSPORT_TABLESPACE
  When specified as `y`, instructs Import to import transportable tablespace metadata from an export file.
- TTS_OWNERS
  When `TRANSPORT_TABLESPACE` is specified as `y`, use this parameter to list the users who own the data in the transportable tablespace set.
- USERID (username/password)
  Specifies the username, password, and an optional connect string of the user performing the import.
- VOLSIZE
  Specifies the maximum number of bytes in a dump file on each volume of tape.

28.8.1 BUFFER

The BUFFER import parameter defines the size, in bytes, of the buffer through which data rows are transferred

Default

Operating system-dependent

Description

The integer specified for BUFFER is the size, in bytes, of the buffer through which data rows are transferred.

BUFFER determines the number of rows in the array inserted by Import. The following formula gives an approximation of the buffer size that inserts a given array of rows:

*buffer_size = rows_in_array * maximum_row_size*

That is, the buffer size is equal to the rows in the array multiplied by the maximum row size.

For tables containing LOBs, LONG, BFILE, REF, ROWID, UROWID, or TIMESTAMP columns, rows are inserted individually. The size of the buffer must be large enough to contain the entire row, except for LOB and LONG columns. If the buffer cannot hold the longest row in a table, then Import attempts to allocate a larger buffer.

For DATE columns, two or more rows are inserted at once if the buffer is large enough.

> **Note:**
>
> See your Oracle operating system-specific documentation to determine the default value for this parameter.
>
> All forms of LONG data types (LONG, LONG RAW, LONG VARCHAR, LONG VARRAW) were deprecated in Oracle8i Release 8.1.6. For succeeding releases, the LONG data type was provided for backward compatibility with existing applications. In new applications developed with later releases, Oracle strongly recommends that you use CLOB and NCLOB data types for large amounts of character data.

28.8.2 COMMIT

The COMMIT import parameter specifies whether Import performs a commit after each array insert

Default

n

Purpose

Specifies whether Import should commit after each array insert. By default, Import commits only after loading each table, and Import performs a rollback when an error occurs, before continuing with the next object.

If a table has nested table columns or attributes, then the contents of the nested tables are imported as separate tables. Therefore, the contents of the nested tables are always committed in a transaction distinct from the transaction used to commit the outer table.

If `COMMIT=n`, and a table is partitioned, then each partition and subpartition in the Export file is imported in a separate transaction.

For tables containing LOBs, `LONG`, `BFILE`, `REF`, `ROWID`, `UROWID`, or `TIMESTAMP` columns, array inserts are not done. If `COMMIT=y`, then Import commits these tables after each row.

> **Note:**
>
> All forms of `LONG` data types (`LONG`, `LONG RAW`, `LONG VARCHAR`, `LONG VARRAW`) were deprecated in Oracle8i Release 8.1.6. For succeeding releases, the `LONG` data type was provided for backward compatibility with existing applications. In new applications developed with later releases, Oracle strongly recommends that you use `CLOB` and `NCLOB` data types for large amounts of character data.

28.8.3 COMPILE

The `COMPILE` Import parameter specifies whether Import compiles packages, procedures, and functions as they are created.

Default

`y`

Purpose

Specifies whether Import compiles packages, procedures, and functions as they are created.

If `COMPILE=n`, then these units are compiled on their first use. For example, packages that are used to build domain indexes are compiled when the domain indexes are created.

Related Topics

- Importing Stored Procedures, Functions, and Packages
  The behavior of Import when a local stored procedure, function, or package is imported depends upon whether the `COMPILE` parameter is set to `y` or to`n`.

28.8.4 CONSTRAINTS

The `CONSTRAINTS` Import parameter specifies whether table constraints are imported.

Default

`y`

Purpose

Specifies whether table constraints are imported. The default is to import constraints (`y`). If you do not want constraints to be imported, then you must set the parameter value to `n`.

Note that primary key constraints for index-organized tables (IOTs) and object tables are always imported.

28.8.5 DATA_ONLY

The DATA_ONLY Import parameter imports only data from a dump file.

Default

n

Purpose

To import only data (no metadata) from a dump file, specify DATA_ONLY=y.

When you specify DATA_ONLY=y, any import parameters related to metadata that are entered on the command line (or in a parameter file) become invalid. This means that no metadata from the dump file will be imported.

The metadata-related parameters are the following: COMPILE, CONSTRAINTS, DATAFILES, DESTROY, GRANTS, IGNORE, INDEXES, INDEXFILE, ROWS=n, SHOW, SKIP_UNUSABLE_INDEXES, STATISTICS, STREAMS_CONFIGURATION, STREAMS_INSTANTIATION, TABLESPACES, TOID_NOVALIDATE, TRANSPORT_TABLESPACE, TTS_OWNERS.

28.8.6 DATAFILES

The DATAFILES Import parameter lists the data files that you want to transport into the database.

Default

None.

Purpose

When TRANSPORT_TABLESPACE is specified as y, use this parameter to list the data files that you want to be transported into the database.

Related Topics

- TRANSPORT_TABLESPACE
  When specified as y, instructs Import to import transportable tablespace metadata from an export file.

28.8.7 DESTROY

The DESTROY Import parameter specifies whether the existing data files making up the database should be reused.

Default

n

Specifies whether the existing data files making up the database should be reused. That is, specifying DESTROY=y causes Import to include the REUSE option in the data file clause of the SQL CREATE TABLESPACE statement, which causes Import to reuse the original database's data files after deleting their contents.

Note that the export file contains the data file names used in each tablespace. If you specify `DESTROY=y` and attempt to create a second database on the same system (for testing or other purposes), then the Import utility will overwrite the first database's data files when it creates the tablespace. In this situation you should use the default, `DESTROY=n`, so that an error occurs if the data files already exist when the tablespace is created. Also, when you need to import into the original database, you will need to specify `IGNORE=y` to add to the existing data files without replacing them.

> **Note:**
>
> If data files are stored on a raw device, then `DESTROY=n` *does not prevent* files from being overwritten.

28.8.8 FEEDBACK

The `FEEDBACK` Import utility parameter specifies that Import should display a progress meter in the form of a period for *n* number of rows imported.

Default: `0` (zero)

Specifies that Import should display a progress meter in the form of a period for *n* number of rows imported. For example, if you specify `FEEDBACK=10`, then Import displays a period each time 10 rows have been imported. The `FEEDBACK` value applies to all tables being imported; it cannot be individually set for each table.

28.8.9 FILE

The `FILE` Import utility parameter specifies the names of the export files to import.

Default: `expdat.dmp`

Description

Specifies the names of the export files to import. The default extension is `.dmp`. Because Export supports multiple export files, it can be necessary to specify multiple file names that you want to be imported.

You do not need to be the user that exported the export files. However, you must have read access to the files. If you did not export the files under your user ID, then you must also have the `IMP_FULL_DATABASE` role granted to you.

Example

```
imp scott IGNORE=y FILE = dat1.dmp, dat2.dmp, dat3.dmp FILESIZE=2048
```

28.8.10 FILESIZE

The `FILESIZE` Import utility parameter lets you specify the same maximum dump file size that you specified on export.

Default: operating system-dependent

Lets you specify the same maximum dump file size that you specified on export.

> **Note:**
>
> The maximum size allowed is operating system-dependent. You should verify this maximum value in your Oracle operating system-specific documentation before specifying `FILESIZE`.

The `FILESIZE` value can be specified as a number followed by KB (number of kilobytes). For example, `FILESIZE=2KB` is the same as `FILESIZE=2048`. Similarly, MB specifies megabytes (1024 * 1024) and GB specifies gigabytes (1024**3). B remains the shorthand for bytes; the number is not multiplied to obtain the final file size (`FILESIZE=2048B` is the same as `FILESIZE=2048`).

28.8.11 FROMUSER

The `FROMUSER` parameter of the Import utility enables you to import a subset of schemas from an export file containing multiple schemas.

Default: none

A comma-delimited list of schemas to import. This parameter is relevant only to users with the `IMP_FULL_DATABASE` role. The parameter enables you to import a subset of schemas from an export file containing multiple schemas (for example, a full export dump file or a multischema, user-mode export dump file).

Schema names that appear inside function-based indexes, functions, procedures, triggers, type bodies, views, and so on, are *not* affected by `FROMUSER` or `TOUSER` processing. Only the *name* of the object is affected. After the import has completed, items in any `TOUSER` schema should be manually checked for references to old (`FROMUSER`) schemas, and corrected if necessary.

You will typically use `FROMUSER` in conjunction with the Import parameter `TOUSER`, which you use to specify a list of usernames whose schemas will be targets for import. The user that you specify with `TOUSER` must exist in the target database before the import operation; otherwise an error is returned.

If you do not specify `TOUSER`, then Import will do the following:

- Import objects into the `FROMUSER` schema if the export file is a full dump or a multischema, user-mode export dump file
- Create objects in the importer's schema (regardless of the presence of or absence of the `FROMUSER` schema on import) if the export file is a single-schema, user-mode export dump file created by an unprivileged user

> **Note:**
>
> Specifying `FROMUSER=SYSTEM` causes only schema objects belonging to user `SYSTEM` to be imported; it does not cause system objects to be imported.

28.8.12 FULL

The `FULL` Import utility parameter specifies whether to import the entire export dump file.

Default: `y`

Specifies whether to import the entire export dump file.

- Points to Consider for Full Database Exports and Imports
  A full database export and import can be a good way to replicate or clean up a database.

28.8.12.1 Points to Consider for Full Database Exports and Imports

A full database export and import can be a good way to replicate or clean up a database.

However, to avoid problems be sure to keep the following points in mind:

- A full export does not export triggers owned by schema `SYS`. You must manually re-create `SYS` triggers either before or after the full import. Oracle recommends that you re-create them after the import in case they define actions that would impede progress of the import.
- A full export also does not export the default profile. If you have modified the default profile in the source database (for example, by adding a password verification function owned by schema `SYS`), then you must manually pre-create the function and modify the default profile in the target database after the import completes.
- If possible, before beginning, make a physical copy of the exported database and the database into which you intend to import. This ensures that any mistakes are reversible.
- Before you begin the export, it is advisable to produce a report that includes the following information:
  - A list of tablespaces and data files
  - A list of rollback segments
  - A count, by user, of each object type such as tables, indexes, and so on

  This information lets you ensure that tablespaces have already been created and that the import was successful.
- If you are creating a completely new database from an export, then remember to create an extra rollback segment in `SYSTEM` and to make it available in your initialization parameter file (`init.ora`) before proceeding with the import.
- When you perform the import, ensure you are pointing at the correct instance. This is very important because on some UNIX systems, just the act of entering a subshell can change the database against which an import operation was performed.
- Do not perform a full import on a system that has more than one database unless you are certain that all tablespaces have already been created. A full import creates any undefined tablespaces using the same data file names as the exported database. This can result in problems in the following situations:
  - If the data files belong to any other database, then they will become corrupted. This is especially true if the exported database is on the same system, because its data files will be reused by the database into which you are importing.
  - If the data files have names that conflict with existing operating system files.

28.8.13 GRANTS

Specifies whether to import object grants.

Default: `y`

By default, the Import utility imports any object grants that were exported. If the export was a user-mode export, then the export file contains only first-level object grants (those granted by the owner).

If the export was a full database mode export, then the export file contains all object grants, including lower-level grants (those granted by users given a privilege with the `WITH GRANT OPTION`). If you specify `GRANTS=n`, then the Import utility does not import object grants. (Note that system grants *are* imported even if `GRANTS=n`.)

