MFG/PRO®

# System Administration Reference Guide

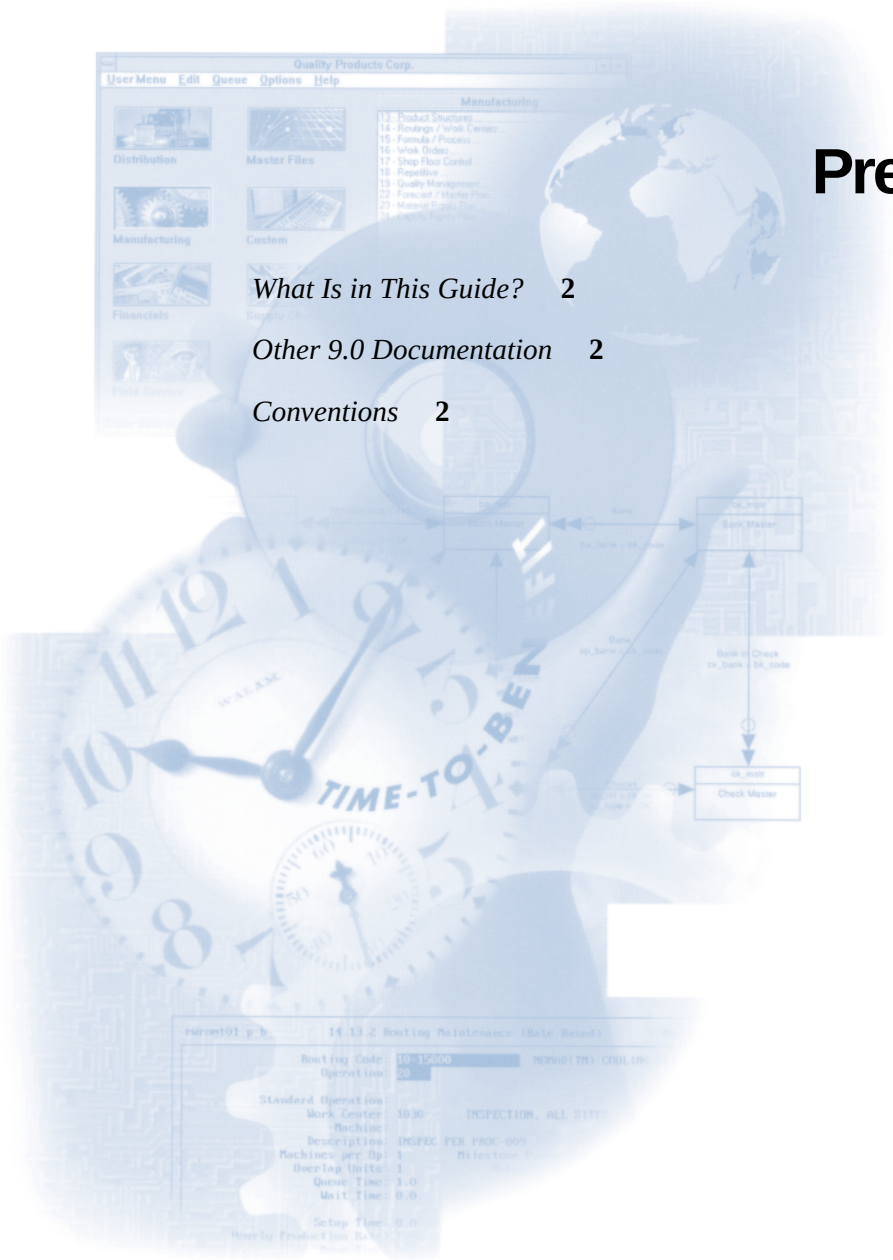
ORACLE DATABASE
ON UNIX SERVER

78-0465A
MFG/PRO Version 9.0
Printed in the U.S.A.
March 1999

78-0465A

**QAD Inc.**
6450 Via Real
Carpinteria, California 93013
Phone (805) 684-6614
Fax (805) 684-1890

# Contents




# Preface

## What Is in This Guide?

This guide covers system administration tasks for MFG/PRO on Oracle that take place outside the MFG/PRO application. Much of the material is familiar to an Oracle database administrator (DBA), but the focus in this guide is specific to the size and complexity of MFG/PRO.

***Important*** This guide does not cover Oracle-specific administration procedures. A basic requirement for all MFG/PRO on Oracle implementations is that a certified Oracle DBA is on site.

## Other 9.0 Documentation

- For software installation instructions, see the *MFG/PRO 9.0 Installation Guides*.
- For information on the entire system, see the *MFG/PRO 9.0 User Guides, Volumes 1–11* of the standard documentation.
- For information on system administration tasks within MFG/PRO, see the *MFG/PRO 9.0 User Guide, Volume 11, Manager Functions*.
- System administration training for MFG/PRO on Oracle is also available. Check with your distributor or your QAD training center.

## Conventions

MFG/PRO 9.0 is available in several interfaces: Windows, character, Web browser, and an interface for object-oriented programs. To standardize presentation, the documentation uses the following conventions:

- Illustrations of the software are in the Windows interface.
- References to keyboard commands are generic; for example, “press Go” refers to F2 in Windows and to F1 in the character-based interface. Use the following tables to identify the keyboard commands for a given interface.

## Windows Keyboard Commands

**Table 0.1** Windows Navigation Commands

| Navigation Commands | Keyboard Entry | Description |
|---|---|---|
| Go | F2 | Moves to next frame. |
| End | Esc | Exits a frame, program, or menu. |
| Previous | F9 or Up Arrow | Retrieves previous record in a key data field. |
| Next | F10 or Down Arrow | Retrieves next record in a key data field. |
| Enter | Enter | Moves to next field within a frame. |
| Tab | Tab | Moves to next field within a frame. |
| Back Tab | Shift+Tab | Moves back one field within a frame. |
| Exit | Alt+X | Closes a program. |
| Run | Ctrl+R | Starts a program by name. |
| Save (object) | F12 | Key frame switches to data entry; in data entry, saves and returns to key frame. |
| Print (object) | Ctrl+P | Prints browse or maintenance information. |

**Table 0.2** Windows Help Commands

| Help Commands | Keyboard Entry | Description |
|---|---|---|
| Field Help | F1 | Opens help on current field. |
| Procedure Help | Shift+F1 | Opens help on current program. |
| Browse | Alt+F1 | Displays choice of records. |
| Look-Up Browse | Alt+F2 | Displays choice of records. |
| About | Ctrl+F1 | Displays the program name. |
| Browse Options | F7 | Opens the browse options window. |
| Browse Options Toggle | Alt+F | Turns the browse options on and off. |
| Browse Graph | Shift+F11 | Opens the browse graphing window. |
| Field Name | Ctrl+F | Displays the field name. |

**Table 0.3** Windows Edit Commands

| Edit Commands | Keyboard Entry | Description |
|---|---|---|
| Delete Record | F5 | Deletes an open record. |
| Cut | Ctrl+X | Cuts a field or selection to clipboard. |
| Copy | Ctrl+C | Copies a field or selection to clipboard. |
| Paste | Ctrl+V | Pastes data from the clipboard. |

## Character Keyboard Commands

**Table 0.4** Character Navigation Commands

| Navigation Commands | Keyboard Entry | Control Key Entry | Description |
|---|---|---|---|
| Go | F1 | Ctrl+X | Moves to next frame. |
| End | F4 | Ctrl+E | Exits a frame, program, or menu. |
| User Menu | F6 | Ctrl+P | Displays list of user programs. |
| Previous | F9 or Up Arrow | Ctrl+K | Scroll up in key fields and browses. |
| Next | F10, Dn Arrow | Ctrl+J | Scroll down in key fields and browses. |
| Enter | Enter | | Moves to next field within a frame. |
| Tab | Tab | | Moves to next field within a frame. |
| Back Tab | Shift+Tab | Ctrl+U | Moves back one field within a frame. |
| Menu Bar | Esc, M | | Accesses the menu bar. |
| Save (object) | F12 | | Key frame, moves to data entry; data entry, saves & returns to key frame |
| Print (object) | Ctrl+P | | Prints browse or maintenance info. |

**Table 0.5** Character Help Commands

| Help Commands | Keyboard Entry | Control Key Entry | Description |
|---|---|---|---|
| Field Help | F2 | Ctrl+W | Opens help on current field. |
| Procedure Help | F2 | Ctrl+W | Opens help on current program. |
| Look-Up Browse | F2 | Ctrl+W | Displays choice of records. |
| Browse Opts | F7 | | Opens the browse options window. |
| Browse Toggle | Alt+F | | Turns the browse options on and off. |
| Field Name | Ctrl+F | Ctrl+F | Displays the field name. |

**Table 0.6** Character Edit Commands

| Edit Commands | Keyboard Entry | Control Key Entry | Description |
|---|---|---|---|
| Insert | F3 | Ctrl+T | Enables text insertion. |
| Delete Record | F5 | Ctrl+D | Deletes an open record. |
| Recall | F7 | Ctrl+R | Recalls last saved value in a field. |
| Cut | F8 | | Clears a field. |
| Copy | F11 | Ctrl+B | Copies a field. |
| Paste | F11 | Ctrl+B | Inserts value that you copied. |
| Multiple Copy | F12 | Ctrl+A | Copies fields; pastes into new record. |
| Clear Date | Shift+? | | Clears the value in date fields. |

CHAPTER 1

# Introduction

MFG/PRO on Oracle provides the advantages of QAD's flagship production, distribution, and financial solutions on the Oracle relational database management system (RDBMS)—the most popular and flexible relational database on the market today.

System administration for MFG/PRO on Oracle is a critical set of tasks. This guide provides you with the specific procedures to follow to manage MFG/PRO code with the data on an Oracle database and to tune the database to obtain the best results.

# Organization of This Book

Each chapter is a self-sufficient reference section.

## Chapter 2, Components and Configurations

This chapter provides an overview of the software required to run MFG/PRO on Oracle—Oracle, PROGRESS, and MFG/PRO—and the possible deployment of that software using basic client/server hardware configurations.

## Chapter 3, Database Management

This chapter covers the tasks required to set up an administrative environment prior to operation, including Oracle management tools, an overview of MFG/PRO on Oracle security, database object and storage administration, and database backups.

## Chapter 4, Administration Utilities

The utilities chapter includes UNIX cron, QAD's Q/ADMIN toolset, compile procedures, and maintaining startup and shutdown scripts and database sets.

## Chapter 5, Performance Tuning

Performance tuning focuses on Oracle tuning procedures for the MFG/PRO application. This includes relevant tuning areas, application tuning, client/server tuning, memory management, tuning disk I/O, resource contention, and relevant init<SID>.ora parameters.

## Chapter 6, Customizations

This chapter covers the methods for porting custom applications (code and databases) from PROGRESS to Oracle, and the procedures recommended for managing custom code in an Oracle environment.

## Appendixes

There are three appendixes. The first is a summary of useful UNIX and vi editor commands. The other two are sample script files used in an MFG/PRO on Oracle implementation and useful performance monitoring tools for Oracle environments.

CHAPTER 2

# Components and Configurations

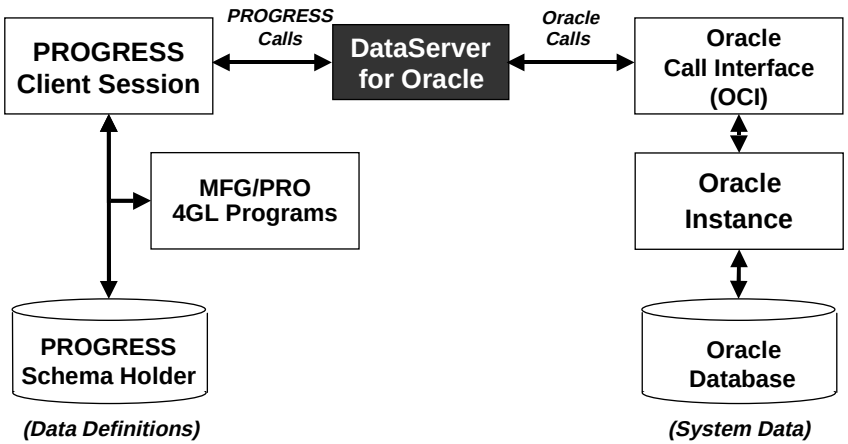
MFG/PRO on Oracle relies on the interaction of several components. These components have specific and complex tasks. Each component is either installed or built, or both. This chapter describes the critical tasks and relationships each component has and, where relevant, how and when that component is built.

