

# Migrating Data To A New Database

## *Introduction*

As databases and user communities grow, and technology continues to improve, users' need for more power and flexibility often leads to the purchase of new server hardware. The database administrator must be prepared to move existing data quickly and completely from the old to the new server, usually in a narrow window of time.

While Oracle provides export and import facilities for the migration of data, they must be used carefully, and in combination with other operations, to ensure the smooth transfer of data. This paper describes a proven technique for populating your new database in the smallest possible amount of time.

## *Overview*

In this section, we'll discuss the skills required to perform this task, a high-level description of the steps involved, the reasons why a particular sequence of operations is necessary for successful and timely completion of a data transfer, and the assumptions we've made in preparing this technique.

## Required skills

First, let's review the skills you should have to perform a full data migration.

Most importantly, you should be an experienced Oracle database administrator (DBA). While the techniques described in this paper have been carefully thought out and tested, there is no guarantee that something won't go wrong, and you need to have diagnostic and troubleshooting skills that are an important part of a DBA's capabilities. You should be familiar with the export and import utilities, and experienced with the technique of generating SQL scripts with an SQL `select` statement (see the `gen*.sql` scripts in Appendix B).

You should also have at least a working knowledge of system administration. You will be called upon to transfer very large files, and may encounter disk space issues that will require system commands to solve. Skills on the new database's operating system are most important, but knowledge of multiple platforms is valuable if you are moving from one OS to another.

## Summary of steps

Briefly, the approach covered here consists of the following steps:

- Prepare source database
- Prepare target database
- Export source data
- Import new data
- Restore target database state

## Why this approach?

Let's consider the need for some of the operations required for this task; in particular, the preparation of the target database, which involves dropping keys and indexes, and disabling triggers.

Think for a moment about database structures such as keys, indexes, and triggers. Typically, these objects exist to ensure that the manipulation of data does not cause inconsistencies, or that certain actions are taken as a consequence of data manipulation. The data in an existing schema, however, has already been processed by triggers and had constraints enforced against it, so there is no need to do so again; in fact, allowing triggers to fire might even cause incorrect results. In addition, since Oracle always imports tables in alphabetical order, it is very likely that import will attempt to insert a child row before its parent row exists, thus causing that insert to fail and your import to finish only partially complete.

If indexes are left in place during import, the operation will not fail; but it will have two quite undesirable results. First, the time to complete the import will be at least an order of magnitude greater than it would be with indexes absent. Second, the index will end in a very unbalanced state in comparison to an index built after all rows are in place; this condition will reduce efficiency of index lookups.

Seen from the point of view of performance, there is an even more compelling reason why certain structures must be removed before importing data: when you are importing a ten-million-row table, do you really want the insert triggers firing for each row?

In summary, we take the steps outlined here for two reasons: to ensure the success of a single import session, and to increase the speed of the import.

## Assumptions

In order for our data transfer approach to succeed, we assume the following:

- The new database structure is complete and is functionally equivalent to the source database; i.e., all schema objects are in place, including grants, synonyms, and the like. This can be accomplished in a number of ways: through third-party software, full database export and import, locally-developed scripts, or any combination of these. The exact method you use is up to you and is not within the scope of this paper.
- There will be no database activity during the transfer. No users are connected to the source database, and the data is consistent.
- The new database must contain an exact copy of the source data.
- The time for performing the transfer is limited: overnight, or over a weekend. There is usually no time for a second try without disrupting the normal course of business.

Before we start any of the work on moving the database, we first have to create a functionally equivalent copy, with or without data, of the existing one.

### ***Detailed sequence of operations***

In this section, we'll cover all of the steps that you will need to take.

## Prepare source database (export)

First, generate a list of tables to export. Exclude snapshot base tables, snapshot log tables, and temporary tables using a select statement such as

```
select ','|| table_name from user_tables t1 where not exists
(select 1 from user_snapshots t2 where t2.table_name = t1.table_name)
and not exists (select 1 from user_snapshot_logs t3 where t3.log_table
= t1.table_name)
```

You can use the resulting file as the basis for an export parameter file; the list of tables can be enclosed in a `tables=( )` list (remember to remove the leading comma from the first table name in the list).

Add to the top of the parameter file the export options:

```
constraints = n
indexes = n
grants = n
direct = y (Oracle 7.3 and later)
```

The first three import parameters mean that the definitions of table constraints, indexes on the tables, and grants on the tables will not be exported. Since our assumption here is that all of these objects already exist in the target database, we don't want them in the export file; excluding them also provides for a slightly faster export session.

Then run your export with the command line

```
exp username/password parfile=<filename>
```

## Prepare target database

Perform an export of your target database structure with a command line such as

```
exp username/password file=db_struct.exp rows=n grants=n
```

The purpose of this step is to save all of the target database's primary key, unique, and referential constraints, and index definitions, in an export file. We don't, of course, want to export the data rows of the target database(`rows=n`), since we're replacing them with the data from the source. We don't need to export grants, either, because they are unaffected by our pre-import operations.