> **Note:**
>
> Export does not export grants on data dictionary views for security reasons that affect Import. If such grants were exported, then access privileges would be changed and the importer would not be aware of this.

28.8.14 HELP

The `HELP` parameter of Import utility displays a description of the Import parameters.

Default: none

Displays a description of the Import parameters. Enter `imp HELP=y` on the command line to display the help content.

28.8.15 IGNORE

The IGNORE Import utility parameter specifies how object creation errors should be handled.

Default: `n`

Specifies how object creation errors should be handled. If you accept the default, `IGNORE=n`, then Import logs or displays object creation errors before continuing.

If you specify `IGNORE=y`, then Import overlooks object creation errors when it attempts to create database objects, and continues without reporting the errors.

Note that only *object creation errors* are ignored; other errors, such as operating system, database, and SQL errors, *are not* ignored and may cause processing to stop.

In situations where multiple refreshes from a single export file are done with `IGNORE=y`, certain objects can be created multiple times (although they will have unique system-defined names). You can prevent this for certain objects (for example, constraints) by doing an import with `CONSTRAINTS=n`. If you do a full import with `CONSTRAINTS=n`, then no constraints for any tables are imported.

If a table already exists and `IGNORE=y`, then rows are imported into existing tables without any errors or messages being given. You might want to import data into tables that already exist in order to use new storage parameters or because you have already created the table in a cluster.

If a table already exists and `IGNORE=n`, then errors are reported and the table is skipped with no rows inserted. Also, objects dependent on tables, such as indexes, grants, and constraints, will not be created.

> **Note:**
>
> When you import into existing tables, if no column in the table is uniquely indexed, rows could be duplicated.

28.8.16 INDEXES

Indexes import parameter specifies whether to import indexes.

Default: `y`

Specifies whether to import indexes. System-generated indexes such as LOB indexes, OID indexes, or unique constraint indexes are re-created by Import regardless of the setting of this parameter.

You can postpone all user-generated index creation until after Import completes, by specifying `INDEXES=n`.

If indexes for the target table already exist at the time of the import, then Import performs index maintenance when data is inserted into the table.

28.8.17 INDEXFILE

INDEXFILE parameter of Import utility specifies a file to receive index-creation statements.

Default: none

Specifies a file to receive index-creation statements.

When this parameter is specified, index-creation statements for the requested mode are extracted and written to the specified file, rather than used to create indexes in the database. No database objects are imported.

If the Import parameter `CONSTRAINTS` is set to `y`, then Import also writes table constraints to the index file.

The file can then be edited (for example, to change storage parameters) and used as a SQL script to create the indexes.

To make it easier to identify the indexes defined in the file, the export file's `CREATE TABLE` statements and `CREATE CLUSTER` statements are included as comments.

Perform the following steps to use this feature:

1. Import using the `INDEXFILE` parameter to create a file of index-creation statements.
2. Edit the file, making certain to add a valid password to the `connect` strings.
3. Rerun Import, specifying `INDEXES=n`.

   (This step imports the database objects while preventing Import from using the index definitions stored in the export file.)
4. Execute the file of index-creation statements as a SQL script to create the index.

The INDEXFILE parameter can be used only with the FULL=y, FROMUSER, TOUSER, or TABLES parameters.

28.8.18 LOG

Specifies a file (for example, import.log) to receive informational and error messages.

Default: none

If you specify a log file, then the Import utility writes all information to the log in addition to the terminal display.

28.8.19 PARFILE

Specifies a file name for a file that contains a list of Import parameters.

Default: none

For more information about using a parameter file, see Parameter Files.

28.8.20 RECORDLENGTH

Specifies the length, in bytes, of the file record.

Default

Operating system-dependent.

Purpose

The RECORDLENGTH parameter is necessary when you must transfer the export file to another operating system that uses a different default value.

If you do not define this parameter, then it defaults to your platform-dependent value for BUFSIZ.

You can set RECORDLENGTH to any value equal to or greater than your system's BUFSIZ. (The highest value is 64 KB.) Changing the RECORDLENGTH parameter affects only the size of data that accumulates before writing to the database. It does not affect the operating system file block size.

You can also use this parameter to specify the size of the Import I/O buffer.

28.8.21 RESUMABLE

The RESUMABLE parameter is used to enable and disable resumable space allocation.

Default

n

Purpose

Because this parameter is disabled by default, you must set RESUMABLE=y to use its associated parameters, RESUMABLE_NAME and RESUMABLE_TIMEOUT.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for more information about resumable space allocation.

28.8.22 RESUMABLE_NAME

The value for the `RESUMABLE_NAME` parameter identifies the statement that is resumable.

Default

```
'User USERNAME (USERID), Session SESSIONID, Instance INSTANCEID'
```

Purpose

This value is a user-defined text string that is inserted in either the `USER_RESUMABLE` or `DBA_RESUMABLE` view to help you identify a specific resumable statement that has been suspended.

This parameter is ignored unless the `RESUMABLE` parameter is set to `y` to enable resumable space allocation.

28.8.23 RESUMABLE_TIMEOUT

The value of the `RESUMABLE_TIMEOUT` parameter specifies the time period during which an error must be fixed.

Default

`7200` seconds (2 hours)

Purpose

If the error is not fixed within the timeout period, then execution of the statement is terminated.

This parameter is ignored unless the `RESUMABLE` parameter is set to `y` to enable resumable space allocation.

28.8.24 ROWS

Specifies whether to import the rows of table data.

Default

`y`

Purpose

If `ROWS=n`, then statistics for all imported tables will be locked after the import operation is finished.

28.8.25 SHOW

Lists the contents of the export file before importing.

Default

`n`

Syntax and Description

When `SHOW=y`, the contents of the export dump file are listed to the display and not imported. The SQL statements contained in the export are displayed in the order in which Import will execute them.

The `SHOW` parameter can be used only with the `FULL=y`, `FROMUSER`, `TOUSER`, or `TABLES` parameter.

28.8.26 SKIP_UNUSABLE_INDEXES

Both Import and the Oracle database provide a `SKIP_UNUSABLE_INDEXES` parameter.

Default: the value of the Oracle database configuration parameter, `SKIP_UNUSABLE_INDEXES`, as specified in the initialization parameter file.

The Import `SKIP_UNUSABLE_INDEXES` parameter is specified at the Import command line. The Oracle database `SKIP_UNUSABLE_INDEXES` parameter is specified as a configuration parameter in the initialization parameter file. It is important to understand how they affect each other.

If you do not specify a value for `SKIP_UNUSABLE_INDEXES` at the Import command line, then Import uses the database setting for the `SKIP_UNUSABLE_INDEXES` configuration parameter, as specified in the initialization parameter file.

If you do specify a value for `SKIP_UNUSABLE_INDEXES` at the Import command line, then it overrides the value of the `SKIP_UNUSABLE_INDEXES` configuration parameter in the initialization parameter file.

A value of `y` means that Import will skip building indexes that were set to the Index Unusable state (by either system or user). Other indexes (not previously set to Index Unusable) continue to be updated as rows are inserted.

This parameter enables you to postpone index maintenance on selected index partitions until after row data has been inserted. You then have the responsibility to rebuild the affected index partitions after the Import.

> **Note:**
>
> Indexes that are unique and marked Unusable are not allowed to skip index maintenance. Therefore, the `SKIP_UNUSABLE_INDEXES` parameter has no effect on unique indexes.

You can use the `INDEXFILE` parameter in conjunction with `INDEXES=n` to provide the SQL scripts for re-creating the index. If the `SKIP_UNUSABLE_INDEXES` parameter is not specified, then row insertions that attempt to update unusable indexes will fail.

> **See Also:**
>
> The `ALTER SESSION` statement in the *Oracle Database SQL Language Reference*

28.8.27 STATISTICS

Specifies what is done with the database optimizer statistics at import time.

Default: `ALWAYS`

The options are:

- `ALWAYS`

  Always import database optimizer statistics regardless of whether they are questionable.

- `NONE`

  Do not import or recalculate the database optimizer statistics.

- `SAFE`

  Import database optimizer statistics only if they are not questionable. If they are questionable, then recalculate the optimizer statistics.

- `RECALCULATE`

  Do not import the database optimizer statistics. Instead, recalculate them on import. This requires that the original export operation that created the dump file must have generated the necessary `ANALYZE` statements (that is, the export was not performed with `STATISTICS=NONE`). These `ANALYZE` statements are included in the dump file and used by the import operation for recalculation of the table's statistics.

> **See Also:**
>
> – *Oracle Database Concepts* for more information about the optimizer and the statistics it uses
>
> – Importing Statistics

28.8.28 STREAMS_CONFIGURATION

Specifies whether to import any general GoldenGate Replication metadata that may be present in the export dump file.

Default: `y`

28.8.29 STREAMS_INSTANTIATION

Specifies whether to import Streams instantiation metadata that may be present in the export dump file.

Default: `n`

Specify `y` if the import is part of an instantiation in a Streams environment.

28.8.30 TABLES

Default: none

Specifies that the import is a table-mode import and lists the table names and partition and subpartition names to import. Table-mode import lets you import entire partitioned or nonpartitioned tables. The `TABLES` parameter restricts the import to the specified tables and their associated objects, as listed in Import Modes. You can specify the following values for the `TABLES` parameter:

- *`tablename`* specifies the name of the table or tables to be imported. If a table in the list is partitioned and you do not specify a partition name, then all its partitions and subpartitions are imported. To import all the exported tables, specify an asterisk (*) as the only table name parameter.

  `tablename` can contain any number of '%' pattern matching characters, which can each match zero or more characters in the table names in the export file. All the tables whose names match all the specified patterns of a specific table name in the list are selected for import. A table name in the list that consists of all pattern matching characters and no partition name results in all exported tables being imported.

- *`partition_name`* and *`subpartition_name`* let you restrict the import to one or more specified partitions or subpartitions within a partitioned table.

The syntax you use to specify the preceding is in the form:

```
tablename:partition_name

tablename:subpartition_name
```

If you use *`tablename`*`:`*`partition_name`*, then the specified table must be partitioned, and *`partition_name`* must be the name of one of its partitions or subpartitions. If the specified table is not partitioned, then the *`partition_name`* is ignored and the entire table is imported.

The number of tables that can be specified at the same time is dependent on command-line limits.

As the export file is processed, each table name in the export file is compared against each table name in the list, in the order in which the table names were specified in the parameter. To avoid ambiguity and excessive processing time, specific table names should appear at the beginning of the list, and more general table names (those with patterns) should appear at the end of the list.

Although you can qualify table names with schema names (as in `scott.emp`) when exporting, you *cannot* do so when importing. In the following example, the `TABLES` parameter is specified incorrectly:

```
imp TABLES=(jones.accts, scott.emp, scott.dept)
```

The valid specification to import these tables is as follows:

```
imp FROMUSER=jones TABLES=(accts)
imp FROMUSER=scott TABLES=(emp,dept)
```

For a more detailed example, see "Example Import Using Pattern Matching to Import Various Tables".

> **Note:**
>
> Some operating systems, such as UNIX, require that you use escape characters before special characters, such as a parenthesis, so that the character is not treated as a special character. On UNIX, use a backslash (\) as the escape character, as shown in the following example:
>
> ```
> TABLES=\(emp,dept\)
> ```

- Table Name Restrictions
  This is an explanation of table name restrictions for Import utility.