# MFG/PRO on Oracle

MFG/PRO on Oracle is an Oracle-based application. MFG/PRO on Oracle runs MFG/PRO programs in the PROGRESS 4GL, but stores and retrieves all data using an Oracle database employing native Oracle data types. The link between PROGRESS and Oracle is the PROGRESS DataServer for Oracle. The DataServer uses native Oracle SQL calls to access the Oracle database just like an application written with Oracle development tools.

The PROGRESS DataServer for Oracle is also an Oracle-based application. The DataServer works in conjunction with a PROGRESS schema holder. The schema holder contains database definitions, but no data. The database definitions map PROGRESS database characteristics to appropriate Oracle schema objects.

**Fig. 2.1**
Basic MFG/PRO on Oracle Components




Any application that accesses Oracle can access the data that MFG/PRO uses in its application, subject to any Oracle-based security constraints. For example, a custom report program developed using Oracle development tools can read the MFG/PRO sales order table, assuming the Oracle database administrator has granted access permission to the user running that report. As with any integrated application, MFG/PRO data should not be updated from non-MFG/PRO programs. Data integrity is almost always lost.

For example, if an external program were to add confirmed sales orders to the MFG/PRO database without making the needed updates to general allocations, a mismatch would occur between sales orders and allocations.

# MFG/PRO Application

MFG/PRO is the application software you use to manage and track the operation and profitability of your facilities. The source code is the same whether you are running on PROGRESS or on Oracle databases. The code you use to run MFG/PRO relies on the database schema during compile to correctly identify the data structures it needs during operation.

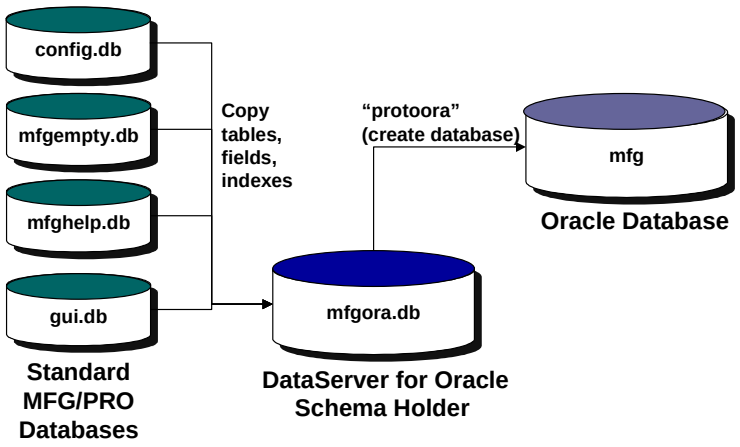
## MFG/PRO Schema

The MFG/PRO schemas are the data structures required for application and facility data that are created and used by MFG/PRO.

In MFG/PRO on PROGRESS installations, each schema represents a PROGRESS database and several interrelated databases are shipped and installed. For example, along with the main schema (mfgempty), there are schemas that contain the help tables (mfghelp) and the graphical interface components (gui). All three of these schemas are required to support an MFG/PRO session.

Oracle allows a great deal more control than PROGRESS over database storage across multiple disks. This enables one Oracle database to grow almost infinitely and removes the need to segregate schemas. Therefore, MFG/PRO schemas for an Oracle installation are combined into a single schema holder and a single Oracle database.

**Fig. 2.2**
Standard MFG/PRO PROGRESS Databases Combined to Create One Schema Holder and Oracle Database




This combining of multiple schemas into a single schema holder is performed by the PROGRESS DataServer. Special SQL scripts are created by the DataServer during the installation process at your site to create the Oracle database from definitions in the schema holder.

## PROGRESS 4GL

PROGRESS 4GL software is required for MFG/PRO to run and to connect with the required databases. The PROGRESS 4GL provides the software environment required to run a PROGRESS application.

The PROGRESS 4GL was designed for and with PROGRESS databases. The PROGRESS language (and therefore applications written in PROGRESS such as MFG/PRO) includes functionality not supported by an Oracle database. These include:

- Case-insensitive data entry and retrieval
- Sub-transaction scoping
- SHARE-LOCKs and NO-LOCKs
- Extended data types—LOGICAL, DECIMAL, INTEGER
- Arrays
- Unknown values
- Zero-length character strings
- Scrolling (FIND FIRST, FIND PREV, etc.)

## PROGRESS DataServer

The PROGRESS Oracle DataServer is a PROGRESS product that enables 4GL clients to access Oracle databases. Most 4GL statements and functions work the same way in both PROGRESS and Oracle database environments. PROGRESS data dictionary features are supported, including labels, format, validation, and 4GL triggers.

See the *Oracle DataServer Guide*, Table 2-10, page 2-35 for the differences between native PROGRESS databases and the Oracle DataServer.

For MFG/PRO on Oracle, the DataServer fulfills two basic requirements. During installation, it builds the MFG/PRO schema holder from the various MFG/PRO schemas. Then, during operation, the DataServer manages the transaction flow between MFG/PRO's PROGRESS 4GL and the Oracle Call Interface (OCI).

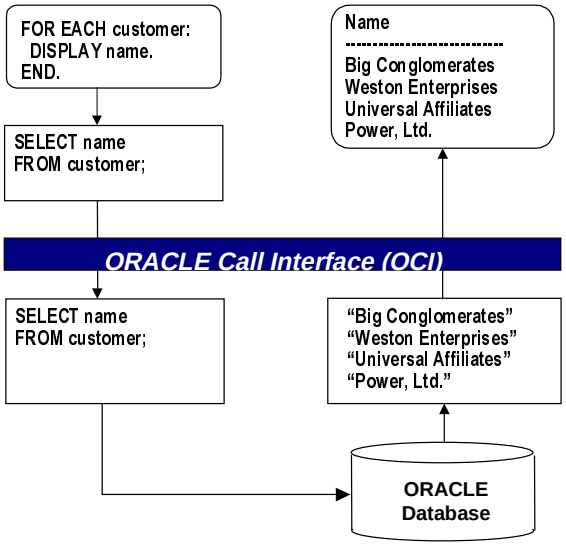
### DataServer Transactions

During operation of MFG/PRO, the DataServer relies on underlying schema structures to communicate between the PROGRESS client session and the Oracle database.

Initially, the PROGRESS client uses schema holder objects to create DataServer calls. The DataServer translates these calls into SQL and passes them to the OCI. The OCI then sends the SQL code to the Oracle instance. SQL results are created by the Oracle instance and passed back

through the OCI to the DataServer. The DataServer formats the results and returns them to the client. Figure 2.3 illustrates this interaction.

**Fig. 2.3** DataServer Operation




## Schema Holders

The schema holder is a small PROGRESS database that allows for PROGRESS dictionary functionality: labels, formats, initial values, descriptions, validations, 4GL triggers, and so on. The schema holder also stores Oracle information necessary to generate SQL statements.

The schema holder contains only the schema definitions—tables, indexes, and fields for all the different schemas normally required to support one MFG/PRO session. It contains no data. The schema holder's function is twofold. It provides a set of data definitions during code compiles and it maps PROGRESS data structures to compatible Oracle structures.

You can compile with only the schema holder connected. You only need to connect to Oracle in order to run programs.

The duplication of schemas, once in the schema holder and once in the Oracle database, means each PROGRESS schema holder contains two names. A physical database name for the schema holder of `qad`, and an

internal database name for the Oracle database, called the foreign database reference, of `qaddb`.

***Note*** Because MFG/PRO uses the logical names `qad` for the PROGRESS schema holder and `qaddb` for the Oracle database logical reference, the Oracle database name cannot be qad or qaddb.

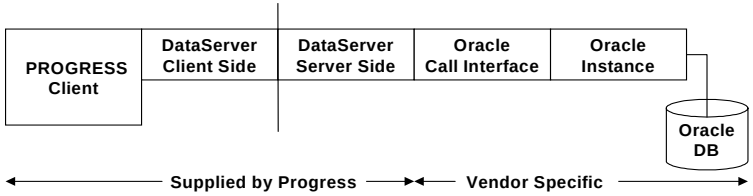
Schema holders should always be used in read-only mode. This does not prevent you from updating the Oracle database; it merely protects the schema from inadvertent modifications. You can store the schema holder on a file server and have read-only connections over a network to it.

## Oracle DataServer Structure

Oracle DataServer software has 2 parts:

- Client side—delivered with any PROGRESS client product
- Server side—purchased as the Oracle DataServer

**Fig. 2.4** DataServer Structure




### Build

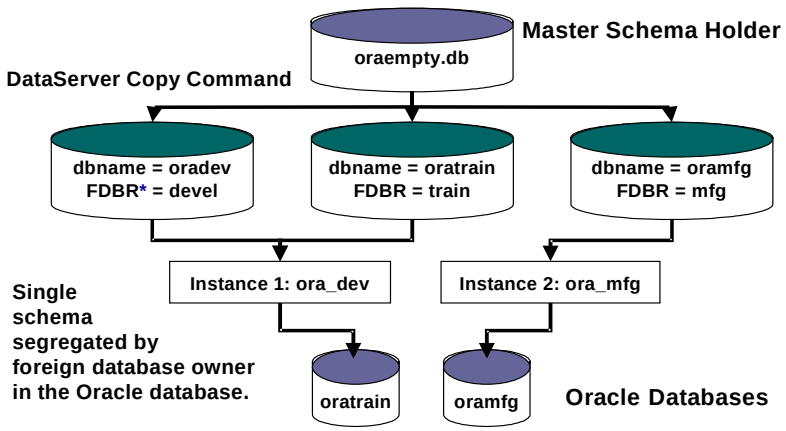
When you build a new schema holder, you are building a master schema holder to compile against and to copy from. The foreign database name is the one used during compiles. When you compile the application code, PROGRESS uses the foreign database reference of `qaddb` to create accurate database pointers in the compiled application code (r-code). For example, a call to the ih_due_date field in the ih_hist table, would be compiled as qaddb.ih_hist.ih_due_date.

## Copy

Most installations require three or more schema holders—production, development, and training—each generated from the same schema holder. To accomplish this, you copy the master schema holder, giving each new schema holder a new physical and foreign database name.

Figure 2.5 illustrates the process of copying the master schema holder with new names and foreign database references for multiple schema holder environments.

**Fig. 2.5**
Copying the Master Schema Holder




* FDBR = Foreign Database Reference

After copying your schema holder, store the master schema holder in a safe disk or tape location.

When you connect with a database during MFG/PRO startup, you connect to both the schema holder and the foreign database reference. The foreign database reference becomes the alias you use to access the Oracle database. After installation, the schema holder and Oracle database are connected through startup scripts.

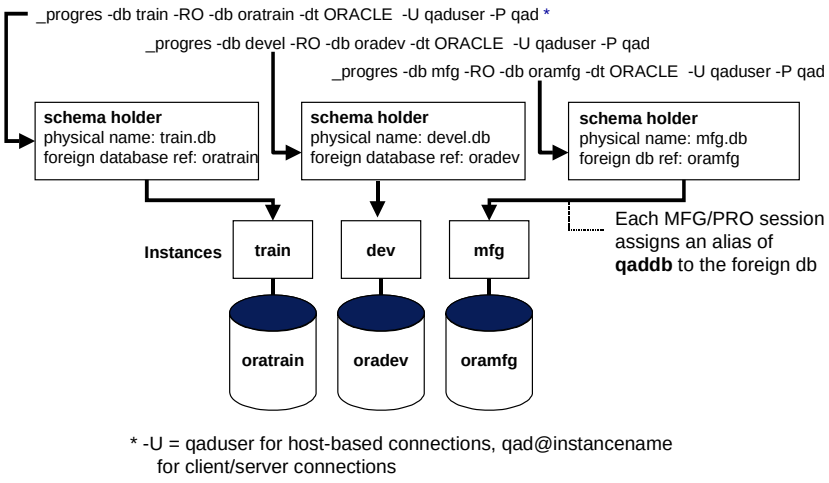



**Fig. 2.6**
Oracle Database and Schema Holder Interface

Figure 2.6 shows three schema holders for three purposes–training, development, and production–each interacting with the Oracle database through separate instances. The `qaddb` referenced in all MFG/PRO compiled code is mapped during connection to the database to `oratrain`, `oradev`, or `oramfg` respectively.