Next generate scripts (using the `gen*.sql` scripts in Appendix B) to drop primary and foreign keys and indexes; to disable and enable all triggers for each table; to drop, re-create, and grant access to sequences; and to truncate tables. Then execute the drop, disable, and truncate scripts.

The shell script `pre_import`, in Appendix B, can be used on a Unix system to perform all of these steps. Note that the script may require customization for your environment.

## Import data

Before beginning the data import process, if your database is in `archivelog` mode, you should shut down the database, start up and mount (but not open) it, and switch to `noarchivelog` mode (`alter database noarchivelog`). Then open the database (`alter database open`).

You may also want to consider increasing the size and/or number of redo logs, to avoid waits for checkpoints to complete while importing your data. You do this with the `alter database add logfile` command. If you choose to increase the size of your log files, you can do so only by adding new log files of the desired size, then dropping the old ones; log files to be dropped must be in `INACTIVE` status before you can drop them. Use `alter system switch logfile` to force a log switch to one of your new log files, if necessary, before dropping the old files with `alter database drop logfile`.

Finally, be sure that you have at least one jumbo-sized rollback segment in your database (a good idea in general), and take all the other rollback segments offline, so that import is forced to use the large one. A rollback segment in its own tablespace of at least several hundred megabytes, with initial and next extent sizes of 10 to 20 megabytes, qualifies as a jumbo-sized rollback segment.

Then, perform the import of your export file from the source database with the following parameters:

- `ignore = y`
- `buffer = 10485760`
- `commit = y`

The parameter `ignore = y` is required; it specifies that import should ignore the fact that the table in the export file already exists in the database, and simply insert the table's exported data into the existing table. The `buffer = 10485760` parameter yields larger array inserts during import. And `commit = y` causes import to issue a commit after each array insert; with very large tables, this helps prevent running out of rollback segment space during the import.

## Restore target database state

Once the data import is complete, it's time to restore the objects that were dropped earlier. Do so by importing the export file you made of the target database structure. The import will restore primary, unique, and foreign keys, and all indexes. You may safely ignore error messages regarding failure to create check constraints due to the fact that they already exist (we don't drop check constraints before import).

Next, enable all triggers by running the script generated before the import, and execute the scripts that drop, create (with new initial values based on the source database), and grant access to sequences (you can perform all of these steps by using the `post_import` shell script in Appendix B).

Finally, if snapshots exist in the database, perform a complete refresh of each one. You can easily create a script that does so by running `gen_refall.sql` from Appendix B.

Your target database is complete.

## ***Where to get more information***

For zip file containing scripts, check `dataxfer.zip` in the free DBA toolkit at [www.oriolen.com](http://www.oriolen.com) or email `paul.baumgartel@aya.yale.edu`

Oracle DBA list server at `www.lazydba.com` (click on ORACLE DBA FORUM)



## ***Appendix A: Quick reference***

### Prepare source database (export)

- Generate a list of tables to export
  - Exclude snapshot base tables
  - Exclude snapshot log tables
  - Exclude temporary tables
- Specify export options
  - Constraints = n
  - Indexes = n
  - Grants = n
  - Direct = y (Oracle 7.3 and later)

### Prepare target database

- Export structure (rows = n)
- Drop primary keys, unique keys, indexes
- Drop referential integrity constraints
- Disable triggers
- Generate sequence drop, create, grant scripts
- Truncate tables

### Import data

- Turn off log archiving
- Considering increasing size and/or number of redo logs
- Set all but largest rollback segment offline
- Import parameters
  - Ignore = y
  - Buffer = 10485760
  - Commit = y

### Restore target database state

- Import the database structure export, restoring:
  - Primary keys
  - Unique keys
  - Foreign keys
  - Indexes
- Enable triggers
- Drop and re-create sequences and their grants
- Execute complete refresh of all snapshots (if applicable)

## Appendix B: Scripts

### Prepare target database (pre-import)

#### **gen\_droprcon.sql**

This script generates a script to drop all referential constraints in the schema.

```
/*  
-----  
--gen_droprcon.sql  
--Paul Baumgartel, Adept Computer Associates, Inc.  
  
set pagesize 0 feedback off linesize 300 trimspool on  
select 'alter table '||table_name||' drop constraint  
'||constraint_name||  
'; ' from user_constraints where constraint_type = 'R'  
  
spool droprcon.sql  
/  
spool off
```

#### **gen\_distrig\_entrig.sql**

This script generates a script to disable all triggers on each table on the schema, and a script to enable all triggers.

```
/*  
-----  
--gen_distrig_entrig.sql  
--Paul Baumgartel, Adept Computer Associates, Inc.  
set pages 0  
set feedb off  
set lines 300  
set trimspool on  
select 'alter table '||table_name||' disable all triggers;'  
from tabs  
  
spool distrig.sql  
/  
spool off  
select 'alter table '||table_name||' enable all triggers;'  
from tabs  
  
spool entrig.sql  
/  
set feedb 8  
set pages 32  
spool off
```

#### **gen\_dropidx.sql**

This script generates a script to drop all indexes that are not associated with primary or unique keys.

```
/*  
-----  
--gen_dropidx.sql  
--Paul Baumgartel, Adept Computer Associates, Inc.  
set pagesize 0 feedback off linesize 300 trimspool on
```

```

select 'drop index '||index_name||';' from user_indexes t1 where
index_name
not like 'I_SNAP%' and not exists (select 1 from user_constraints t2
where t2.constraint_type in ('U','P')
and t2.constraint_name = t1.index_name)

```

```

spool dropidx.sql
/

```

### gen\_droppkuk.sql

This script generates a script to drop all primary and unique keys in the schema.

```

/*****/

```

```

--gen_droppkuk.sql
--Paul Baumgartel, Adept Computer Associates, Inc.