28.8.30.1 Table Name Restrictions

This is an explanation of table name restrictions for Import utility.

The following restrictions apply to table names:

- By default, table names in a database are stored as uppercase. If you have a table name in mixed-case or lowercase, and you want to preserve case-sensitivity for the table name, then you must enclose the name in quotation marks. The name must exactly match the table name stored in the database.

  Some operating systems require that quotation marks on the command line be preceded by an escape character. The following are examples of how case-sensitivity can be preserved in the different Import modes.

  - In command-line mode:

    ```
    tables='\"Emp\"'
    ```

  - In interactive mode:

    ```
    Table(T) to be exported: "Exp"
    ```

  - In parameter file mode:

    ```
    tables='"Emp"'
    ```

- Table names specified on the command line cannot include a pound (#) sign, unless the table name is enclosed in quotation marks. Similarly, in the parameter file, if a table name includes a pound (#) sign, then the Import utility interprets the rest of the line as a comment, unless the table name is enclosed in quotation marks.

  For example, if the parameter file contains the following line, then Import interprets everything on the line after `emp#` as a comment and does not import the tables `dept` and `mydata:`

  ```
  TABLES=(emp#, dept, mydata)
  ```

  However, given the following line, the Import utility imports all three tables because `emp#` is enclosed in quotation marks:

  ```
  TABLES=("emp#", dept, mydata)
  ```

> **Note:**
>
> Some operating systems require single quotation marks rather than double quotation marks, or the reverse; see your Oracle operating system-specific documentation. Different operating systems also have other restrictions on table naming.
>
> For example, the UNIX C shell attaches a special meaning to a dollar sign ($) or pound sign (#) (or certain other special characters). You must use escape characters to get such characters in the name past the shell and into Import.

28.8.31 TABLESPACES

The `TABLESPACES` parameter for the Import utility.

Default: none

When `TRANSPORT_TABLESPACE` is specified as `y`, use this parameter to list the tablespaces to be transported into the database. If there is more than one tablespace in the export file, then you must specify all of them as part of the import operation.

See TRANSPORT_TABLESPACE for more information.

28.8.32 TOID_NOVALIDATE

Use the `TOID_NOVALIDATE` parameter to specify types to exclude from TOID comparison.

Default: none

When you import a table that references a type, but a type of that name already exists in the database, Import attempts to verify that the preexisting type is, in fact, the type used by the table (rather than a different type that just happens to have the same name).

To do this, Import compares the type's unique identifier (TOID) with the identifier stored in the export file. Import will not import the table rows if the TOIDs do not match.

In some situations, you may not want this validation to occur on specified types (for example, if the types were created by a cartridge installation). You can use the `TOID_NOVALIDATE` parameter to specify types to exclude from TOID comparison.

The syntax is as follows:

```
TOID_NOVALIDATE=([schemaname.]typename [, ...])
```

For example:

```
imp scott TABLES=jobs TOID_NOVALIDATE=typ1
imp scott TABLES=salaries TOID_NOVALIDATE=(fred.typ0,sally.typ2,typ3)
```

If you do not specify a schema name for the type, then it defaults to the schema of the importing user. For example, in the first preceding example, the type `typ1` defaults to `scott.typ1` and in the second example, the type `typ3` defaults to `scott.typ3`.

Note that `TOID_NOVALIDATE` deals only with table column types. It has no effect on table types.

The output of a typical import with excluded types would contain entries similar to the following:

```
[...]
. importing IMP3's objects into IMP3
. . skipping TOID validation on type IMP2.TOIDTYP0
. . importing table                  "TOIDTAB3"
[...]
```

> **Note:**
>
> When you inhibit validation of the type identifier, it is your responsibility to ensure that the attribute list of the imported type matches the attribute list of the existing type. If these attribute lists do not match, then results are unpredictable.

28.8.33 TOUSER

Specifies a list of user names whose schemas will be targets for Import.

Default: none

The user names must exist before the import operation; otherwise an error is returned. The `IMP_FULL_DATABASE` role is required to use this parameter. To import to a different schema than the one that originally contained the object, specify `TOUSER`. For example:

```
imp FROMUSER=scott TOUSER=joe TABLES=emp
```

If multiple schemas are specified, then the schema names are paired. The following example imports `scott`'s objects into `joe`'s schema, and `fred`'s objects into `ted`'s schema:

```
imp FROMUSER=scott,fred TOUSER=joe,ted
```

If the `FROMUSER` list is longer than the `TOUSER` list, then the remaining schemas will be imported into either the `FROMUSER` schema, or into the importer's schema, based on normal defaulting rules. You can use the following syntax to ensure that any extra objects go into the `TOUSER` schema:

```
imp FROMUSER=scott,adams TOUSER=ted,ted
```

Note that user `ted` is listed twice.

> **See Also:**
>
> FROMUSER for information about restrictions when using `FROMUSER` and `TOUSER`

28.8.34 TRANSPORT_TABLESPACE

When specified as `y`, instructs Import to import transportable tablespace metadata from an export file.

Default: `n`

Encrypted columns are not supported in transportable tablespace mode.

> **Note:**
>
> You cannot export transportable tablespaces and then import them into a database at a lower release level. The target database must be at the same or later release level as the source database.

28.8.35 TTS_OWNERS

When `TRANSPORT_TABLESPACE` is specified as `y`, use this parameter to list the users who own the data in the transportable tablespace set.

Default: none

See TRANSPORT_TABLESPACE.

28.8.36 USERID (username/password)

Specifies the username, password, and an optional connect string of the user performing the import.

Default: none

If you connect as user `SYS`, then you must also specify `AS SYSDBA` in the connect string. Your operating system may require you to treat `AS SYSDBA` as a special string, in which case the entire string would be enclosed in quotation marks.

> **See Also:**
>
> The user's guide for your Oracle Net protocol for information about specifying a connect string for Oracle Net.

28.8.37 VOLSIZE

Specifies the maximum number of bytes in a dump file on each volume of tape.

Default: none

The `VOLSIZE` parameter has a maximum value equal to the maximum value that can be stored in 64 bits on your platform.

The `VOLSIZE` value can be specified as number followed by KB (number of kilobytes). For example, `VOLSIZE=2KB` is the same as `VOLSIZE=2048`. Similarly, MB specifies megabytes (1024

* 1024) and GB specifies gigabytes (1024**3). The shorthand for bytes remains B; the number is not multiplied to get the final file size (`VOLSIZE=2048B` is the same as `VOLSIZE=2048`).

28.9 Example Import Sessions

These sections give some examples of import sessions that show you how to use the parameter file and command-line methods.

- Example Import of Selected Tables for a Specific User
- Example Import of Tables Exported by Another User
- Example Import of Tables from One User to Another
- Example Import Session Using Partition-Level Import
- Example Import Using Pattern Matching to Import Various Tables

28.9.1 Example Import of Selected Tables for a Specific User

In this example, using a full database export file, an administrator imports the `dept` and `emp` tables into the `scott` schema.

Parameter File Method

```
> imp PARFILE=params.dat
```

The `params.dat` file contains the following information:

```
FILE=dba.dmp
SHOW=n
IGNORE=n
GRANTS=y
FROMUSER=scott
TABLES=(dept,emp)
```

Command-Line Method

```
> imp FILE=dba.dmp FROMUSER=scott TABLES=(dept,emp)
```

Import Messages

Information is displayed about the release of Import you are using and the release of Oracle Database that you are connected to. Status messages are also displayed.

28.9.2 Example Import of Tables Exported by Another User

This example illustrates importing the `unit` and `manager` tables from a file exported by `blake` into the `scott` schema.

Parameter File Method

```
> imp PARFILE=params.dat
```

The `params.dat` file contains the following information:

```
FILE=blake.dmp
SHOW=n
IGNORE=n
GRANTS=y
```

```
ROWS=y
FROMUSER=blake
TOUSER=scott
TABLES=(unit,manager)
```

Command-Line Method

```
> imp FROMUSER=blake TOUSER=scott FILE=blake.dmp TABLES=(unit,manager)
```

Import Messages

Information is displayed about the release of Import you are using and the release of Oracle Database that you are connected to. Status messages are also displayed.

28.9.3 Example Import of Tables from One User to Another

In this example, a database administrator (DBA) imports all tables belonging to `scott` into user `blake`'s account.

Parameter File Method

```
 > imp PARFILE=params.dat
```

The `params.dat` file contains the following information:

```
FILE=scott.dmp
FROMUSER=scott
TOUSER=blake
TABLES=(*)
```

Command-Line Method

```
> imp FILE=scott.dmp FROMUSER=scott TOUSER=blake TABLES=(*)
```

Import Messages

Information is displayed about the release of Import you are using and the release of Oracle Database that you are connected to. Then, status messages similar to the following are shown:

```
.
.
.
Warning: the objects were exported by SCOTT, not by you

import done in WE8DEC character set and AL16UTF16 NCHAR character set
. importing SCOTT's objects into BLAKE
. . importing table                          "BONUS"          0 rows imported
. . importing table                           "DEPT"          4 rows imported
. . importing table                            "EMP"         14 rows imported
. . importing table                        "SALGRADE"          5 rows imported
Import terminated successfully without warnings.
```

28.9.4 Example Import Session Using Partition-Level Import

This section describes an import of a table with multiple partitions, a table with partitions and subpartitions, and repartitioning a table on different columns.

- Example 1: A Partition-Level Import
- Example 2: A Partition-Level Import of a Composite Partitioned Table

- Example 3: Repartitioning a Table on a Different Column

28.9.4.1 Example 1: A Partition-Level Import

In this example, `emp` is a partitioned table with three partitions: `P1`, `P2`, and `P3`.

A table-level export file was created using the following command:

```
> exp scott TABLES=emp FILE=exmpexp.dat ROWS=y
```

Export Messages

Information is displayed about the release of Export you are using and the release of Oracle Database that you are connected to. Then, status messages similar to the following are shown:

```
.
.
.
About to export specified tables via Conventional Path ...
. . exporting table                            EMP
. . exporting partition                             P1          7 rows exported
. . exporting partition                             P2         12 rows exported
. . exporting partition                             P3          3 rows exported
Export terminated successfully without warnings.
```

In a partition-level Import you can specify the specific partitions of an exported table that you want to import. In this example, these are `P1` and `P3` of table `emp`:

```
> imp scott TABLES=(emp:p1,emp:p3) FILE=exmpexp.dat ROWS=y
```

Import Messages

Information is displayed about the release of Import you are using and the release of Oracle Database that you are connected to. Status messages are also displayed.

28.9.4.2 Example 2: A Partition-Level Import of a Composite Partitioned Table

This example demonstrates that the partitions and subpartitions of a composite partitioned table are imported. `emp` is a partitioned table with two composite partitions: `P1` and `P2`. Partition `P1` has three subpartitions: `P1_SP1`, `P1_SP2`, and `P1_SP3`. Partition `P2` has two subpartitions: `P2_SP1` and `P2_SP2`.