If you run multiple production databases from a single physical disk location, you can connect all of them during a single MFG/PRO session. However, like multiple databases under PROGRESS, you can assign only one alias at a time. You must also assign separate foreign database owners (for example, `qad1`, `qad2`, `qad3` as in Figure 2.7) during the schema holder copy. You can then switch databases during the active session. In this case schema holders and therefore Oracle data ownership.

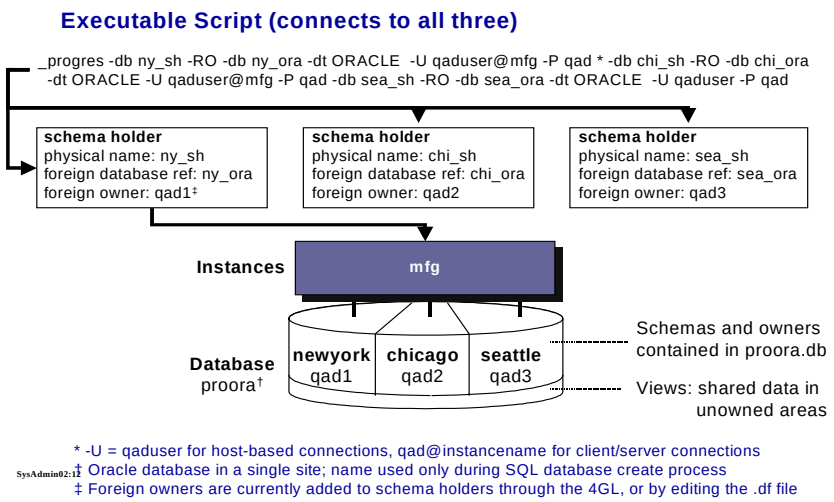



**Fig. 2.7** Multiple Production Databases Under a Single Oracle Instance

### Update

Use a single-user connection when you update the schema holder. Whenever the schema holder has to be updated:

1. Update a copy of the schema holder.
2. Truncate the .bi (before image) file.
3. Copy the new schema holder over the old.
4. Have users restart their PROGRESS sessions.

## Schema Holder Definitions

The schema holder is used by the DataServer to generate the SQL scripts that create the MFG/PRO Oracle database. The schema holder contains the tables, fields and indexes that are created. But these objects are not identical in PROGRESS and Oracle.

Each MFG/PRO table built in Oracle has the same structure as a PROGRESS table except that additional fields are added to the Oracle tables to enable PROGRESS code to use case-insensitive indexing on an Oracle database and to support the progress_recid.

### Case Insensitivity

In PROGRESS applications, and in their databases, text is case insensitive, which allows for faster, more flexible record lookups. For example, in PROGRESS, if you create a new customer record by typing in BEST CUSTOMER, this is exactly what is stored. If you are running a browse against Best Customer or best customer, it will find BEST CUSTOMER even though the match is not the exact letter case, because PROGRESS is case insensitive.

In Oracle databases, by default you need to enter BEST CUSTOMER in a lookup function to find that record again.

To correct this issue, the schema holder generates a shadow column (U##) preceding each indexed character-format column in the schema. When a value is written to the character-format index column, a shadow entry in all uppercase letters is automatically created in the shadow column.

Inquiries, reports, and browses use the shadow column for lookups, converting all user-entered lookup parameters into uppercase. Then, for each record found, the value in the non-shadow column is returned.

Third-party products working with an MFG/PRO on Oracle database should use U## columns in queries instead of the standard columns.

For example, the MFG/PRO budget detail table format in Oracle contains the columns `U##bgd_project` and `bgd_project`. During record creation, a project name entered as Reassembly is stored in `U##bgd_project` as REASSEMBLY and in `bgd_project` as Reassembly. During a lookup, Oracle uses the `U##bgd_project` column (REASSEMBLY), and returns the equivalent value from `bgd_project` (Reassembly).

Oracle indexes are also built using these U## column-name columns in place of the original (case-insensitive) columns. This ensures that values in the Oracle tables are accessible to either case sensitive or case insensitive searches through Oracle utilities or the PROGRESS DataServer.

### recid and Oracle Sequences

The recid (record ID) is a default database field in PROGRESS databases. The recid field maintains a sequential, numeric value for every PROGRESS table. This value makes scrolling cursors possible and supports record sorting, especially on nonunique key searches. To support the progress_recid function, the DataServer adds a progress_recid column to the end of each Oracle table. This column is also built into its own unique index and added to the end of every nonunique index.

**Tip**
The progress_recid value is unique to the database in a pure PROGRESS environment.

The progress_recid value is unique to each Oracle table but not to the database and is based on the sequence of record creation. That sequence number is determined by the value of an Oracle object, called a sequence, which is built for every MFG/PRO table in the database. A table's sequence is a named object, which has its initial value, increment, and number of values to store in cache memory specified at creation time. For example, the sequence `abd_det_seq` is built for the `abd_det` table on Oracle:

```
CREATE SEQUENCE abd_det_seq START WITH 1 INCREMENT BY 1
CACHE 75
```

This sequence begins at one (1) and is increased in value by one each time a record is written to the table. System performance is improved by caching the next 75 sequence numbers in memory because the next range of sequence numbers are not read from disk until all the numbers in the cache have been used–75 records have been written.

## DataServer Parameters

The Oracle DataServer uses several run-time parameters that impact the operation of MFG/PRO. The following three parameters should be set as recommended.

**Table 2.1** Recommended Dataserver Parameters

| Parameter | Description | Recommendation |
|---|---|---|
| -noindexhint. | Disable Oracle hint generation by default. | Not recommended. |
| -nojoinbysqldb. | Disable default join by SQL server. | Not recommended. |
| -znotrim. | Disable trimming of character arguments. | Required by MFG/PRO on Oracle. |

### Connection Parameters

When you use the Oracle DataServer, two database connections are required.

1. To the schema holder itself.

   This is a PROGRESS database, and accepts normal PROGRESS database connection parameters.

2. To the Oracle database.

   Valid parameters are:

   - -db–DB connection separator/physical DB name
   - -ld–logical DB name
   - -dt–DB type
   - -c–max # of cursors
   - -Dsrv–DataServer options

***Note*** A read-only connection to the schema holder does not prevent you from performing updates against Oracle.

3. In a read-only database such as the schema holder, the .bi (before-image) file should always be zero (0) bytes. If you start the schema holder without the `-RO` parameter, the database updates the .bi. To correct this, truncate the bi of a PROGRESS schema holder with:

   ```
   proutil dbname -C truncate bi
   ```

## Physical/Logical DB Name for Oracle (-db/-ld)

Each schema holder has one or more non-PROGRESS database definitions. Each of these is an Oracle schema. Each is anchored by the logical DB name. The physical DB name is ignored for Oracle. Oracle OCI/SQL*Net finds which database to connect to.

Use the -db parameter to set physical and logical dbnames, and do not use -ld. For example, an MFG/PRO schema holder is usually called `oraempty`, while the Oracle schema in it has a logical dbname of `qaddb`.

## Database Type (-dt)

This parameter is not required since the database type is already on the schema holder. If used, it should be set to Oracle (case insensitive). It can help to catch connect parameter typing errors.

## Max number of cursors (-c)

This parameter defines the maximum number of cursors the Oracle DataServer uses before reusing existing ones. Too few can cause wasted time parsing SQL statements. Too many can waste memory. Between 200 to 400 is usually appropriate. The -c value must be smaller than the Oracle `open_cursors` parameter. First set the Oracle `open_cursors` to the desired value, then set -c to 10 less or 5% less than the `open_cursors` value, whichever is larger.

## DataServer options (-Dsrv)

The -Dsrv parameters define defaults for query tuning options such as debugging info and how you want PROGRESS to interact with Oracle. Chain all -Dsrv parameters with a comma.

See the *Oracle DataServer Guide*, page 2-50.

- qt_debug, <extended/sql/decimal code>
- qt_lookahead/qt_no_lookahead
- qt_cache_size, size, <qt_row>
- qt_bind_where/qt_no_bind_where

***Example***

```
-Dsrv qt_debug,extended, qt_lookahead,qt_cache_size,
   20,qt_row
```

# Oracle Components

The Oracle components of an MFG/PRO on Oracle installation include the following.

- Oracle database
- Oracle instance
- Oracle Call Interface (OCI)
- SQL*Net

The instance and database form a cohesive unit. The OCI is the interface to any non-Oracle application language. SQL*Net enables client/server transactions.

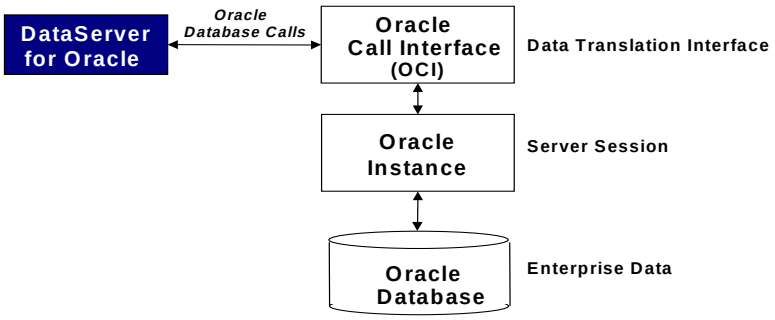
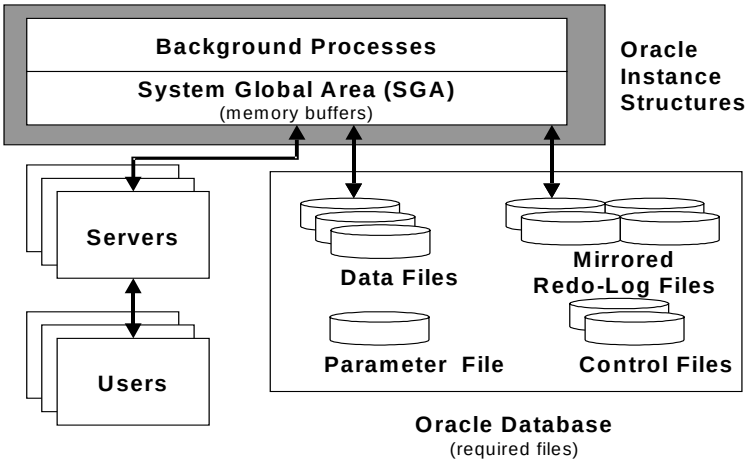



**Fig. 2.8**
DataServer with Oracle Components

## Oracle Instances

The Oracle instance is a combination of shared memory structures and active database processes on the database host machine. The instance can be roughly divided into background processes and the system global area (SGA).

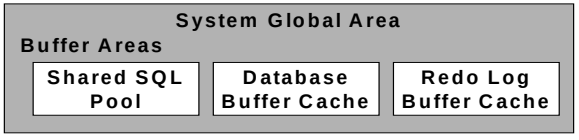
**Fig. 2.9**
Relationship Between the Instance and the Oracle Database




One set of background processes exists in the instance and provides services for all user (client) sessions. Background processes are responsible for database I/O, maintaining data integrity, and optimizing system performance.

The system global area is a set of buffer areas that Oracle processes use to store and manipulate data.

**Fig. 2.10**
SGA Components



## System Global Area (SGA)

The SGA consists minimally of a shared SQL pool, a database buffer cache, and a redo log buffer. Several processes associated with these memory management sectors communicate between the SGA and the database. A critical aspect of MFG/PRO on Oracle is sizing the SGA appropriately.

See "Memory Tuning" on page 79.