```

```

set pagesize 0 feedback off linesize 300 trimspool on
select 'alter table '||table_name||' drop constraint '||constraint_name
||';' from user_constraints where constraint_type in ('U','P')

```

```

spool droppkuk.sql
/

```

### regen\_seq.sql

This script generates three scripts: one to drop all sequences, one to create the sequences with a “start with” value based on the last value produced in the source database, and one to restore any existing grants on the sequences. The first and last scripts are produced with information from the target database, and the second script uses the source database data dictionary.

```

/*****/

```

```

-- regen_seq.sql
-- Paul Baumgartel, Adept Computer Associates, Inc.

```

```

-- Parameter: Name of database link to source database schema owner

```

```

set pages 0 feedb off verify off trimspool on
select 'grant select on '||table_name||' to '||grantee||';' from
user_tab_privs_made t1 where exists (select 1 from seq t2 where
t1.table_name = t2.sequence_name)

```

```

spool grantseq.sql
/

```

```

spool off
select 'drop sequence '||sequence_name||';' from seq

```

```

spool dropseq.sql
/

```

```

spool off
select 'create sequence '||sequence_name||chr(10)||
decode(min_value, null,' nominvalue ',' minvalue '||min_value)||
decode(max_value, null,' nomaxvalue ',' maxvalue '||max_value)||
' increment by '||increment_by||chr(10)||
' start with '||to_char(last_number+1)||
decode(cycle_flag, 'Y',' CYCLE ',' NOCYCLE ')||
decode(order_flag, 'Y',' ORDER ',' NOORDER ')||

```

```
decode(cache_size,0,' nocache ',' cache '||cache_size)||';'  
from seq@&1
```

```
spool creseq.sql  
/  
spool off
```

### **gen\_trunct.sql**

This script produces a script to truncate all tables in the schema, except for snapshot base tables.

```
/*****  
--gen_trunct.sql  
--Paul Baumgartel, Adept Computer Associates, Inc.  
  
set feedback off pagesize 0  
select 'truncate table '||table_name||';' from tabs t1 where not exists  
(select 1 from user_snapshots t2 where t1.table_name = t2.table_name)  
  
spool trunct.sql  
/  
spool off
```

### **pre\_import.sql**

This script simply runs the separate pre-import scripts in the correct order. First, it runs the script-generating scripts, then the generated scripts themselves.

```
/*****  
-- pre_import.sql  
-- Paul Baumgartel, Adept Computer Associates, Inc.  
set termout off  
@gen_droprcon  
@gen_distrig_entrig  
@gen_dropidx  
@gen_droppkuk  
@gen_trunct  
@regen_seq &1  
set feedback 8 pagesize 36  
@droprcon  
@distrig  
@dropidx  
@droppkuk  
@trunct  
exit
```

### **pre\_import**

This shell script provides a convenient single command to perform all pre-import steps.

```
/*****  
#!/bin/sh  
# Arguments: Target database username, password  
# Name of database link to source database schema owner  
exp $1/$2 file=$1_struct.exp rows=n  
sqlplus $1/$2 @pre_import $3
```

## Restore target database (post-import)

### **gen\_refall.sql**

This script generates a script that performs individual complete refreshes of all snapshots in the schema.

```
/*  
--gen_refall.sql  
--Paul Baumgartel, Adept Computer Associates, Inc.  
  
set pagesize 0 feedback off trimspool on  
select 'exec dbms_snapshot.refresh(''|name|'|','C')' from  
user_snapshots  
  
spool refall.sql  
/
```

### **post\_import.sql**

This script runs the various post-import scripts that were generated before the import.

```
/*  
-- post_import.sql  
-- Paul Baumgartel, Adept Computer Associates, Inc.  
  
@entrig  
@dropseq  
@creseq  
@grantseq  
@refall  
exit
```

### **post\_import**

This shell script provides a convenient single command to perform all post-import steps.

```
/*  
#!/bin/sh  
# Parameters:  target database username, password  
imp $1/$2 file=$1_struct.exp ignore=y buffer=104858760 full=y  
sqlplus $1/$2 @post_import.sql
```

### **Author**

Paul Baumgartel is Principal Consultant at Adept Computer Associates, Inc. He has 18 years' experience in software engineering, systems integration, and Oracle database administration. You can contact him at [paul.baumgartel@aya.yale.edu](mailto:paul.baumgartel@aya.yale.edu).