A table-level export file was created using the following command:

```
> exp scott TABLES=emp FILE=exmpexp.dat ROWS=y
```

Export Messages

Information is displayed about the release of Export you are using and the release of Oracle Database that you are connected to. Then, status messages similar to the following are shown:

When the command executes, the following Export messages are displayed:

```
.
.
.
About to export specified tables via Conventional Path ...
. . exporting table                            EMP
. . exporting composite partition                   P1
. . exporting subpartition                       P1_SP1          2 rows exported
. . exporting subpartition                       P1_SP2         10 rows exported
```

```
. . exporting subpartition                     P1_SP3          7 rows exported
. . exporting composite partition                 P2
. . exporting subpartition                     P2_SP1          4 rows exported
. . exporting subpartition                     P2_SP2          2 rows exported
Export terminated successfully without warnings.
```

The following Import command results in the importing of subpartition `P1_SP2` and `P1_SP3` of composite partition `P1` in table `emp` and all subpartitions of composite partition `P2` in table `emp`.

```
> imp scott TABLES=(emp:p1_sp2,emp:p1_sp3,emp:p2) FILE=exmpexp.dat ROWS=y
```

Import Messages

Information is displayed about the release of Import you are using and the release of Oracle Database that you are connected to. Then, status messages similar to the following are shown:

```
.
.
.
. importing SCOTT's objects into SCOTT
. . importing subpartition              "EMP":"P1_SP2"         10 rows imported
. . importing subpartition              "EMP":"P1_SP3"          7 rows imported
. . importing subpartition              "EMP":"P2_SP1"          4 rows imported
. . importing subpartition              "EMP":"P2_SP2"          2 rows imported
Import terminated successfully without warnings.
```

28.9.4.3 Example 3: Repartitioning a Table on a Different Column

This example assumes the `emp` table has two partitions based on the `empno` column. This example repartitions the `emp` table on the `deptno` column.

Perform the following steps to repartition a table on a different column:

1. Export the table to save the data.
2. Drop the table from the database.
3. Create the table again with the new partitions.
4. Import the table data.

The following example illustrates these steps.

```
> exp scott table=emp file=empexp.dat
.
.
.

About to export specified tables via Conventional Path ...
. . exporting table                            EMP
. . exporting partition                        EMP_LOW          4 rows exported
. . exporting partition                       EMP_HIGH         10 rows exported
Export terminated successfully without warnings.

SQL> connect scott
Connected.
SQL> drop table emp cascade constraints;
Statement processed.
SQL> create table emp
  2    (
  3    empno    number(4) not null,
  4    ename    varchar2(10),
  5    job      varchar2(9),
```

```
  6    mgr      number(4),
  7    hiredate date,
  8    sal      number(7,2),
  9    comm     number(7,2),
 10    deptno   number(2)
 11    )
 12 partition by range (deptno)
 13   (
 14   partition dept_low values less than (15)
 15     tablespace tbs_1,
 16   partition dept_mid values less than (25)
 17     tablespace tbs_2,
 18   partition dept_high values less than (35)
 19     tablespace tbs_3
 20   );
Statement processed.
SQL> exit

> imp scott tables=emp file=empexp.dat ignore=y
.
.
.
import done in WE8DEC character set and AL16UTF16 NCHAR character set
. importing SCOTT's objects into SCOTT
. . importing partition                "EMP":"EMP_LOW"          4 rows imported
. . importing partition               "EMP":"EMP_HIGH"         10 rows imported
Import terminated successfully without warnings.
```

The following SQL `SELECT` statements show that the data is partitioned on the `deptno` column:

```
SQL> connect scott
Connected.
SQL> select empno, deptno from emp partition (dept_low);
EMPNO      DEPTNO
---------- ----------
      7782         10
      7839         10
      7934         10
3 rows selected.
SQL> select empno, deptno from emp partition (dept_mid);
EMPNO      DEPTNO
---------- ----------
      7369         20
      7566         20
      7788         20
      7876         20
      7902         20
5 rows selected.
SQL> select empno, deptno from emp partition (dept_high);
EMPNO      DEPTNO
---------- ----------
      7499         30
      7521         30
      7654         30
      7698         30
      7844         30
      7900         30
6 rows selected.
SQL> exit;
```

28.9.5 Example Import Using Pattern Matching to Import Various Tables

In this example, pattern matching is used to import various tables for user `scott`.

Parameter File Method

```
imp PARFILE=params.dat
```

The `params.dat` file contains the following information:

```
FILE=scott.dmp
IGNORE=n
GRANTS=y
ROWS=y
FROMUSER=scott
TABLES=(%d%,b%s)
```

Command-Line Method

```
imp FROMUSER=scott FILE=scott.dmp TABLES=(%d%,b%s)
```

Import Messages

Information is displayed about the release of Import you are using and the release of Oracle Database that you are connected to. Then, status messages similar to the following are shown:

```
.
.
.
import done in US7ASCII character set and AL16UTF16 NCHAR character set
import server uses JA16SJIS character set (possible charset conversion)
. importing SCOTT's objects into SCOTT
. . importing table                  "BONUS"          0 rows imported
. . importing table                   "DEPT"          4 rows imported
. . importing table                "SALGRADE"         5 rows imported
Import terminated successfully without warnings.
```

28.10 Exit Codes for Inspection and Display

Import provides the results of an operation immediately upon completion. Depending on the platform, the outcome may be reported in a process exit code and the results recorded in the log file.

Import Process Exit Codes

Reporting the result in a process exit code enables you to check the outcome from the command line or script. The following table shows the exit codes that are returned for various results.

Table 28-4 Exit Codes for Original Import

Result	Exit Code
Import terminated successfully without warnings	`EX_SUCC`
Import terminated successfully with warnings	`EX_OKWARN`
Import terminated unsuccessfully	`EX_FAIL`

Example 28-1 Log file Exit Code Output

For Unix and Linux platforms, the exit codes are as follows:

```
EX_SUCC   0
EX_OKWARN 0
EX_FAIL   1
```

28.11 Error Handling During an Import

These sections describe errors that can occur when you import database objects.

- Row Errors
  If a row is rejected due to an integrity constraint violation or invalid data, then Import displays a warning message but continues processing the rest of the table.
- Errors Importing Database Objects
  Errors can occur for many reasons when you import database objects, as described in these sections.

28.11.1 Row Errors

If a row is rejected due to an integrity constraint violation or invalid data, then Import displays a warning message but continues processing the rest of the table.

Some errors, such as "tablespace full," apply to all subsequent rows in the table. These errors cause Import to stop processing the current table and skip to the next table.

A "tablespace full" error can suspend the import if the `RESUMABLE=y` parameter is specified.

- Failed Integrity Constraints
  A row error is generated if a row violates one of the integrity constraints in force on your system.
- Invalid Data
  Row errors can also occur when the column definition for a table in a database is different from the column definition in the export file.

28.11.1.1 Failed Integrity Constraints

A row error is generated if a row violates one of the integrity constraints in force on your system.

Including:

- `NOT NULL` constraints
- Uniqueness constraints
- Primary key (not null and unique) constraints
- Referential integrity constraints
- Check constraints

> **See Also:**
>
> – *Oracle Database Development Guide* for information about using integrity constraints in applications
> – *Oracle Database Concepts* for more information about integrity constraints

28.11.1.2 Invalid Data

Row errors can also occur when the column definition for a table in a database is different from the column definition in the export file.

The error is caused by data that is too long to fit into a new table's columns, by invalid data types, or by any other `INSERT` error.

> **Note:**
>
> All forms of `LONG` data types (`LONG`, `LONG RAW`, `LONG VARCHAR`, `LONG VARRAW`) were deprecated in Oracle8i Release 8.1.6. For succeeding releases, the `LONG` data type was provided for backward compatibility with existing applications. In new applications developed with later releases, Oracle strongly recommends that you use `CLOB` and `NCLOB` data types for large amounts of character data.

28.11.2 Errors Importing Database Objects

Errors can occur for many reasons when you import database objects, as described in these sections.

When these errors occur, import of the current database object is discontinued. Import then attempts to continue with the next database object in the export file.

- Object Already Exists
  If a database object to be imported already exists in the database, then an object creation error occurs.
- Sequences
  If sequence numbers need to be reset to the value in an export file as part of an import, then you should drop sequences.
- Resource Errors
  Resource limitations can cause objects to be skipped. When you are importing tables, for example, resource errors can occur because of internal problems or when a resource such as memory has been exhausted.
- Domain Index Metadata
  Domain indexes can have associated application-specific metadata that is imported using anonymous PL/SQL blocks.

28.11.2.1 Object Already Exists

If a database object to be imported already exists in the database, then an object creation error occurs.

What happens next depends on the setting of the `IGNORE` parameter.

If `IGNORE=n` (the default), then the error is reported, and Import continues with the next database object. The current database object is not replaced. For tables, this behavior means that rows contained in the export file are not imported.

If `IGNORE=y`, then object creation errors are not reported. The database object is not replaced. If the object is a table, then rows are imported into it. Note that only *object creation errors* are ignored; all other errors (such as operating system, database, and SQL errors) *are* reported and processing may stop.

> **Note:**
>
> Specifying `IGNORE=y` can cause duplicate rows to be entered into a table unless one or more columns of the table are specified with the `UNIQUE` integrity constraint. This could occur, for example, if Import were run twice.

28.11.2.2 Sequences

If sequence numbers need to be reset to the value in an export file as part of an import, then you should drop sequences.

If a sequence is not dropped before the import, then it is not set to the value captured in the export file, because Import does not drop and re-create a sequence that already exists. If the sequence already exists, then the export file's `CREATE SEQUENCE` statement fails and the sequence is not imported.

28.11.2.3 Resource Errors

Resource limitations can cause objects to be skipped. When you are importing tables, for example, resource errors can occur because of internal problems or when a resource such as memory has been exhausted.

If a resource error occurs while you are importing a row, then Import stops processing the current table and skips to the next table. If you have specified `COMMIT=y`, then Import commits the partial import of the current table. If not, then a rollback of the current table occurs before Import continues. See the description of COMMIT.

28.11.2.4 Domain Index Metadata

Domain indexes can have associated application-specific metadata that is imported using anonymous PL/SQL blocks.

These PL/SQL blocks are executed at import time, before the `CREATE INDEX` statement. If a PL/SQL block causes an error, then the associated index is not created because the metadata is considered an integral part of the index.

28.12 Table-Level and Partition-Level Import

You can import tables, partitions, and subpartitions.

Specifically:

- Table-level Import: Imports all data from the specified tables in an export file.

- Partition-level Import: Imports only data from the specified source partitions or subpartitions.
- Guidelines for Using Table-Level Import
  For each specified table, table-level Import imports all rows of the table.
- Guidelines for Using Partition-Level Import
  Partition-level Import can only be specified in table mode. It lets you selectively load data from specified partitions or subpartitions in an export file.
- Migrating Data Across Partitions and Tables
  If you specify a partition name for a composite partition, then all subpartitions within the composite partition are used as the source.

28.12.1 Guidelines for Using Table-Level Import

For each specified table, table-level Import imports all rows of the table.

With table-level Import:

- All tables exported using any Export mode (except `TRANSPORT_TABLESPACES`) can be imported.
- Users can import the entire (partitioned or nonpartitioned) table, partitions, or subpartitions from a table-level export file into a (partitioned or nonpartitioned) target table with the same name.

If the table does not exist, and if the exported table was partitioned, then table-level Import creates a partitioned table. If the table creation is successful, then table-level Import reads all source data from the export file into the target table. After Import, the target table contains the partition definitions of *all* partitions and subpartitions associated with the source table in the export file. This operation ensures that the physical and logical attributes (including partition bounds) of the source partitions are maintained on import.

28.12.2 Guidelines for Using Partition-Level Import

Partition-level Import can only be specified in table mode. It lets you selectively load data from specified partitions or subpartitions in an export file.