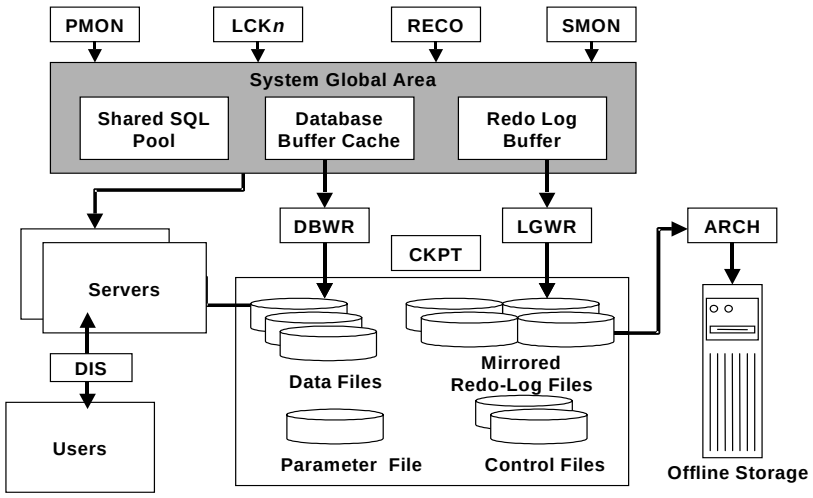



**Fig. 2.11** The Interaction of SGA Processes in an Oracle Environment

Processes run from the SGA include:

**Table 2.2** Processes Run From the SGA

| Process | Name | Description |
|---|---|---|
| DBWR | Database Writer | Writes modified data blocks from the database buffer cache to the physical data files. |
| LGWR | Log Writer | Writes redo log entries from the redo-log buffers to the redo-log files. |
| PMON | Process Monitor | Cleans the buffer cache, frees resources when user processes fail, and restarts failed server processes. |
| SMON | System Monitor | Performs instance recovery at instance startup and cleans up temporary segments (i.e., sort space) no longer in use. |

*SGA Processes continued on next page*

| Process | Name | Description |
|---|---|---|
| Dnnn | Dispatcher | In a multiuser environment, routes requests from user processes to shared server processes and returns responses back to appropriate user processes. Allows many clients to share a small number of servers. One dispatcher per communication protocol in use (D000 to Dnnn). |
| Snnn | Shared Server | Processes user requests against the database. Can be shared by more than one client. There can be multiple active server processes (S000 to Snnn). |
| ARCH | Archiver | Copies online redo-log files to archived, off-line storage (other disk files or tape) when the online redo-log file becomes full. |
| CKPT | Checkpointer | Signals DBWR at specific times (at set checkpoint intervals) to write all modified data buffers in the SGA to disk. |
| LCKn | Lock Process | Used only in multiple instance configurations so one instance can lock rows in another instance's database, preventing multiple instances from updating the same row simultaneously. There may be several lock processes running. |
| RECO | Distributed Transaction Recoverer | Commits or rolls back transactions pending due to a system or network failure in a distributed database configuration (instances running on separate machines). |
| SNPn | Sanpshot Process | Allows the replication of a master table to instances running on other machines in a distributed database configurations. |

***Note*** DBWR, LGWR, PMON, and SMON processes are required for Oracle to run. The others are optional.

### Oracle vs PROGRESS Processes and Files

The processes described above have counterparts in PROGRESS. If you are familiar with the PROGRESS environments, a comparison may be useful.

**Table 2.3** PROGRESS and Oracle Structure Comparison

| Structure | PROGRESS | Oracle |
|---|---|---|
| Database data files | databasename.db | Any name; conventional extensions include .ts and .dbf. |
| Database log files | databasename.lg | alert_<SID>.log and process trace (.trc) files located in the directory specified by the `background_dump_dest` parameter in the init<SID>.ora parameter file. |
| Before-image files | databasename.bi | Rollback segments (logical structures named in the init<SID>.ora file). Oracle uses these segments to undo, or roll back, uncommitted (or incomplete) transactions. |
| After-image files | databasename.ai | Redo logs and archived redo-log files (on tape or disk). |
| Parameter file | username.pf | init<SID>.ora |
| Dumped data files | filename.d | tablename.exp |
| Broker process | _mprosrv | Dnnn |
| Server process | _mprosrv | Snnn |
| Data buffer writer | APW or _mprosrv | DBWR |
| Before-image buffer writer | BIW or _mprosrv | Snnn |
| After-image writer | AIW | LGWR |
| Watchdog process | _mprshut db -C watchdog | PMON & SMON |

## Oracle Databases

Oracle databases are made up of several types of physical files that comprise the database. These include data files, redo-log files, control files, and a startup parameter file. Data storage objects, of which data files are one, make up the main structure of an Oracle database, and are either physical or logical. Each Oracle database object has ownership attributes.

### Physical Files

*Data files.* These contain all the database data. There is no standard naming scheme, but conventional name extensions are .dbf (for database file) or .ts (for tablespace). Example: mfg01m74h.dbf.

*Redo-log files.* These contain a record of all changes made to data. These files are used to recover a database from a failure that prevents data from being written to the datafiles (used in roll-forward recovery). Redo-log files can be mirrored to separate disks for added safety. There is no standard naming scheme, but they commonly begin with *log* or have an extension of .rdo. Example: log01m74h.dbf or m74hlog01.rdo.

*Control file.* This is a binary file that records the physical structure of the database. It contains the database name, names and locations of all data files and redo-log files, and a timestamp of database creation. The control file is used during database start-up so Oracle can identify files that must be opened. This file can also be mirrored for protection. There is no standard naming scheme, but these files commonly have a .ctl extension. Example: ctrl1m74h.ctl.

*Parameter file.* This file contains a list of instance configuration parameters. This is read at database start-up to initialize memory and process settings. Among other things, it tells Oracle the name of the database, the location of the control file(s), how much memory to use for structures in the SGA, and the names of rollback segments in the database. Usually named init<SID>.ora , where <SID> is the Oracle system ID (instance name). Example: initm74h.ora.

These files are opened and closed sequentially during start-up and shutdown of the Oracle database.

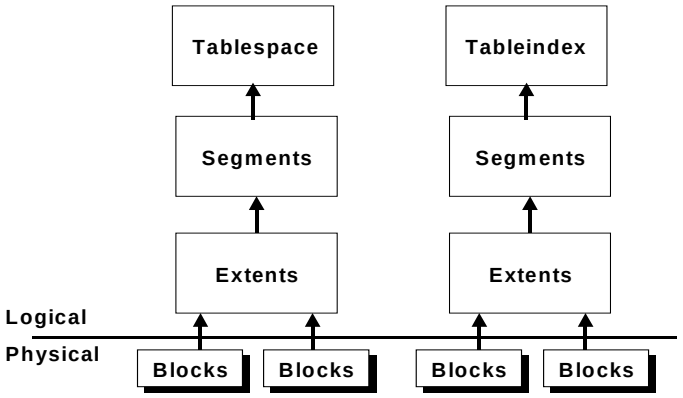
## Data Storage Structures

Oracle databases also consist of logical structures that combine with the physical files to act as storage structures.

**Table 2.4** Oracle Database Storage Structures

| Structure | Logical or Physical | Description |
|---|---|---|
| Database | Physical & Logical | A collection of shared data stored in tablespaces. |
| Tablespace | Logical | Repository for physically grouped data. Each tablespace is comprised of at least one data file (and one or more segments). (See Appendix A of the *MFG/PRO Installation Guide–Oracle Database on UNIX Server* for default tablespaces created by MFG/PRO.) |
| Data file | Physical | File belonging to a single tablespace. Data files reside on the host system (i.e., UNIX) disks. Each data file belongs to one and only one tablespace (tablespaces can contain multiple data files). Data files physically store database data. |
| Segment | Logical | Contains all the data for a specific structure within a tablespace. Each segment is owned by one and only one tablespace (although tablespaces usually have many segments). Individual tables and indexes are examples of segments. Segments are comprised of one or more extents. |
| Table | Logical | (Also called a data segment.) Contains rows (records) of data. Comprised of columns (fields). |
| Index | Logical | (Also called an index segment.) Contains a table's key columns (fields) to assist in row retrieval. |
| Extent | Logical | A set of contiguous database blocks. Although the blocks are physically part of a data file, an extent is a logical division. Segments are assigned a maximum number of extents into which they can grow. Each extent is assigned a number of data blocks. When data in a segment fills the current extent, another extent is allocated to the segment. |
| Block | Physical | File blocks allocated from an existing data file. Blocks are read from disk into memory and vice versa. Oracle blocksize is OS-dependent and is typically 1k, 2k, or 4k. |

**Fig. 2.12**
Data Components Hierarchy in Oracle




Oracle components such as tablespaces, data files, and index and table segments each have ownership attributes. This allows the Oracle database to support multiple, distinct uses of a single database.

Additional considerations regarding the Oracle database in MFG/PRO implementations include the treatment of Oracle integrity constraints, the character set and language used, the Oracle user name, and the number of concurrent users.

### Integrity Constraints

Referential Oracle integrity constraints are not defined in the MFG/PRO on Oracle database. They are handled by MFG/PRO and the schema holder.

### Character Set and Language

The Oracle database is created with the default character set of `WE8ISO8859P1` and in U.S. English. Twenty-odd foreign languages are supported by MFG/PRO. If an MFG/PRO on Oracle installation is using a language other than U.S. English, the Oracle database must be created with the appropriate character set and must have the proper NLS parameters set in the init<SID>.ora.

### Oracle User Name

MFG/PRO application uses one single Oracle user name `qad` for all the user sessions. This does not permit tracking of different users within the Oracle database. For a workaround, see “Implementing Unique Oracle User Names” on page 43.

### Number of Concurrent Users

Oracle databases can support up to 5,000 concurrent MFG/PRO user sessions.

## Oracle Call Interface

The Oracle Call Interface (OCI) manages calls to Oracle and enables client-server architecture. The OCI passes the SQL code from the PROGRESS DataServer to the Oracle instance. The Oracle instance then draws the results from the database and passes them back to the OCI where the DataServer picks them up.

## SQL*Net

The last component in an MFG/PRO on Oracle installation is SQL*Net, an Oracle product that uses the local networking protocol (such as TCP/IP) to connect application layers between servers or between clients and servers.

# Configurations

There are three basic configurations for MFG/PRO on Oracle: host-based, client/server with a local DataServer, and client/server with a remote DataServer configuration.

**Fig. 2.13**
MFG/PRO on Oracle Using a Local DataServer

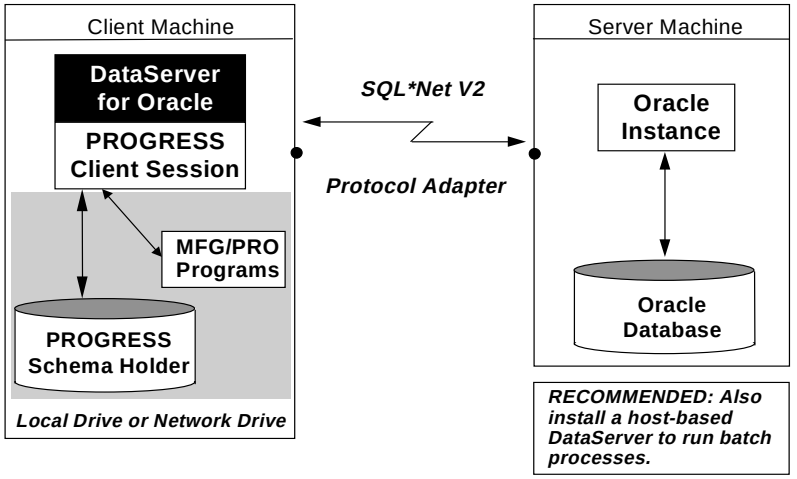
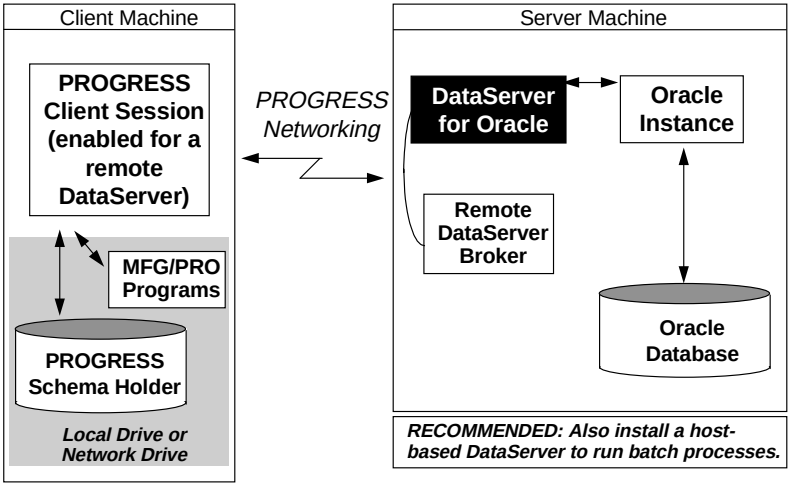



**Fig. 2.14**
MFG/PRO on Oracle Using a Remote DataServer




## UNIX Server with Host-Based Clients

In a host-based configuration, all components run on a single server. To accomplish this on a UNIX machine, the DataServer is built into the client executable during installation. The host-based configuration avoids network delays, but requires greater server capacity and size.

A host-based installation on UNIX assumes a character interface on the client terminals.

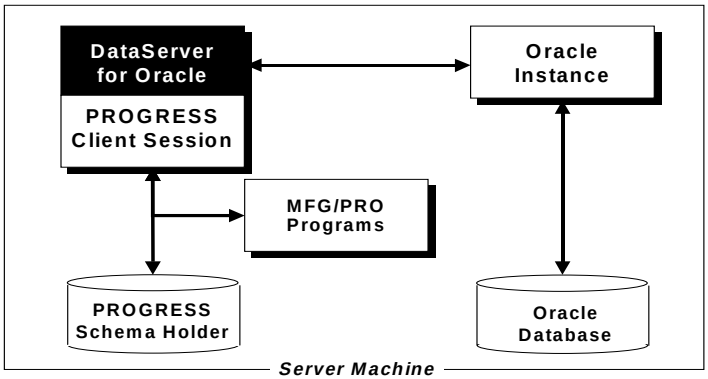



**Fig. 2.15**
Host-Based UNIX Configuration

## Host-Based Requirements

A UNIX host machine must contain MFG/PRO, PROGRESS, and Oracle software as follows.

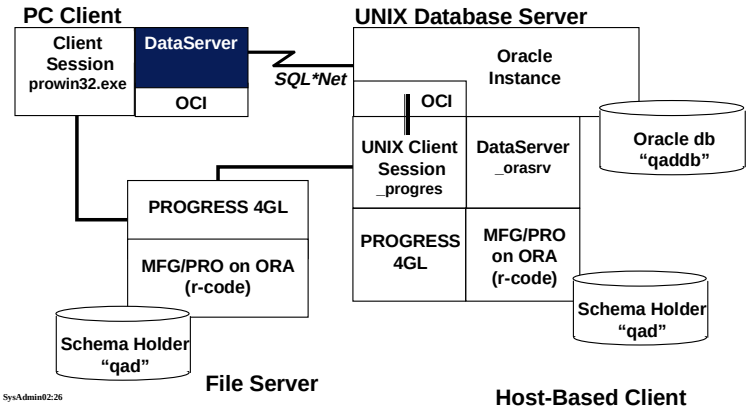
- MFG/PRO for Oracle, server media
- Full development license for Oracle RDBMS (verify compatibility with the PROGRESS DataServer)
- Oracle SQL*DBA or Oracle SQL*PLUS and Server Manager (SVRMGR)
- Oracle PRO*C Library (OCI)
- Oracle SQL*Net
- PROGRESS Oracle DataServer (check with PROGRESS or QAD sales representatives for the lettered release best suited to specific hardware platforms)
- Full development license for PROGRESS 4GL or ProVISION

Database servers can run on any major and most minor UNIX platforms as well as Windows NT. Plan a minimum of three gigabytes for storage. One gigabyte is required for MFG/PRO and PROGRESS and an additional 600 megabytes are required for each additional MFG/PRO language. If you maintain a source code cross-reference, an additional 5 megabytes are needed. Add disk space for enterprise data on top of this.

## UNIX Server with PC Clients (Local DataServer)

Running PC clients requires the installation of the DataServer on the clients. An additional file server is recommended to store program files and the schema holders. In addition, at least one host-based client should be implemented to run batch and other large MFG/PRO processes.

**Fig. 2.16**
Local DataServer on PC Clients with a File Server and Host-Based Client




### PC Client Requirements

The client machines required for Oracle under a client/server configuration are robust PCs running either Windows 95 or Windows NT 4.0. Client machines contain the MFG/PRO application and the DataServer. Having PROGRESS 4GL on the client machines (or on a networked file server) improves performance.

- MFG/PRO for Oracle, Windows media
- Query Runtime Version of PROGRESS. (This may be loaded on the file server.)
- PROGRESS or Oracle DataServer
- Oracle SQL*Net Client
- Oracle Call Interface library

PC client machines should be robust 150MHz or higher Pentium processors with a gigabyte or more of disk space, running Windows 95 or Windows NT 4.0 or better. In addition, the client should have:

- 32M RAM minimum

- Super VGA
- 32-bit network card

The host-based client may be a PC as specified here, or a telnet or other character-based client. The lower-end hardware is recommended for the batch processing role.

## File Server

For efficient installation and protection of source media, a fully networked file server is recommended. The file server should be the equivalent of one of the PC clients with the addition of a CD-ROM drive. You can optionally load PROGRESS on this machine with client access.

## NT Server with PC Clients (Host and Local)

Under NT, the configurations are roughly the same. The differences are that more hardware resources are required on the server and the installation process does not require that you build executables.

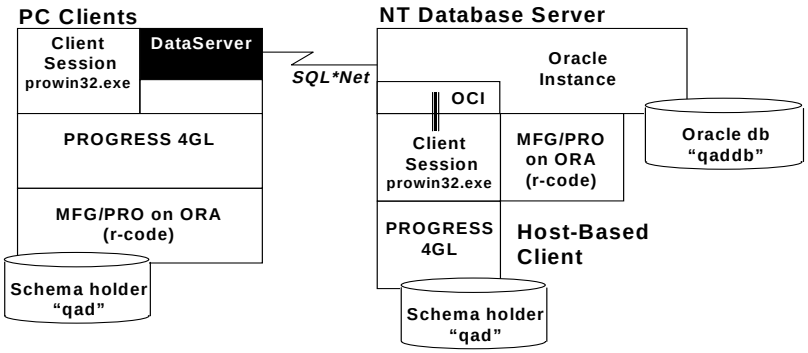



**Fig. 2.17** Local NT DataServer Without the File Server

CHAPTER 3

# Database Management

This chapter provides some insight into common administrative tasks in a MFG/PRO on Oracle system. The following topics are covered in this chapter:

# Oracle Database Administration

In general, use standard Oracle procedures for administering the database, including start-up, shutdown, and backup procedures. The PROGRESS schema holder database will not require as much administration, such as backups, because it does not contain data.

***Note*** If you do need to back up the schema holder, make sure no client sessions are running against it, truncate the before image file (extension `.bi`), and use UNIX commands to make a copy.

*Index Rebuilds*: As with any Oracle database, periodic index rebuilds are recommended, because the index tends to become fragmented.

*Application Security*: MFG/PRO on Oracle still relies on the system security available in PROGRESS.

*Database Security*: You can utilize any of the Oracle database security features in addition to the MFG/PRO security. For example, if you use other tools to access the Oracle database directly, you will probably want to implement Oracle database security.

*Upgrades*: Before you upgrade Oracle or PROGRESS, check to see if it is compatible with the other components. For example, if Oracle releases a new version, the PROGRESS DataServer may not immediately support it.

## Modifying the Oracle Database

If you are creating a new custom program, you may need to add tables or tablespaces to the Oracle database. However, you should *not* modify the standard MFG/PRO tables.

If you change Oracle, you must also change the PROGRESS schema holder. To do this, use the DataServer utility, Update/Add Oracle Table Definitions. This utility uses the objects in the Oracle schema to create the schema holder. You can then compile your custom code against the new schema holder.

# Oracle Enterprise Manager Overview

Oracle Enterprise Manager (OEMGR) is the preferred tool to administer MFG/PRO on Oracle. Reference is made to relevant components and features of OEMGR throughout this chapter.

Oracle Enterprise Manager consists of multiple components including the following.

### Console and Related Services

Provides the framework for Enterprise Manager components that manage and monitor the services in the network environment.

### Database Tools and Utilities

Provides easy-to-use tools for administering databases in the network.

### Administrator Toolbar

Provides quick access to the Oracle Enterprise Manager database administration tools.

### Intelligent Agents, Communication Daemon and Daemon Manager

Provide the communication link for the Console to manage and monitor services such as databases on nodes throughout the network.

### Additional Products

Additional Oracle products, such as the Performance Pack, that can be purchased to help manage your system.

### Online Help

Provides context-sensitive help for components of Oracle Enterprise Manager.

### Database Tools and Utilities

The database tool and utility applications are the primary administrative components of Oracle Enterprise Manager. Use these applications to perform most of your DBA administration tasks. The tools and utilities include:

- Security Manager
- Storage Manager
- Schema Manager
- Data Manager
- Backup Manager
- Instance Manager
- Server Manager Line Mode
- Software Manager
- SQL Worksheet

You can launch these tools through separate applications in the Console Tools menu or launch palette on objects selected in the Navigator or Map. Some of the functions of these applications can also be performed from the Navigator; however, the full set of functions is accessed through each application.

## MFG/PRO Security Overview

In an application such as MFG/PRO on Oracle, access to different parts of the application and to data must be controlled to prevent inadvertent or malicious modifications to vital company data. Both Oracle and MFG/PRO have features that enable control of access. All MFG/PRO users have their own login name and password for use on the MFG/PRO login screen. Their actions are logged by MFG/PRO into audit records.

However, each MFG/PRO user connects to the Oracle database as a single Oracle user named `qad`. This makes it difficult for Oracle database administrators to identify users individually. Additionally, the Oracle method of having the operating system identify users (OPS$ login), is not available in MFG/PRO.

The current solution to this problem is to custom implement unique Oracle user names for all MFG/PRO users (see “Implementing Unique Oracle User Names” on page 43). This allows Oracle database administrators to identify users individually through database monitoring tools.

MFG/PRO provides native support for unique Oracle user names.

### Oracle Object Ownership

In Oracle, when a user creates an object, such as a table, the user becomes the owner of the object and only the owner can access the object. Other users can access the object only if they have been granted privileges to do so. In MFG/PRO all database objects are owned by the QAD user account created during installation. The MFG/PRO application manages access to and sharing of database objects among different users.

## Implementing Unique Oracle User Names

Although MFG/PRO creates and uses only one Oracle user name, `qad`, for all MFG/PRO users, a custom solution can be implemented to create unique Oracle user names. Unique Oracle user names allow DBAs to track individual MFG/PRO users within the Oracle database.

To create unique Oracle user names, the following additional steps are necessary after installation of MFG/PRO:

1. Create a database Role with the necessary Oracle privileges.
2. Create a unique Oracle user name for each MFG/PRO user.
3. Grant the Role to the Oracle users.
4. Set up security so that the MFG/PRO user will not know his Oracle database password and is not able to access the database explicitly.
5. Create a .pf file for each user.

These steps are outlined in following section.

## Create a Database Role

In Oracle, Role is a named group of privileges. Create a Role QADUSER. Grant select, insert, update, delete permissions on all the objects owned by the qad user to Role QADUSER. This Role can be used to grant necessary privileges to each MFG/PRO user. Since there is no direct command to create a role, the following script can be used for this purpose.