Keep the following guidelines in mind when using partition-level Import.

- Import always stores the rows according to the partitioning scheme of the target table.
- Partition-level Import inserts only the row data from the specified source partitions or subpartitions.
- If the target table is partitioned, then partition-level Import rejects any rows that fall above the highest partition of the target table.
- Partition-level Import cannot import a nonpartitioned exported table. However, a partitioned table can be imported from a nonpartitioned exported table using table-level Import.
- Partition-level Import is legal only if the source table (that is, the table called tablename at export time) was partitioned and exists in the export file.
- If the partition or subpartition name is not a valid partition in the export file, then Import generates a warning.
- The partition or subpartition name in the parameter refers to only the partition or subpartition in the export file, which may not contain all of the data of the table on the export source system.

- If `ROWS=y` (default), and the table does not exist in the import target system, then the table is created and all rows from the source partition or subpartition are inserted into the partition or subpartition of the target table.
- If `ROWS=y` (default) and `IGNORE=y`, but the table already existed before import, then all rows for the specified partition or subpartition in the table are inserted into the table. The rows are stored according to the existing partitioning scheme of the target table.
- If `ROWS=n`, then Import does not insert data into the target table and continues to process other objects associated with the specified table and partition or subpartition in the file.
- If the target table is nonpartitioned, then the partitions and subpartitions are imported into the entire table. Import requires `IGNORE=y` to import one or more partitions or subpartitions from the export file into a nonpartitioned table on the import target system.

28.12.3 Migrating Data Across Partitions and Tables

If you specify a partition name for a composite partition, then all subpartitions within the composite partition are used as the source.

In the following example, the partition specified by the partition name is a composite partition. All of its subpartitions will be imported:

```
imp SYSTEM FILE=expdat.dmp FROMUSER=scott TABLES=b:py
```

The following example causes row data of partitions `qc` and `qd` of table `scott.e` to be imported into the table `scott.e`:

```
imp scott FILE=expdat.dmp TABLES=(e:qc, e:qd) IGNORE=y
```

If table `e` does not exist in the import target database, then it is created and data is inserted into the same partitions. If table `e` existed on the target system before import, then the row data is inserted into the partitions whose range allows insertion. The row data can end up in partitions of names other than `qc` and `qd`.

> **Note:**
>
> With partition-level Import to an existing table, you *must* set up the target partitions or subpartitions properly and use `IGNORE=y`.

28.13 Controlling Index Creation and Maintenance

These sections describe the behavior of Import with respect to index creation and maintenance.

- Delaying Index Creation
  Import provides you with the capability of delaying index creation and maintenance services until after completion of the import and insertion of exported data.
- Index Creation and Maintenance Controls
  Describes index creation and maintenance controls.

28.13.1 Delaying Index Creation

Import provides you with the capability of delaying index creation and maintenance services until after completion of the import and insertion of exported data.

Performing index creation, re-creation, or maintenance after Import completes is generally faster than updating the indexes for each row inserted by Import.

Index creation can be time consuming, and therefore can be done more efficiently after the import of all other objects has completed. You can postpone creation of indexes until after the import completes by specifying `INDEXES=n`. (`INDEXES=y` is the default.) You can then store the missing index definitions in a SQL script by running Import while using the `INDEXFILE` parameter. The index-creation statements that would otherwise be issued by Import are instead stored in the specified file.

After the import is complete, you must create the indexes, typically by using the contents of the file (specified with `INDEXFILE`) as a SQL script after specifying passwords for the connect statements.

28.13.2 Index Creation and Maintenance Controls

Describes index creation and maintenance controls.

If `SKIP_UNUSABLE_INDEXES=y`, then the Import utility postpones maintenance on all indexes that were set to Index Unusable before the Import. Other indexes (not previously set to Index Unusable) continue to be updated as rows are inserted. This approach saves on index updates during the import of existing tables.

Delayed index maintenance may cause a violation of an existing unique integrity constraint supported by the index. The existence of a unique integrity constraint on a table does not prevent existence of duplicate keys in a table that was imported with `INDEXES=n`. The supporting index will be in an `UNUSABLE` state until the duplicates are removed and the index is rebuilt.

- Example of Postponing Index Maintenance
  Shows an example of postponing index maintenance.

28.13.2.1 Example of Postponing Index Maintenance

Shows an example of postponing index maintenance.

Assume that partitioned table `t` with partitions `p1` and `p2` exists on the import target system. Assume that local indexes `p1_ind` on partition `p1` and `p2_ind` on partition `p2` exist also. Assume that partition `p1` contains a much larger amount of data in the existing table `t`, compared with the amount of data to be inserted by the export file (`expdat.dmp`). Assume that the reverse is true for `p2`.

Consequently, performing index updates for `p1_ind` during table data insertion time is more efficient than at partition index rebuild time. The opposite is true for `p2_ind`.

Users can postpone local index maintenance for `p2_ind` during import by using the following steps:

1. Issue the following SQL statement before import:

   ```
   ALTER TABLE t MODIFY PARTITION p2 UNUSABLE LOCAL INDEXES;
   ```

2. Issue the following Import command:

```
imp scott FILE=expdat.dmp TABLES = (t:p1, t:p2) IGNORE=y
SKIP_UNUSABLE_INDEXES=y
```

   This example executes the `ALTER SESSION SET SKIP_UNUSABLE_INDEXES=y` statement before performing the import.

3. Issue the following SQL statement after import:

```
ALTER TABLE t MODIFY PARTITION p2 REBUILD UNUSABLE LOCAL INDEXES;
```

In this example, local index `p1_ind` on `p1` will be updated when table data is inserted into partition `p1` during import. Local index `p2_ind` on `p2` will be updated at index rebuild time, after import.

28.14 Network Considerations for Using Oracle Net with Original Import

To perform imports over a network, you can use the Oracle Data Pump original Import utility (`imp`) with Oracle Net.

For example, if you run Import locally, then you can read data into a remote Oracle Database instance.

To use Import with Oracle Net, when you run the `imp` command and enter the username and password, include the connection qualifier string `@connect_string`. For the exact syntax of this clause, see the user's guide for your Oracle Net protocol.

Related Topics

- Entering a Connect String

28.15 Character Set and Globalization Support Considerations

These sections describe the globalization support behavior of Import with respect to character set conversion of user data and data definition language (DDL).

- User Data
  The Export utility always exports user data, including Unicode data, in the character sets of the Export server. (Character sets are specified at database creation.)
- Data Definition Language (DDL)
  Up to three character set conversions may be required for data definition language (DDL) during an export/import operation.
- Single-Byte Character Sets
  Some 8-bit characters can be lost (that is, converted to 7-bit equivalents) when you import an 8-bit character set export file.
- Multibyte Character Sets
  During character set conversion, any characters in the export file that have no equivalent in the target character set are replaced with a default character. (The default character is defined by the target character set.)

28.15.1 User Data

The Export utility always exports user data, including Unicode data, in the character sets of the Export server. (Character sets are specified at database creation.)

If the character sets of the source database are different than the character sets of the import database, then a single conversion is performed to automatically convert the data to the character sets of the Import server.

- Effect of Character Set Sorting Order on Conversions
  If the export character set has a different sorting order than the import character set, then tables that are partitioned on character columns may yield unpredictable results.

28.15.1.1 Effect of Character Set Sorting Order on Conversions

If the export character set has a different sorting order than the import character set, then tables that are partitioned on character columns may yield unpredictable results.

For example, consider the following table definition, which is produced on a database having an ASCII character set:

```
CREATE TABLE partlist
   (
   part     VARCHAR2(10),
   partno   NUMBER(2)
   )
PARTITION BY RANGE (part)
  (
  PARTITION part_low VALUES LESS THAN ('Z')
    TABLESPACE tbs_1,
  PARTITION part_mid VALUES LESS THAN ('z')
    TABLESPACE tbs_2,
  PARTITION part_high VALUES LESS THAN (MAXVALUE)
    TABLESPACE tbs_3
  );
```

This partitioning scheme makes sense because `z` comes after `Z` in ASCII character sets.

When this table is imported into a database based upon an EBCDIC character set, all of the rows in the `part_mid` partition will migrate to the `part_low` partition because `z` comes before `Z` in EBCDIC character sets. To obtain the desired results, the owner of `partlist` must repartition the table following the import.

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

28.15.2 Data Definition Language (DDL)

Up to three character set conversions may be required for data definition language (DDL) during an export/import operation.

Specifically:

1. Export writes export files using the character set specified in the `NLS_LANG` environment variable for the user session. A character set conversion is performed if the value of `NLS_LANG` differs from the database character set.
2. If the export file's character set is different than the import user session character set, then Import converts the character set to its user session character set. Import can only perform this conversion for single-byte character sets. This means that for multibyte character sets, the import file's character set must be identical to the export file's character set.
3. A final character set conversion may be performed if the target database's character set is different from the character set used by the import user session.

To minimize data loss due to character set conversions, ensure that the export database, the export user session, the import user session, and the import database all use the same character set.

28.15.3 Single-Byte Character Sets

Some 8-bit characters can be lost (that is, converted to 7-bit equivalents) when you import an 8-bit character set export file.

This occurs if the system on which the import occurs has a native 7-bit character set, or the `NLS_LANG` operating system environment variable is set to a 7-bit character set. Most often, this is apparent when accented characters lose the accent mark.

To avoid this unwanted conversion, you can set the `NLS_LANG` operating system environment variable to be that of the export file character set.

28.15.4 Multibyte Character Sets

During character set conversion, any characters in the export file that have no equivalent in the target character set are replaced with a default character. (The default character is defined by the target character set.)

To guarantee 100% conversion, the target character set must be a superset (or equivalent) of the source character set.

> **Note:**
>
> When the character set width differs between the Export server and the Import server, truncation of data can occur if conversion causes expansion of data. If truncation occurs, then Import displays a warning message.

28.16 Using Instance Affinity

You can use instance affinity to associate jobs with instances in databases you plan to export and import.

Be aware that there may be some compatibility issues if you are using a combination of releases.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for more information about affinity

28.17 Considerations When Importing Database Objects

These sections describe restrictions and points you should consider when you import particular database objects.

- Importing Object Identifiers
- Importing Existing Object Tables and Tables That Contain Object Types
  Importing existing Object Tables and tables that contain Object Types is one of the considerations when importing database objects. The tables must be created with the same definitions as were previously used or a compatible format (except for storage parameters).
- Importing Nested Tables
- Importing REF Data
  Importing `REF` data is one of the considerations when importing database objects. `REF` columns and attributes may contain a hidden `ROWID` that points to the referenced type instance.
- Importing BFILE Columns and Directory Aliases
  Importing BFILE Columns and Directory Aliases is one of the considerations when importing database objects. When you import table data that contains BFILE columns, the BFILE locator is imported with the directory alias and file name that was present at export time.
- Importing Foreign Function Libraries
  Importing Foreign Function Libraries is one of the considerations when importing database objects. Import does not verify that the location referenced by the foreign function library is correct.
- Importing Stored Procedures, Functions, and Packages
  The behavior of Import when a local stored procedure, function, or package is imported depends upon whether the `COMPILE` parameter is set to `y` or to`n`.
- Importing Java Objects
  Importing Java Objects is one of the considerations when importing database objects. When you import Java objects into any schema, the Import utility leaves the resolver unchanged.
- Importing External Tables
  Importing external tables is one of the considerations when importing database objects. Import does not verify that the location referenced by the external table is correct.
- Importing Advanced Queue (AQ) Tables
  Importing Advanced Queue Tables is a one of the considerations when importing database objects. Importing a queue table also imports any underlying queues and the related dictionary information.
- Importing LONG Columns
  Importing `LONG` columns is one of the considerations when importing database objects. In importing and exporting, the `LONG` columns must fit into memory with the rest of each row's data.