Run the following at the UNIX prompt:

```
unix >sqlplus -s qad/qad <qaduser.sql
```

***Note*** SQL*Plus must connect to Oracle database as QAD user.

where `qaduser.sql` script is:

```
-- qaduser.sql - Script to create QADUSER Role.
set head off heading off echo off
set pagesize 0 verify off feedback off
drop role QADUSER cascade;
create role QADUSER;
spool qadpriv.sql
select 'grant all on ' ||table_name || ' to QADUSER;' from
user_tables;
select 'grant all on ' ||sequence_name ||
       ' to QADUSER;'
from user_sequences;
spool off
@qadpriv.sql
quit
```

## Create Oracle Users and Grant Privileges

Create a unique Oracle user account for each MFG/PRO user by using the `create user`... command. The password can be randomly assigned word and letter combinations known only to the security administrator and not given to users. Grant connect privilege to this account. Grant role QADUSER to the newly created user. Create a .pf file for each user.

### Set Up Additional Security

Additional security setup is necessary so that users are not able to view .pf files (which have the Oracle password listed) and users cannot run SQL*Plus, etc. The PROGRESS user should be able to read the .pf file and this can be achieved by setting root ID. Limiting access to SQL*Plus can be achieved by using the `product_user_profile` table.

- See your SQL*Plus documentation from Oracle.

## Storage and Object Administration

Typically, MFG/PRO on Oracle is installed on larger systems with multiple disks. This section explains Oracle Enterprise Manager (OEMGR) components that are available for database administration with respect to use of tablespaces, data files, tables, data, and so on.

### Tablespaces and Data Files

The physical Oracle database is logically split into tablespaces with names such as SYSTEM, RBS, MFGHELP, GUI, CONTROL. Each of these consists of one or more operating system data files or raw devices. By default, MFG/PRO uses 40+ tablespaces.

#### Creation of Tablespaces

The Oracle database administrator must create the necessary tablespaces before MFG/PRO can use the database. You can do this using the SQL scripts generated during the MFG/PRO installation process in svrmgrl.

Designating tables and indexes to tablespaces is done by MFG/PRO scripts which create tables and indexes. These scripts make use of tablespace clause of `create` ... command.

### Using OEMGR to Administer Storage

The OEMGR tool can be used to administer and display various storage information. The following components are available:

- Oracle Storage Manager
- Oracle Schema Manager
- Oracle Data Manager

## Oracle Storage Manager

With Oracle Storage Manager you can perform administrative tasks associated with managing database storage. These tasks include managing tablespaces and rollback segments, and adding and renaming data files. Storage Manager displays a tree listing and multi-column lists allowing you to manipulate and view objects.

## Oracle Schema Manager

With Oracle Schema Manager you can create, edit, and examine schema objects. The schema application manages the following groups:

**Table 3.1** Schema Objects Managed with the Oracle Schema Manager

| | |
|---|---|
| Clusters | Refresh Groups |
| Database Links | Sequences |
| Functions | Snapshot Logs |
| Indexes | Snapshots |
| Package Bodies | Synonyms |
| Packages | Tables |
| Procedures | Triggers |
| Queues | Views |

## Oracle Data Manager

Oracle Data Manager lets you transfer data to and from Oracle databases. Data Manager displays a tree listing the User table and partition property sheets to let you manipulate and view objects. Wizards are accessed with the Data menu.

Oracle Data Manager provides wizards for an easy and efficient interface for exporting, importing, and loading data to and from an Oracle database.

### Export Wizard

The Export wizard guides you through the process of exporting data from an Oracle database. You can import the exported data into an Oracle database later on. The Export option is accessed from the Data menu.

### Import Wizard

The Import wizard guides you through the process of importing data to an Oracle database. This data has been previously exported from an Oracle database. The Import option is accessed from the Data menu.

### Load Wizard

The Load wizard guides you through the process of loading data into an Oracle database. This data is in text or in another non-Oracle data format. The Load option is accessed from the Data menu.

# Tablespaces, Rollbacks, and Extents

This section provides a very brief overview of these topics. See your Oracle documentation for details.

## Administrative Tablespaces

All Oracle databases have at least a SYSTEM tablespace. Other tablespaces provide data storage and administration support. The administration tablespaces include RBS for rollback segments, TOOLS for DBA toolsets, TEMP for temporary swapping or backup, and USERS to contain user files.

The additional tablespaces are useful to segregate users from the data dictionary and to spread data across multiple disks. You can also take individual tablespaces on and offline for administration purposes. Individual tablespaces can also be backed up in ARCHIVELOG mode.

## Rollback Segments

Rollback segments are stored in tablespaces and allocated in extents. Each rollback segment has at least two extents. Multiuser databases should have multiple rollback segments to reduce rollback segment contention and improve database performance.

All rollback segments are the same size except one large and/or one small segment for transactions assigned specifically to this segment. You must

create a rollback segment in the SYSTEM tablespace before creating additional rollback segments in other tablespaces.

### Controlling Extent Allocation

Storage parameters define how Oracle allocates free database space for temporary, rollback, data, and index segments. Settings should maximize use of contiguous free space, and reduce segment or tablespace fragmentation. The parameters are:

**Table 3.2** Oracle Storage Parameters

| Parameter | Description |
|---|---|
| initial | Size, in bytes, of the first extent allocated to a segment. |
| next | Size, in bytes, of the next incremental extent in a segment. |
| maxextents | Total number of extents allocated to a segment. |
| minextents | Number of extents allocated when a segment is created. |
| pctincrease | Percentage by which each incremental extent (after the second extent) grows over the previous extent allocated for that segment. |
| optimal | Optimal size, in bytes, for a rollback segment. |
| freelists | Number of lists of free blocks available for row inserts. |

## Start-Up and Shutdown

The Oracle instance and database are started and shut down using staged or complete steps.

### Start-up Options

See “Basic Recovery Steps” on page 55.

The variety of start-up options provides you with the means to control access and recovery of failed sessions.

**Table 3.3** Oracle Start-Up Options

| Start-up Option | Description |
|---|---|
| Startup Nomount | This starts the instance and is used prior to database creation. |
| Startup Mount | Starts the instance and database for the purpose of maintaining the control files. |
| Startup Open | This initiates a full start-up of the instance and database with no access restrictions |

| Start-up Option | Description |
|---|---|
| Startup Restrict | This initiates a full start-up of the instance and database with access for administration tasks only. |
| Startup Recover | This initiates a full start-up after a media failure. The automated Oracle recovery routines are run during this process. |
| Startup Force | This aborts the currently running instance and starts a new instance. |

## Shutdown Options

The shutdown options provide different ways of performing a complete shutdown.

**Table 3.4**
Oracle Shutdown Options

| Shutdown Option | Description |
|---|---|
| Normal Shutdown | This option allows current connections to remain live, but permits no new connections. When the last connection is closed, the database and instance are shutdown. |
| Immediate Shutdown | This option disconnects all users and shuts down the instance and database. Any open SQL statements are terminated and all uncommitted transactions are rolled back. |
| Abort | Exactly like the immediate shutdown but does not roll back uncommitted transactions. A full instance recovery is required for the next start-up. This option is used during media failures. |

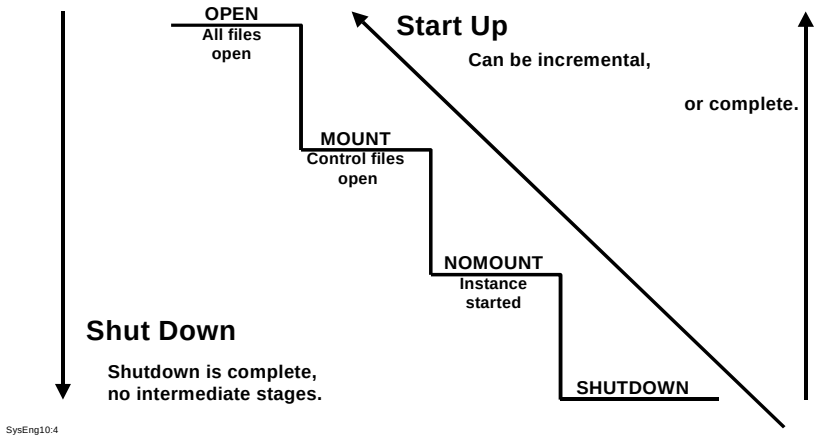



**Fig. 3.1**
Oracle Start-Up and Shutdown Options

# Database Backup and Recovery

This section covers the structures and software mechanisms used by Oracle to provide

- Database recovery required by different types of failures
- Flexible recovery operations to suit any situation
- Availability of data during backup and recovery operations so that system users can continue to work

## Why Is Recovery Important?

In every database system, the possibility of a system or hardware failure always exists. Should a failure occur and affect the database, the database must be recovered. The goals after a failure are to ensure that the effects of all committed transactions are reflected in the recovered database and to return to normal operation as quickly as possible while insulating users from problems caused by the failure.

## Types of Failures

Several circumstances can halt the operation of an Oracle database. The most common types of failure are described below.

### User Error

User errors may require a database to be recovered to a point in time before the error occurred. To allow recovery from user errors and accommodate other unique recovery requirements, Oracle provides for exact point-in-time recovery. For example, if a user accidentally drops a table, the database can be recovered to the instant in time before the table was dropped.

### Statement and Process Failure

Statement failure occurs when there is a logical failure in the handling of a statement in an Oracle program (for example, the statement is not a valid SQL construction). When statement failure occurs, the effects (if any) of the statement are automatically undone by Oracle and control is returned to the user.

A process failure is a failure in a user process accessing Oracle, such as an abnormal disconnection or process termination. The failed user process cannot continue work, although Oracle and other user processes can. The Oracle background process PMON automatically detects the failed user process or is informed of it by SQL*Net. PMON resolves the problem by rolling back the uncommitted transaction of the user process and releasing any resources that the process was using.

Common problems such as erroneous SQL statement constructions and aborted user processes should never halt the database system as a whole. Further, Oracle automatically performs necessary recovery from uncommitted transaction changes and locked resources with minimal impact on the system or other users.

### Instance Failure

Instance failure occurs when a problem arises that prevents an instance (system global area and background processes) from continuing work. Instance failure may result from a hardware problem such as a power outage or a software problem such as an operating system crash. When an instance failure occurs, the data in the buffers of the system global area is not written to the data files.

Instance failure requires instance recovery. Instance recovery is automatically performed by Oracle when the instance is restarted. The redo log is used to recover the committed data in the SGA database buffers that was lost due to the instance failure.

### Media (Disk) Failure

An error can arise when trying to write or read a file that is required to operate the database. This is called disk failure because there is a physical problem reading or writing physical files on disk. A common example is a disk head crash, which causes the loss of all files on a disk drive. Different files may be affected by this type of disk failure, including the data files, the redo-log files, and the control files. Because the database instance cannot continue to function properly, the data in the database buffers of the system global area cannot be permanently written to the data files.

A disk failure requires media recovery. Media recovery restores database data files so that the information in them corresponds to the most recent time point before the disk failure, including the committed data in memory that was lost because of the failure. To complete a recovery from a disk failure, the following is required: backups of the database's data files, as well as all online and necessary archived redo-log files.

Oracle provides for complete and quick recovery from all possible types of hardware failures including disk crashes. Options are provided so that a database can be completely recovered or partially recovered to a specific point in time.

If some data files are damaged in a disk failure, but most of the database is intact and operational, the database can remain open while the required tablespaces are individually recovered. Therefore, undamaged portions of a database are available for normal use while damaged portions are being recovered.

## Structures Used for Recovery

Oracle uses several structures to provide complete recovery from an instance or disk failure:  the redo log, rollback segments, a control file, and necessary database backups.

### Redo Logs

The redo log is a set of files that protect altered database data in memory that has not been written to the data files. The redo log can be comprised of two parts: online redo logs and archived redo logs.

### Online Redo Logs

The online redo log is a set of two or more online redo-log files that record all committed changes made to the database. Whenever a transaction is committed, the corresponding redo entries that are temporarily stored in redo log buffers of the system global area are written to an online redo-log file by the background process LGWR.

The online redo-log files are used in a cyclical fashion. For example, if two files constitute the online redo log, the first file is filled, the second file is filled, the first file is reused and filled, the second file is reused and filled, and so on. Each time a file is filled, it is assigned a log sequence number to identify the set of redo entries.