- Importing LOB Columns When Triggers Are Present
  Importing LOB columns when triggers are present is one of the considerations when importing database objects. The Import utility automatically changes all LOBs that were empty at export time to be `NULL` after they are imported.
- Importing Views
  Importing views that contain references to tables in other schemas requires that the importer have the `READ ANY TABLE` or `SELECT ANY TABLE` privilege.
- Importing Partitioned Tables
  Importing partitioned tables is one of the considerations when importing database objects. Import attempts to create a partitioned table with the same partition or subpartition names as the exported partitioned table, including names of the form `SYS_P`*nnn*.

28.17.1 Importing Object Identifiers

The Oracle database assigns object identifiers to uniquely identify object types, object tables, and rows in object tables. These object identifiers are preserved by Import.

When you import a table that references a type, but a type of that name already exists in the database, Import attempts to verify that the preexisting type is, in fact, the type used by the table (rather than a different type that just happens to have the same name).

To do this, Import compares the types's unique identifier (TOID) with the identifier stored in the export file. If those match, then Import then compares the type's unique hashcode with that stored in the export file. Import will not import table rows if the TOIDs or hashcodes do not match.

In some situations, you may not want this validation to occur on specified types (for example, if the types were created by a cartridge installation). You can use the parameter `TOID_NOVALIDATE` to specify types to exclude from the TOID and hashcode comparison. See TOID_NOVALIDATE for more information.

> **Note:**
>
> Be very careful about using `TOID_NOVALIDATE`, because type validation provides an important capability that helps avoid data corruption. Be sure you are confident of your knowledge of type validation and how it works before attempting to perform an import operation with this feature disabled.

Import uses the following criteria to decide how to handle object types, object tables, and rows in object tables:

- For object types, if `IGNORE=y`, the object type already exists, and the object identifiers, hashcodes, and type descriptors match, then no error is reported. If the object identifiers or hashcodes do not match and the parameter `TOID_NOVALIDATE` has not been set to ignore the object type, then an error is reported and any tables using the object type are not imported.
- For object types, if `IGNORE=n` and the object type already exists, then an error is reported. If the object identifiers, hashcodes, or type descriptors do not match and the parameter `TOID_NOVALIDATE` has not been set to ignore the object type, then any tables using the object type are not imported.
- For object tables, if `IGNORE=y`, then the table already exists, and the object identifiers, hashcodes, and type descriptors match, no error is reported. Rows are imported into the

object table. Import of rows may fail if rows with the same object identifier already exist in the object table. If the object identifiers, hashcodes, or type descriptors do not match, and the parameter TOID_NOVALIDATE has not been set to ignore the object type, then an error is reported and the table is not imported.

- For object tables, if IGNORE=n and the table already exists, then an error is reported and the table is not imported.

Because Import preserves object identifiers of object types and object tables, consider the following when you import objects from one schema into another schema using the FROMUSER and TOUSER parameters:

- If the FROMUSER object types and object tables already exist on the target system, then errors occur because the object identifiers of the TOUSER object types and object tables are already in use. The FROMUSER object types and object tables must be dropped from the system before the import is started.
- If an object table was created using the OID AS option to assign it the same object identifier as another table, then both tables cannot be imported. You can import one of the tables, but the second table receives an error because the object identifier is already in use.

28.17.2 Importing Existing Object Tables and Tables That Contain Object Types

Importing existing Object Tables and tables that contain Object Types is one of the considerations when importing database objects. The tables must be created with the same definitions as were previously used or a compatible format (except for storage parameters).

Users frequently create tables before importing data to reorganize tablespace usage or to change a table's storage parameters. The tables must be created with the same definitions as were previously used or a compatible format (except for storage parameters). For object tables and tables that contain columns of object types, format compatibilities are more restrictive.

For object tables and for tables containing columns of objects, each object the table references has its name, structure, and version information written out to the export file. Export also includes object type information from different schemas, as needed.

Import verifies the existence of each object type required by a table before importing the table data. This verification consists of a check of the object type's name followed by a comparison of the object type's structure and version from the import system with that found in the export file.

If an object type name is found on the import system, but the structure or version do not match that from the export file, then an error message is generated and the table data is not imported.

The Import parameter TOID_NOVALIDATE can be used to disable the verification of the object type's structure and version for specific objects.

28.17.3 Importing Nested Tables

Inner nested tables are exported separately from the outer table. Therefore, situations may arise where data in an inner nested table might not be properly imported:

- Suppose a table with an inner nested table is exported and then imported without dropping the table or removing rows from the table. If the IGNORE=y parameter is used, then there will be a constraint violation when inserting each row in the outer table. However, data in the inner nested table may be successfully imported, resulting in duplicate rows in the inner table.

- If nonrecoverable errors occur inserting data in outer tables, then the rest of the data in the outer table is skipped, but the corresponding inner table rows are not skipped. This may result in inner table rows not being referenced by any row in the outer table.
- If an insert to an inner table fails after a recoverable error, then its outer table row will already have been inserted in the outer table and data will continue to be inserted into it and any other inner tables of the containing table. This circumstance results in a partial logical row.
- If nonrecoverable errors occur inserting data in an inner table, then Import skips the rest of that inner table's data but does not skip the outer table or other nested tables.

You should always carefully examine the log file for errors in outer tables and inner tables. To be consistent, table data may need to be modified or deleted.

Because inner nested tables are imported separately from the outer table, attempts to access data from them while importing may produce unexpected results. For example, if an outer row is accessed before its inner rows are imported, an incomplete row may be returned to the user.

28.17.4 Importing REF Data

Importing `REF` data is one of the considerations when importing database objects. `REF` columns and attributes may contain a hidden `ROWID` that points to the referenced type instance.

`REF` columns and attributes may contain a hidden `ROWID` that points to the referenced type instance. Import does not automatically recompute these `ROWID`s for the target database. You should execute the following statement to reset the `ROWID`s to their proper values:

```
ANALYZE TABLE [schema.]table VALIDATE REF UPDATE;
```

> **See Also:**
>
> *Oracle Database SQL Language Reference* for more information about the `ANALYZE` statement

28.17.5 Importing BFILE Columns and Directory Aliases

Importing BFILE Columns and Directory Aliases is one of the considerations when importing database objects. When you import table data that contains BFILE columns, the BFILE locator is imported with the directory alias and file name that was present at export time.

Export and Import do not copy data referenced by `BFILE` columns and attributes from the source database to the target database. Export and Import only propagate the names of the files and the directory aliases referenced by the `BFILE` columns. It is the responsibility of the DBA or user to move the actual files referenced through `BFILE` columns and attributes.

When you import table data that contains `BFILE` columns, the `BFILE` locator is imported with the directory alias and file name that was present at export time. Import does not verify that the directory alias or file exists. If the directory alias or file does not exist, then an error occurs when the user accesses the `BFILE` data.

For directory aliases, if the operating system directory syntax used in the export system is not valid on the import system, then no error is reported at import time. The error occurs when the

user seeks subsequent access to the file data. It is the responsibility of the DBA or user to ensure the directory alias is valid on the import system.

28.17.6 Importing Foreign Function Libraries

Importing Foreign Function Libraries is one of the considerations when importing database objects. Import does not verify that the location referenced by the foreign function library is correct.

Import does not verify that the location referenced by the foreign function library is correct. If the formats for directory and file names used in the library's specification on the export file are invalid on the import system, then no error is reported at import time. Subsequent usage of the callout functions will receive an error.

It is the responsibility of the DBA or user to manually move the library and ensure the library's specification is valid on the import system.

28.17.7 Importing Stored Procedures, Functions, and Packages

The behavior of Import when a local stored procedure, function, or package is imported depends upon whether the `COMPILE` parameter is set to `y` or to`n`.

The behavior of Import when a local stored procedure, function, or package is imported depends upon whether the `COMPILE` parameter is set to `y` or to `n`.

When a local stored procedure, function, or package is imported and `COMPILE=y`, the procedure, function, or package is recompiled upon import and retains its original timestamp specification. If the compilation is successful, then it can be accessed by remote procedures without error.

If `COMPILE=n`, then the procedure, function, or package is still imported, but the original timestamp is lost. The compilation takes place the next time the procedure, function, or package is used.

> **See Also:**
>
> COMPILE

28.17.8 Importing Java Objects

Importing Java Objects is one of the considerations when importing database objects. When you import Java objects into any schema, the Import utility leaves the resolver unchanged.

When you import Java objects into any schema, the Import utility leaves the resolver unchanged. (The resolver is the list of schemas used to resolve Java full names.) This means that after an import, all user classes are left in an invalid state until they are either implicitly or explicitly revalidated. An implicit revalidation occurs the first time the classes are referenced. An explicit revalidation occurs when the SQL statement `ALTER JAVA CLASS...RESOLVE` is used. Both methods result in the user classes being resolved successfully and becoming valid.

28.17.9 Importing External Tables

Importing external tables is one of the considerations when importing database objects. Import does not verify that the location referenced by the external table is correct.

Import does not verify that the location referenced by the external table is correct. If the formats for directory and file names used in the table's specification on the export file are invalid on the import system, then no error is reported at import time. Subsequent usage of the callout functions will result in an error.

It is the responsibility of the DBA or user to manually move the table and ensure the table's specification is valid on the import system.

28.17.10 Importing Advanced Queue (AQ) Tables

Importing Advanced Queue Tables is a one of the considerations when importing database objects. Importing a queue table also imports any underlying queues and the related dictionary information.

Importing a queue table also imports any underlying queues and the related dictionary information. A queue can be imported only at the granularity level of the queue table. When a queue table is imported, export pre-table and post-table action procedures maintain the queue dictionary.

> **See Also:**
>
> *Oracle Database Advanced Queuing User's Guide*

28.17.11 Importing LONG Columns

Importing `LONG` columns is one of the considerations when importing database objects. In importing and exporting, the `LONG` columns must fit into memory with the rest of each row's data.

> **Caution:**
>
> This feature is deprecated, and can be desupported in a future release.

All forms of `LONG` data types (`LONG`, `LONG RAW`, `LONG VARCHAR`, `LONG VARRAW`) were deprecated in Oracle8i Release 8.1.6. For succeeding releases, the `LONG` data type was provided for backward compatibility with existing applications. In new applications developed with later releases, Oracle strongly recommends that you use `CLOB` and `NCLOB` data types for large amounts of character data.

`LONG` columns can be up to 2 gigabytes in length. In importing and exporting, the `LONG` columns must fit into memory with the rest of each row's data. The memory used to store `LONG` columns, however, does not need to be contiguous, because `LONG` data is loaded in sections.

You should use Import to convert `LONG` columns to `CLOB` columns. To convert `LONG` columns, first create a table specifying the new `CLOB` column. When Import is run, the `LONG` data is converted

to CLOB format. The same technique can be used to convert LONG RAW columns to BLOB columns.

> **Note:**
>
> Because LONG data types are deprecated, Oracle strongly recommends that you convert existing LONG columns to LOB columns, and update applications accordingly. LOB columns are subject to far fewer restrictions than LONG columns.

28.17.12 Importing LOB Columns When Triggers Are Present

Importing LOB columns when triggers are present is one of the considerations when importing database objects. The Import utility automatically changes all LOBs that were empty at export time to be NULL after they are imported.

As of Oracle Database 10g, LOB handling has been improved to ensure that triggers work properly and that performance remains high when LOBs are being loaded. To achieve these improvements, the Import utility automatically changes all LOBs that were empty at export time to be NULL after they are imported.

If you have applications that expect the LOBs to be empty rather than NULL, then after the import you can issue a SQL UPDATE statement for each LOB column. Depending on whether the LOB column type was a BLOB or a CLOB, the syntax would be one of the following:

```
UPDATE <tablename> SET <lob column> = EMPTY_BLOB() WHERE <lob column> = IS
NULL;
UPDATE <tablename> SET <lob column> = EMPTY_CLOB() WHERE <lob column> = IS
NULL;
```

It is important to note that once the import is performed, there is no way to distinguish between LOB columns that are NULL versus those that are empty. Therefore, if that information is important to the integrity of your data, then be sure you know which LOB columns are NULL and which are empty before you perform the import.

28.17.13 Importing Views

Importing views that contain references to tables in other schemas requires that the importer have the READ ANY TABLE or SELECT ANY TABLE privilege.

Views are exported in dependency order. In some cases, Export must determine the ordering, rather than obtaining the order from the database. In doing so, Export may not always be able to duplicate the correct ordering, resulting in compilation warnings when a view is imported, and the failure to import column comments on such views.

In particular, if viewa uses the stored procedure procb, and procb uses the view viewc, then Export cannot determine the proper ordering of viewa and viewc. If viewa is exported before viewc, and procb already exists on the import system, then viewa receives compilation warnings at import time.

Grants on views are imported even if a view has compilation errors. A view could have compilation errors if an object it depends on, such as a table, procedure, or another view, does not exist when the view is created. If a base table does not exist, then the server cannot validate that the grantor has the proper privileges on the base table with the GRANT option.

Access violations could occur when the view is used if the grantor does not have the proper privileges after the missing tables are created.

Importing views that contain references to tables in other schemas requires that the importer have the `READ ANY TABLE` or `SELECT ANY TABLE` privilege. If the importer has not been granted this privilege, then the views will be imported in an uncompiled state. Note that granting the privilege to a role is insufficient. For the view to be compiled, the privilege must be granted directly to the importer.

28.17.14 Importing Partitioned Tables

Importing partitioned tables is one of the considerations when importing database objects. Import attempts to create a partitioned table with the same partition or subpartition names as the exported partitioned table, including names of the form `SYS_P`*nnn*.

Import attempts to create a partitioned table with the same partition or subpartition names as the exported partitioned table, including names of the form `SYS_P`*nnn*. If a table with the same name already exists, then Import processing depends on the value of the `IGNORE` parameter.

Unless `SKIP_UNUSABLE_INDEXES=y`, inserting the exported data into the target table fails if Import cannot update a nonpartitioned index or index partition that is marked Indexes Unusable or is otherwise not suitable.

28.18 Support for Fine-Grained Access Control

To restore the fine-grained access control policies, the user who imports from an export file containing such tables must have the `EXECUTE` privilege on the `DBMS_RLS` package, so that the security policies on the tables can be reinstated.

If a user without the correct privileges attempts to import from an export file that contains tables with fine-grained access control policies, then a warning message is issued.

28.19 Snapshots and Snapshot Logs

In certain situations, particularly those involving data warehousing, snapshots may be referred to as *materialized views*. These sections retain the term snapshot.

- Snapshot Log
  The snapshot log in a dump file is imported if the Data Pump control table already exists for the database to which you are importing, and it has a snapshot log.
- Snapshots
  A snapshot that has been restored from an export file has reverted to a previous state.

28.19.1 Snapshot Log

The snapshot log in a dump file is imported if the Data Pump control table already exists for the database to which you are importing, and it has a snapshot log.

When a `ROWID` snapshot log is exported, The `ROWID` values stored in the snapshot log have no meaning upon import. As a result, each `ROWID` snapshot's first attempt to do a fast refresh fails, generating an error indicating that a complete refresh is required.

To avoid the refresh error, do a complete refresh after importing a `ROWID` snapshot log. After you have done a complete refresh, subsequent fast refreshes will work properly. In contrast,

when a primary key snapshot log is exported, the values of the primary keys do retain their meaning upon import. Therefore, primary key snapshots can do a fast refresh after the import.

28.19.2 Snapshots

A snapshot that has been restored from an export file has reverted to a previous state.

On import, the time of the last refresh is imported as part of the snapshot table definition. The function that calculates the next refresh time is also imported.

Each refresh leaves a signature. A fast refresh uses the log entries that date from the time of that signature to bring the snapshot up to date. When the fast refresh is complete, the signature is deleted and a new signature is created. Any log entries that are not needed to refresh other snapshots are also deleted (all log entries with times before the earliest remaining signature).

- Importing a Snapshot
  When you restore a snapshot from an export file, you may encounter a problem under certain circumstances.
- Importing a Snapshot into a Different Schema
  Snapshots and related items are exported with the schema name given in the DDL statements.

28.19.2.1 Importing a Snapshot

When you restore a snapshot from an export file, you may encounter a problem under certain circumstances.

Assume that a snapshot is refreshed at time A, exported at time B, and refreshed again at time C. Then, because of corruption or other problems, the snapshot needs to be restored by dropping the snapshot and importing it again. The newly imported version has the last refresh time recorded as time A. However, log entries needed for a fast refresh may no longer exist. If the log entries do exist (because they are needed for another snapshot that has yet to be refreshed), then they are used, and the fast refresh completes successfully. Otherwise, the fast refresh fails, generating an error that says a complete refresh is required.

28.19.2.2 Importing a Snapshot into a Different Schema

Snapshots and related items are exported with the schema name given in the DDL statements.

To import them into a different schema, use the `FROMUSER` and `TOUSER` parameters. This does not apply to snapshot logs, which cannot be imported into a different schema.

> **Note:**
>
> Schema names that appear inside function-based indexes, functions, procedures, triggers, type bodies, views, and so on, are *not* affected by `FROMUSER` or `TOUSER` processing. Only the *name* of the object is affected. After the import has completed, items in any `TOUSER` schema should be manually checked for references to old (`FROMUSER`) schemas, and corrected if necessary.

28.20 Transportable Tablespaces

The transportable tablespace feature enables you to move a set of tablespaces from one Oracle database to another.

> **Note:**
>
> You cannot export transportable tablespaces and then import them into a database at a lower release level. The target database must be at the same or later release level as the source database.

To move or copy a set of tablespaces, you must make the tablespaces read-only, manually copy the data files of these tablespaces to the target database, and use Export and Import to move the database information (metadata) stored in the data dictionary over to the target database. The transport of the data files can be done using any facility for copying flat binary files, such as the operating system copying facility, binary-mode FTP, or publishing on CD-ROMs.

After copying the data files and exporting the metadata, you can optionally put the tablespaces in read/write mode.

Export and Import provide the following parameters to enable movement of transportable tablespace metadata.

- `TABLESPACES`
- `TRANSPORT_TABLESPACE`

See TABLESPACES and TRANSPORT_TABLESPACE for information about using these parameters during an import operation.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for details about managing transportable tablespaces

28.21 Storage Parameters

By default, a table is imported into its original tablespace.

If the tablespace no longer exists, or the user does not have sufficient quota in the tablespace, then the system uses the default tablespace for that user, unless the table:

- Is partitioned
- Is a type table
- Contains LOB, `VARRAY`, or `OPAQUE` type columns
- Has an index-organized table (IOT) overflow segment

If the user does not have sufficient quota in the default tablespace, then the user's tables are not imported. See Reorganizing Tablespaces to see how you can use this to your advantage.

- The OPTIMAL Parameter
  The storage parameter `OPTIMAL` for rollback segments is not preserved during export and import.
- Storage Parameters for OID Indexes and LOB Columns
  Tables are exported with their current storage parameters.
- Overriding Storage Parameters
  Before using the Import utility to import data, you may want to create large tables with different storage parameters.

28.21.1 The OPTIMAL Parameter

The storage parameter `OPTIMAL` for rollback segments is not preserved during export and import.

28.21.2 Storage Parameters for OID Indexes and LOB Columns

Tables are exported with their current storage parameters.

For object tables, the OIDINDEX is created with its current storage parameters and name, if given. For tables that contain LOB, `VARRAY`, or `OPAQUE` type columns, LOB, `VARRAY`, or `OPAQUE` type data is created with their current storage parameters.

If you alter the storage parameters of existing tables before exporting, then the tables are exported using those altered storage parameters. Note, however, that storage parameters for LOB data cannot be altered before exporting (for example, chunk size for a LOB column, whether a LOB column is `CACHE` or `NOCACHE`, and so forth).

Note that LOB data might not reside in the same tablespace as the containing table. The tablespace for that data must be read/write at the time of import or the table will not be imported.

If LOB data resides in a tablespace that does not exist at the time of import, or the user does not have the necessary quota in that tablespace, then the table will not be imported. Because there can be multiple tablespace clauses, including one for the table, Import cannot determine which tablespace clause caused the error.

28.21.3 Overriding Storage Parameters

Before using the Import utility to import data, you may want to create large tables with different storage parameters.

If so, then you must specify `IGNORE=y` on the command line or in the parameter file.

28.22 Read-Only Tablespaces

Read-only tablespaces can be exported. On import, if the tablespace does not already exist in the target database, then the tablespace is created as a read/write tablespace.

To get read-only functionality, you must manually make the tablespace read-only after the import.

If the tablespace already exists in the target database and is read-only, then you must make it read/write before the import.

28.23 Dropping a Tablespace

You can drop a tablespace by redefining the objects to use different tablespaces before the import. You can then issue the `imp` command and specify `IGNORE=y`.

In many cases, you can drop a tablespace by doing a full database export, then creating a zero-block tablespace with the same name (before logging off) as the tablespace you want to drop. During import, with `IGNORE=y`, the relevant `CREATE TABLESPACE` statement will fail and prevent the creation of the unwanted tablespace.

All objects from that tablespace will be imported into their owner's default tablespace except for partitioned tables, type tables, and tables that contain LOB or `VARRAY` columns or index-only tables with overflow segments. Import cannot determine which tablespace caused the error. Instead, you must first create a table and then import the table again, specifying `IGNORE=y`.

Objects are not imported into the default tablespace if the tablespace does not exist, or you do not have the necessary quotas for your default tablespace.

28.24 Reorganizing Tablespaces

If a user's quota allows it, the user's tables are imported into the same tablespace from which they were exported.

However, if the tablespace no longer exists or the user does not have the necessary quota, then the system uses the default tablespace for that user as long as the table is unpartitioned, contains no LOB or `VARRAY` columns, is not a type table, and is not an index-only table with an overflow segment. This scenario can be used to move a user's tables from one tablespace to another.

For example, you need to move `joe`'s tables from tablespace `A` to tablespace `B` after a full database export. Follow these steps:

1. If `joe` has the `UNLIMITED TABLESPACE` privilege, then revoke it. Set `joe`'s quota on tablespace `A` to zero. Also revoke all roles that might have such privileges or quotas.