To avoid losing the database due to a single point of failure, Oracle can maintain multiple sets of online redo-log files. A multiplexed online redo log consists of copies of online redo-log files physically located on separate disks; changes made to one member of the group are made to all members.

If a disk that contains an online redo-log file fails, other copies are still intact and available to Oracle. System operation is not interrupted and the lost online redo-log files can be easily recovered using an intact copy.

### Archived Redo Logs

Optionally, filled online redo files can be archived before being reused, creating an archived redo log. Archived (off-line) redo-log files constitute the archived redo log.

The presence or absence of an archived redo log is determined by the mode that the redo log is using:

- ARCHIVELOG—The filled online redo-log files are archived before they are reused in the cycle.
- NOARCHIVELOG—The filled online redo-log files are not archived.

In ARCHIVELOG mode, the database can be completely recovered from both instance and disk failure. The database can also be backed up while it is open and available for use. However, additional administrative operations are required to maintain the archived redo log.

If the database's redo log is operated in NOARCHIVELOG mode, the database can be completely recovered from instance failure, but not from a disk failure. The database can be backed up only while it is completely closed. Because no archived redo log is created, no extra work is required by the database administrator.

### Control Files

The control files of a database keep, among other things, information about the file structure of the database and the current log sequence number being written by LGWR. During normal recovery procedures, the information in a control file is used to guide the automated progression of the recovery operation.

#### Multiplexed Control Files

This feature is similar to the multiplexed redo log feature. A number of identical control files may be maintained by Oracle, which updates all of them simultaneously.

### Rollback Segments

Rollback segments record rollback information used by several functions of Oracle. During database recovery, after all changes recorded in the redo log have been applied, Oracle uses rollback segment information to undo any uncommitted transactions.

Because rollback segments are stored in the database buffers, this important recovery information is automatically protected by the redo log.

## Database Backups

Because one or more files can be physically damaged as the result of a disk failure, media recovery requires the restoration of the damaged files from the most recent operating system backup of a database. There are several ways to back up the files of a database.

### Full Backups

A full backup is an operating system backup of all data files, online redo-log files, and the control file that constitutes an Oracle database. Full backups are performed when the database is closed and unavailable for use.

### Partial Backups

A partial backup is an operating system backup of part of a database. The backup of individual tablespace data files or the backup of a control file are examples of partial backups. Partial backups are useful only when the database's redo log is operated in ARCHIVELOG mode.

A variety of partial backups can be taken to accommodate any backup strategy. For example, you can back up data files and control files when the database is open or closed, or when a specific tablespace is online or offline. Because the redo log is operated in ARCHIVELOG mode, additional backups of the redo log are not necessary. The archived redo log is a backup of filled online redo-log files.

## Basic Recovery Steps

Due to the way in which DBWR writes database buffers to data files, at any given point in time a data file may contain some data blocks tentatively modified by uncommitted transactions and may not contain some blocks modified by committed transactions. Therefore, two potential situations can result after a failure:

Blocks containing committed modifications are not written to the data files, so the changes may only appear in the redo log. Therefore, the redo log contains committed data that must be applied to the data files.

Since the redo log may contain data that was not committed, uncommitted transaction changes applied by the redo log during recovery must be erased from the data files.

To solve this situation, two separate steps are always used by Oracle during recovery from an instance or media failure:  rolling forward and rolling back.

### Rolling Forward

The first step of recovery is to roll forward, that is, reapply all of the changes recorded in the redo log to the data files. Rolling forward proceeds through as many redo-log files as necessary to bring the data files forward to the required time.

If all needed redo information is online, Oracle performs this recovery step automatically when the database starts. After roll forward, the data files contain all committed changes as well as any uncommitted changes that were recorded in the redo log.

### Rolling Back

The roll forward is only half of recovery. After the roll forward, any changes that were not committed must be undone. After the redo-log files have been applied, then the rollback segments are used to identify and undo transactions that were never committed, yet were recorded in the redo log. This process is called rolling back. Oracle completes this step automatically.

## Backup and Recovery Sample Strategies

All production systems have a potential risk for failure. The hardware may fail, the different parts of the software may fail, and users may make mistakes. Designing the correct backup strategy is crucial, and it is very important that this be done as early as possible. Some examples of backup strategies are listed here.

### Simple Backup Strategy

A simple backup strategy that works well in many cases is to completely back up the database at regular intervals. The actual backup procedure contains these elements:

1. Log off all users and shut down the database.
2. Back up all database, redo log, and control files.
3. Start up the database.

4 At regular intervals (for example, once per week) make a backup using the Oracle export utility.

5 Make a fixed schedule for reuse of backup media, such as having one set per workday. Keep one set from each week for a longer period, and keep one set per month even longer.

This simple approach provides a backup in case of any failure and, if performed regularly, ensures that you lose at most one day of work.

### Providing Extra Security with an Oracle Export Backup

The Oracle export utility can be used to back up your entire database. It is a good idea to create a database export at regular intervals (for example, once per week or twice per month), since this gives you more possibilities for the backup of the various database files. Using an export file, you can create the database on a different system, even on a different operating system, this can be of great value in an emergency situation. The normal backup can only be restored in the actual system where it was created.

### Complex Backup Scenario

When databases become larger and more complex, run the database in ARCHIVELOG mode. This means that all redo-log files are archived during operation, and you can perform database file backups without shutting down the database. A setup that provides good security is:

- Put database files and multiple control files on non-mirrored devices.
- Put redo-log files on a mirrored device.
- Run the database in ARCHIVELOG mode.
- Make a backup schedule for all database files. Since the database is in ARCHIVELOG mode, the backups can be made without shutting down the database and without having users disconnect.

This lets you back up until the last committed transaction in case of any single device failure. You can do online recovery if the operating system permits switching of devices without shutting down the system, and only some users may be affected by a failure on one device.

The SYSTEM and ROLLBACK tablespaces are the most important part of the database, since users cannot work without access to them. If one of

these tablespaces is unavailable due to a device failure, the entire database is unavailable. Putting these two tablespaces on mirrored devices decreases the risk of having the database unavailable in case of a single disk failure.

# Useful Oracle Commands

## Checking Connectivity from a Server

```
SQL> $sqlplus system/manager@connect_string
```

## Creating a User ‘test’

```
SQL> $sqlplus system/manager@inst1
SQL> create user test
  identified by pwd
  default tablespace users
  temporary tablespace temp;
```

## Granting test Privileges

```
SQL> grant connect, resource to test;
```

## Identifying Users in a Database

```
SQL> select * from all_users;
```

## Creating Tablespace test_ts

```
SQL> create tablespace test_ts
datafile ‘d1/oradata/test_ts01.dbf’size 10M
default storage (initial 1M next 1Mpctincrease 0);
```

## Extending a Tablespace

### Adding a Data File

```
SQL> alter tablespace test_ts
add datafile ‘d1/oradata/test/test_ts02.dbf’ size 10M;
```

### Extending the Existing Data File

```
SQL> alter database datafile
‘d1/oradata/test/test_ts01.dbf’   resize 20M;
```

## Rename or Move a Data File

```
SQL> alter tablespace test_ts offline;
SQL> alter database rename file
‘/d1/oradata/test/test_ts01.dbf’ to
‘/d2/oradata/test/test_ts01.dbf’;
SQL> alter tablespace test_ts online;
```

## Adding a Rollback Segment

```
SQL> create rollback segment rb01
tablespace rbs
storage (initial 2M next 2M minextents 2);
SQL> alter rollback segment rb01 online;
```

***Note*** Edit the rollback_segs parameter in the init<sid>.ora file to ensure the rollback segment comes online automatically.

## Exporting the Database

### Full Database

```
$exp system/manager file=xx.dmp buffer=100000
full=y
log=export.log
```

### User Level

```
$exp system/manager userid=qad file=xx.dmp
buffer=100000
log=export.log
```

## Importing the Database

### Full Database

```
$imp system/manager file=xx.dmp buffer=100000 commit=Y
full=Y ignore=Y indexes=Y grants=Y
destroy=N
log=import.log
```

### User Level

```
$imp system/manager fromuser=user1 touser=user2
file=xx.dmp buffer=100000 commit=Y ignore=Y
rows=Y indexes=Y grants=Y destroy=N
log=import.log
```

## Data Dictionary Views

The following data dictionary views are those most useful for an application like MFG/PRO.

**Table 3.5** Data Dictionary Views

| Data Dictionary | Description |
|---|---|
| `dba_data_files` | List of data files. |
| `V$controlfile` | List of control files. |
| `dba_tablespaces` | List of tablespaces. |
| `dba_rollback_segs` | List of rollback segments. |
| `v$logfile` | List of redo-log files. |
| `v$database` | Database instance name. |
| `v$session` | List of active user sessions. |

## Q/ADMIN and MFG/UTIL for System Admin

There are various system maintenance programs in MFG/UTIL. Some are listed in Table 3.6. For details on using MFG/UTIL, refer to Chapter 4, “Administration Utilities,” on page 63.

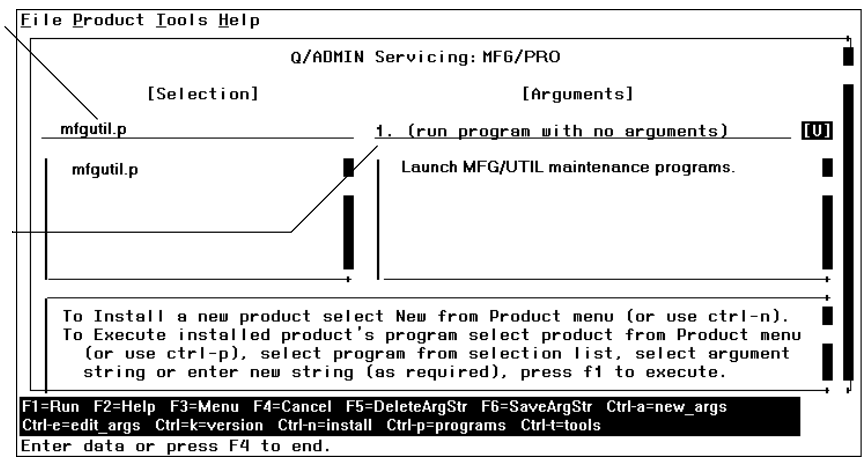
**Table 3.6** MFG/UTIL System Maintenance Programs

| Maintenance Program | Description |
|---|---|
| Start MFG/UTIL | From the MFG/PRO installation directory, type: `./mfgutil`. |
| Modify Start-Up Scripts | From the MFG/UTIL Configure menu, choose Any Database Set and complete the Database Set Configuration window. To save your changes to the start-up scripts, choose one of the generate options from the Scripts menu. |
| Compile Code | To compile source programs, select the MFG/UTIL Program menu and choose Compile Procedures. |
| Generate Oracle Database Scripts | Changes the settings in the SQL scripts, which you run to create the Oracle database. You must answer various prompts, including the directory locations for control files, dump directories, system data files, log files, and more. This information affects the scripts:<br>- crdb1<ORACLE_SID>.sql<br>- crdb1<ORACLE_SID>.sql<br>- init<ORACLE_SID>.ora<br>- config.<ORACLE_SID><br>where <ORACLE_SID> is your Oracle system ID. |

| Maintenance Program | Description |
|---|---|
| Start MFG/UTIL | From the MFG/PRO installation directory, type: `./mfgutil`. |
| Create New Schema Holder from Oraempty | Copies the oraempty schema holder to a new schema holder. |
| Change Oracle Connection Parameters | Alters an existing schema holder to connect to a different Oracle database. |