   When you revoke a role, it does not have a cascade effect. Therefore, users who were granted other roles by `joe` will be unaffected.
2. Export `joe`'s tables.
3. Drop `joe`'s tables from tablespace `A`.
4. Give `joe` a quota on tablespace `B` and make it the default tablespace for `joe`.
5. Import `joe`'s tables. (By default, Import puts `joe`'s tables into tablespace `B`.)

28.25 Importing Statistics

If statistics are requested at export time and analyzer statistics are available for a table, then Export will include the `ANALYZE` statement used to recalculate the statistics for the table into the dump file.

In most circumstances, Export will also write the precalculated optimizer statistics for tables, indexes, and columns to the dump file. See the description of the Import parameter STATISTICS.

Because of the time it takes to perform an `ANALYZE` statement, it is usually preferable for Import to use the precalculated optimizer statistics for a table (and its indexes and columns) rather than execute the `ANALYZE` statement saved by Export. By default, Import will always use the precalculated statistics that are found in the export dump file.

The Export utility flags certain precalculated statistics as questionable. The importer might want to import only unquestionable statistics, not precalculated statistics, in the following situations:

- Character set translations between the dump file and the import client and the import database could potentially change collating sequences that are implicit in the precalculated statistics.
- Row errors occurred while importing the table.
- A partition level import is performed (column statistics will no longer be accurate).

> **Note:**
>
> Specifying `ROWS=n` will not prevent the use of precalculated statistics. This feature allows plan generation for queries to be tuned in a nonproduction database using statistics from a production database. In these cases, the import should specify `STATISTICS=SAFE`.

In certain situations, the importer might want to always use `ANALYZE` statements rather than precalculated statistics. For example, the statistics gathered from a fragmented database may not be relevant when the data is imported in a compressed form. In these cases, the importer should specify `STATISTICS=RECALCULATE` to force the recalculation of statistics.

If you do not want any statistics to be established by Import, then you should specify `STATISTICS=NONE`.

28.26 Using Export and Import to Partition a Database Migration

When you use the Export and Import utilities to migrate a large database, it may be more efficient to partition the migration into multiple export and import jobs.

If you decide to partition the migration, then be aware of the following advantages and disadvantages.

- Advantages of Partitioning a Migration
  Describes the advantages of partitioning a migration.
- Disadvantages of Partitioning a Migration
  Describes the disadvantages of partitioning a migration.
- How to Use Export and Import to Partition a Database Migration
  To use Export and Import to perform a database migration in a partitioned manner, complete this procedure.

28.26.1 Advantages of Partitioning a Migration

Describes the advantages of partitioning a migration.

Specifically:

- Time required for the migration may be reduced, because many of the subjobs can be run in parallel.
- The import can start as soon as the first export subjob completes, rather than waiting for the entire export to complete.

28.26.2 Disadvantages of Partitioning a Migration

Describes the disadvantages of partitioning a migration.

Specifically:

- The export and import processes become more complex.
- Support of cross-schema references for certain types of objects may be compromised. For example, if a schema contains a table with a foreign key constraint against a table in a different schema, then you may not have the required parent records when you import the table into the dependent schema.

28.26.3 How to Use Export and Import to Partition a Database Migration

To use Export and Import to perform a database migration in a partitioned manner, complete this procedure.

> **Note:**
>
> Original Export (`exp`) was desupported in Oracle Database 11g. However, you can use `exp` on an earlier release Oracle Database, and use the original Import utility (`imp`) to a newer release Oracle Database that supports Oracle Data Pump.

1. For all top-level metadata in the database, issue the following commands:
   a. `exp FILE=full FULL=y CONSTRAINTS=n TRIGGERS=n ROWS=n INDEXES=n`
   b. `imp FILE=full FULL=y`
2. For each schema*n* in the database, issue the following commands:
   a. `exp OWNER=schema`*n* `FILE=schema`*n*
   b. `imp FILE=schema`*n* `FROMUSER=schema`*n* `TOUSER=schema`*n* `IGNORE=y`

All exports can be done in parallel. When the import of `full.dmp` completes, all remaining imports can also be done in parallel.

28.27 Tuning Considerations for Import Operations

These sections discuss some ways to improve the performance of an import operation.

- Changing System-Level Options
  Describes system-level options that may help improve the performance of an import operation.
- Changing Initialization Parameters
  These suggestions about settings in your initialization parameter file may help improve performance of an import operation.

- Changing Import Options
  These suggestions about the usage of import options may help improve performance.
- Dealing with Large Amounts of LOB Data
  Describes importing large amounts of LOB data.
- Dealing with Large Amounts of LONG Data
  Keep in mind that importing a table with a `LONG` column can cause a higher rate of I/O and disk usage, resulting in reduced performance of the import operation.

28.27.1 Changing System-Level Options

Describes system-level options that may help improve the performance of an import operation.

Specifically :

- Create and use one large rollback segment and take all other rollback segments offline. Generally a rollback segment that is one half the size of the largest table being imported should be big enough. It can also help if the rollback segment is created with the minimum number of two extents, of equal size.

> **Note:**
>
> Oracle recommends that you use automatic undo management instead of rollback segments.

- Put the database in `NOARCHIVELOG` mode until the import is complete. This will reduce the overhead of creating and managing archive logs.
- Create several large redo files and take any small redo log files offline. This will result in fewer log switches being made.
- If possible, have the rollback segment, table data, and redo log files all on separate disks. This will reduce I/O contention and increase throughput.
- If possible, do not run any other jobs at the same time that may compete with the import operation for system resources.
- Ensure that there are no statistics on dictionary tables.
- Set `TRACE_LEVEL_CLIENT=OFF` in the `sqlnet.ora` file.
- If possible, increase the value of `DB_BLOCK_SIZE` when you re-create the database. The larger the block size, the smaller the number of I/O cycles needed. *This change is permanent, so be sure to carefully consider all effects it will have before making it.*

28.27.2 Changing Initialization Parameters

These suggestions about settings in your initialization parameter file may help improve performance of an import operation.

- Set `LOG_CHECKPOINT_INTERVAL` to a number that is larger than the size of the redo log files. This number is in operating system blocks (512 on most UNIX systems). This reduces checkpoints to a minimum (at log switching time).
- Increase the value of `SORT_AREA_SIZE`. The amount you increase it depends on other activity taking place on the system and on the amount of free memory available. (If the system begins swapping and paging, then the value is probably set too high.)

- Increase the value for `DB_BLOCK_BUFFERS` and `SHARED_POOL_SIZE`.

28.27.3 Changing Import Options

These suggestions about the usage of import options may help improve performance.

Be sure to also read the individual descriptions of all the available options in Import Parameters.

- Set `COMMIT=N`. This causes Import to commit after each object (table), not after each buffer. This is why one large rollback segment is needed. (Because rollback segments will be deprecated in future releases, Oracle recommends that you use automatic undo management instead.)
- Specify a large value for `BUFFER` or `RECORDLENGTH`, depending on system activity, database size, and so on. A larger size reduces the number of times that the export file has to be accessed for data. Several megabytes is usually enough. Be sure to check your system for excessive paging and swapping activity, which can indicate that the buffer size is too large.
- Consider setting `INDEXES=N` because indexes can be created at some point after the import, when time is not a factor. If you choose to do this, then you need to use the `INDEXFILE` parameter to extract the DLL for the index creation or to rerun the import with `INDEXES=Y` and `ROWS=N`.

28.27.4 Dealing with Large Amounts of LOB Data

Describes importing large amounts of LOB data.

Specifically:

- Eliminating indexes significantly reduces total import time. This is because LOB data requires special consideration during an import because the LOB locator has a primary key that cannot be explicitly dropped or ignored during an import.
- Ensure that there is enough space available in large contiguous chunks to complete the data load.

28.27.5 Dealing with Large Amounts of LONG Data

Keep in mind that importing a table with a `LONG` column can cause a higher rate of I/O and disk usage, resulting in reduced performance of the import operation.

> **Caution:**
>
> This feature is deprecated, and can be desupported in a future release.

All forms of `LONG` data types (`LONG`, `LONG RAW`, `LONG VARCHAR`, `LONG VARRAW`) were deprecated in Oracle8i Release 8.1.6. For succeeding releases, the `LONG` data type was provided for backward compatibility with existing applications. In new applications developed with later releases, Oracle strongly recommends that you use `CLOB` and `NCLOB` data types for large amounts of character data.

There are no specific parameters that will improve performance during an import of large amounts of `LONG` data, although some of the more general tuning suggestions made in this section may help overall performance.

> **See Also:**
>
> Importing LONG Columns

28.28 Using Different Releases of Export and Import

These sections describe compatibility issues that relate to using different releases of Export and the Oracle database.

Whenever you are moving data between different releases of the Oracle database, the following basic rules apply:

- The Import utility and the database to which data is being imported (the target database) must be the same version. For example, if you try to use the Import utility 9.2.0.7 to import into a 9.2.0.8 database, then you may encounter errors.
- The version of the Export utility must be equal to the version of either the source or target database, whichever is earlier.

  For example, to create an export file for an import into a later release database, use a version of the Export utility that equals the source database. Conversely, to create an export file for an import into an earlier release database, use a version of the Export utility that equals the version of the target database.

  – In general, you can use the Export utility from any Oracle8 release to export from an Oracle9*i* server and create an Oracle8 export file.
- Restrictions When Using Different Releases of Export and Import
  Restrictions apply when you are using different releases of Export and Import.
- Examples of Using Different Releases of Export and Import
  Using different releases of Export and Import.

28.28.1 Restrictions When Using Different Releases of Export and Import

Restrictions apply when you are using different releases of Export and Import.

Specifically:

- Export dump files can be read only by the Import utility because they are stored in a special binary format.
- Any export dump file can be imported into a later release of the Oracle database.
- The Import utility cannot read export dump files created by the Export utility of a later maintenance release or version. For example, a release 9.2 export dump file cannot be imported by a release 9.0.1 Import utility.
- Whenever a lower version of the Export utility runs with a later version of the Oracle database, categories of database objects that did not exist in the earlier version are excluded from the export.
- Export files generated by Oracle9*i* Export, either direct path or conventional path, are incompatible with earlier releases of Import and can be imported only with Oracle9*i* Import. When backward compatibility is an issue, use the earlier release or version of the Export utility against the Oracle9*i* database.

28.28.2 Examples of Using Different Releases of Export and Import

Using different releases of Export and Import.

Table 28-5 shows some examples of which Export and Import releases to use when moving data between different releases of the Oracle database.

Table 28-5 Using Different Releases of Export and Import

Export from->Import to	Use Export Release	Use Import Release
8.1.6 -> 8.1.6	8.1.6	8.1.6
8.1.5 -> 8.0.6	8.0.6	8.0.6
8.1.7 -> 8.1.6	8.1.6	8.1.6
9.0.1 -> 8.1.6	8.1.6	8.1.6
9.0.1 -> 9.0.2	9.0.1	9.0.2
9.0.2 -> 10.1.0	9.0.2	10.1.0
10.1.0 -> 9.0.2	9.0.2	9.0.2

Table 28-5 covers moving data only between the original Export and Import utilities. For Oracle Database 10*g* release 1 (10.1) or later, Oracle recommends the Data Pump Export and Import utilities in most cases because these utilities provide greatly enhanced performance compared to the original Export and Import utilities.

> **See Also:**
>
> *Oracle Database Upgrade Guide* for more information about exporting and importing data between different releases, including releases later than 10.1