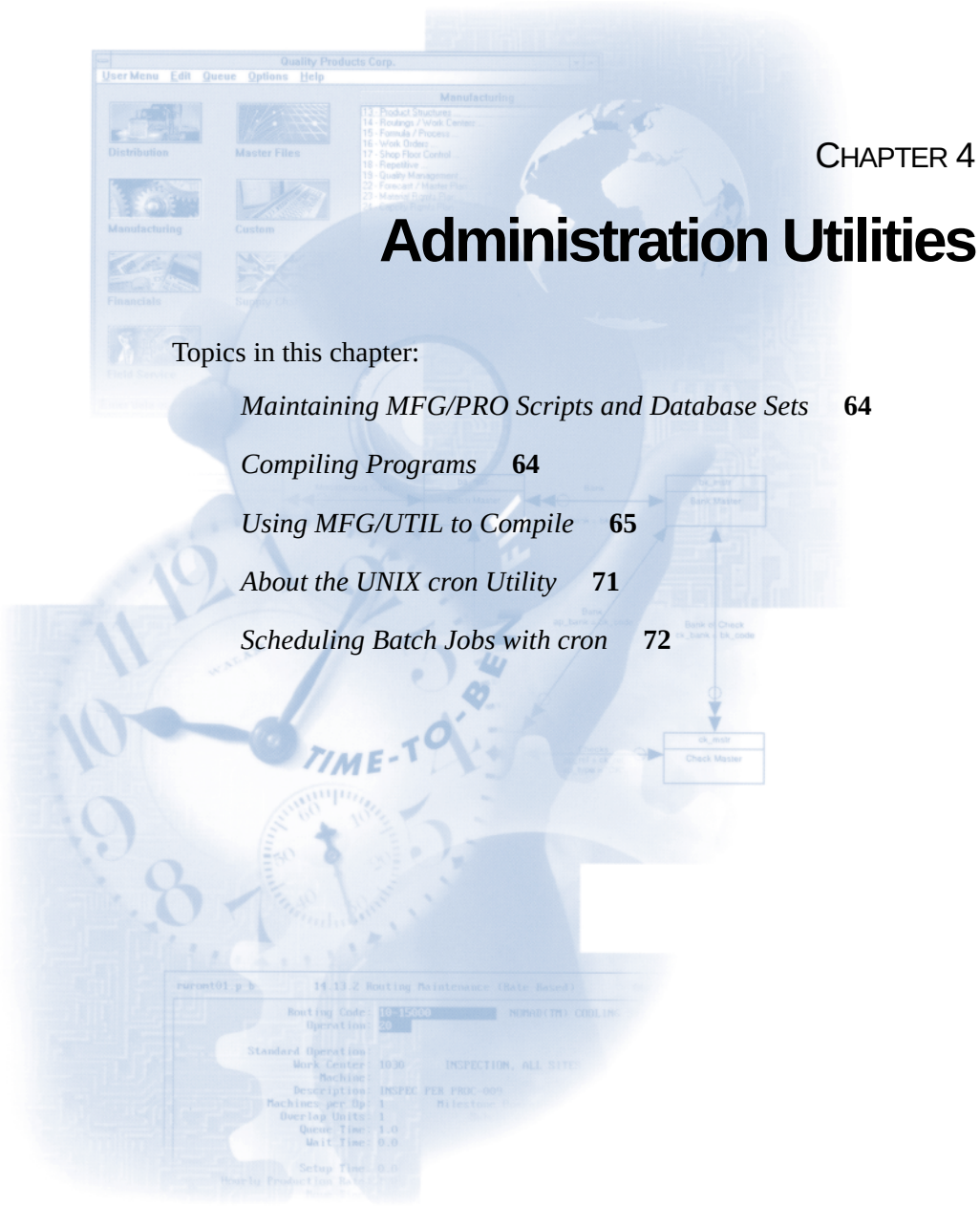
As an option, you can also start MFG/UTIL from the Q/ADMIN interface. Refer to the following illustration for guidelines. To start Q/ADMIN, change to the `bin` sub-directory of your Q/ADMIN installation directory and run the start script.

**Fig. 3.2**
Parts of the MFG/UTIL Screen




***Note*** If you encounter errors when starting MFG/UTIL from Q/ADMIN, try opening the script mfgutil and adding the following line before the command that executes MFG/UTIL.

```
cd InstallDir
```



CHAPTER 4

# Administration Utilities

Topics in this chapter:

## Maintaining MFG/PRO Scripts and Database Sets

To modify the MFG/PRO start-up and shut-down scripts and icons, you can use the MFG/UTIL utility. The scripts and icons are based on database sets. A database set defines the connection parameters for a group of databases.

1. From the MFG/UTIL Configure menu, choose Any Database Set and complete the Database Set Configuration window.
2. To save your changes to the start-up scripts, choose one of the generate options from the Scripts menu.
3. If you added a new database name, set it up in the hosts and services files on both the server and client PCs. For detailed instructions, refer to the MFG/PRO installation guide.
4. Remember to make your changes on both the server and, if applicable, Windows clients. There is one version of MFG/UTIL on the server and another on the Windows client.

## Compiling Programs

Sometimes you may need to compile a custom program, or recompile an existing program. Do this using MFG/UTIL. You must have the source file you want to compile.

Once the source file is compiled, an object file with the same name as the source file is created, but it has a .r extension. By default, MFG/UTIL saves the .r program in a two-letter sub-directory (first two letters of the program name), below the two-letter language sub-directory (such as us), below the main MFG/PRO install directory.

Keep in mind that you have two sets of code to maintain: the Windows character programs on the server and the Windows client programs on the file server. The Windows client source code is optional at MFG/PRO client installation.

# Using MFG/UTIL to Compile

MFG/UTIL provides a compile function that lets you compile against an active database. Use the instructions in “Compiling Code” on page 69 to compile.

***Important*** Before compiling MFG/PRO code in an Oracle environment, you may need to perform additional if you are compile under one of the following circumstances.

- You are performing a non-English language compile.
- You are compiling on a Windows character client.

## Non-English Language Compile

If you are compiling for a non-English language, you must first create a compile database set. Refer to the instructions in your installation guide, “Multiple Language Setup” chapter.

## Compiling Character Code

Prior to PROGRESS versions 8.3A03 for Windows and 8.3A02 for UNIX, compiled character code is not completely portable across platforms. A difference in program frame layouts exists between character r-code compiled on various UNIX machines. Differences also exist between character r-code compiled on Windows character client machines and UNIX machines.

***Important*** Because many MFG/PRO programs reference, or share, the same program frame layout, run-time errors occur when character code compiled on different machines is mixed.

QAD compiles all MFG/PRO character code on UNIX. The compiled character code (.r extension, referred to as r-code) is initially portable across platforms because all the program frames have the same layout.

A problem arises when users compile programs at their sites, for custom programs or patch implementation. The recompiled programs reference frame layouts specific to the machine where they were compiled. For example, programs compiled on Windows character clients reference

Windows frame layouts and can no longer reference the existing UNIX character code frames.

Additionally, not all code compiled in the field on UNIX machines is compatible with the delivered r-code. Any UNIX system that stores integers in byte-swapped order generates incompatible r-code. An example is UNIX486V4.

When incompatible code is run, the following PROGRESS error occurs. In this example, *ProgramName* refers to the recompiled program name, and *FrameName* refers to the shared frame the program uses.

```
ProgramName Shared frame layouts do not match for frame
FrameName. (683)
```

### Solution

To solve this problem, choose one of the following options.

- Perform all compiles on a UNIX machine that does not store integers in byte-swapped order; for example, Sun, HP, or IBM machines. All r-code generated on these machines uses the same frame layouts as the delivered MFG/PRO r-code.
- Recompile all character programs on a single machine and perform all future compiles on the same machine.

  For example, recompile all character code on a Windows character client and perform all future compiles on a Windows character client.

***Note*** This option does not require any additional setup or software.

- Recompile all character programs using the PROGRESS 8.3 `-portableframes` start-up parameter. Perform all future compiles using this parameter.

### Compiling with the `-portableframes` Parameter

The `-portableframes` parameter enables you to compile on either UNIX or Windows machines. Use the following instructions to implement this solution.

Beginning with PROGRESS Versions 8.3A03 for Windows and 8.3A02 for UNIX, the `-portableframes` start-up parameter is provided to

enable the creation of portable character code from either Windows character clients or UNIX machines.

***Important*** Once this parameter is set, all character code must be compiled using it.

Run-time errors occur if you mix character code compiled with the `-portableframes` parameter and character code compiled without it. If you choose to use the `-portableframes` start-up parameter, you must recompile all MFG/PRO character code with it. Future compiles can be performed on either UNIX or Windows character client machines, but must have the `-portableframes` parameter set.

To implement the `-portableframes` start-up parameter, use the instructions appropriate to the operating system you will compile with.

***Important*** QAD recommends that you recompile all programs during installation. If not, you must recompile all MFG/PRO programs when you compile your first patch or custom program.

### Windows

1. Right click the Start menu button.
2. Select Open All Users and the Start program group is displayed.
3. Double click on the Programs icon. The individual programs are displayed.
4. Double click the MFG/PRO program folder.
5. Edit the properties of the MFG/UTIL icon by performing the following:
   a. Right click the MFG_UTIL icon and select Properties. From the Properties window, select the Shortcut folder tab.
   b. Add the `-portableframes` parameter to the end of the Target field.
   c. Click the Apply button to save your changes

6 When using PROGRESS version 8.2 or 8.3 and MFG/UTIL to compile all—or a large number of—MFG/PRO programs, you may receive the following error.

```
Error: Insufficient disk space or write access denied
(291).
```

To avoid this error, add the following start-up parameters to the target field after the `-d` and `-yy` parameters and before the `-ininame` parameter.

```
-D 50 -TM 31 -TB 31 -B 100
```

7 Perform all MFG/PRO compiles using this MFG/UTIL icon.

### UNIX

***Note*** If your UNIX system does not store integers in byte-swapped order, no further action is required. Do not compile programs with the `-portableframes` parameter unless you plan to recompile all MFG/PRO programs with this parameter.

1 Locate the MFG/UTIL start-up script on the machine you will use for compiling.

2 Add the `-portableframes` start-up option to the end of the command line.

3 Once the `-portableframes` start-up option is set, all code must be compiled using it. If you run mixed portable frame and non-portable frame compiled code, run-time errors occur.

## Compiling Code

1 Start an MFG/UTIL session on either the server or a Windows client PC. This session must run against a copy of PROGRESS ProVISION or 4GL.

2 From the MFG/UTIL main menu, select Program, then Compile Procedures.

3 In the Source Directory field, enter the source directory where the .p and other source files are located. Using the two radio buttons, you can compile all of the files in the source directory or point to a work file that contains a list of the files to be compiled. If you select the Files listed in the work file option, another input field displays. Enter the full path name of the work file in this field.

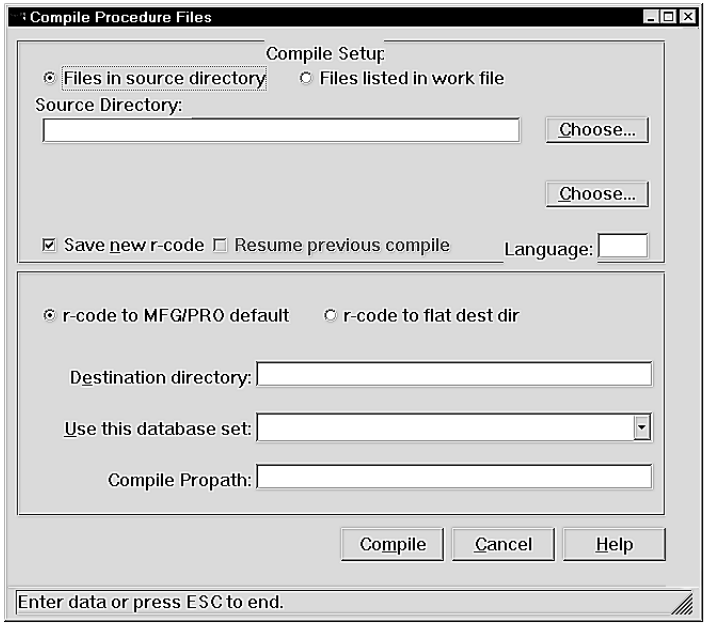



**Fig. 4.1** Compile Procedures Screen in MFG/UTIL

4 Indicate if you want the new .r files saved. If you are performing test compiles, leave this box unchecked until you are ready to do your final compile. In the Destination directory field, enter the full path name of the destination directory.

5 If you choose to save the compiled files, do the following substeps.

a Enter or accept the default for the two-letter language code in the Language field. It is appended to the end of the directory you specify in the Destination directory field.

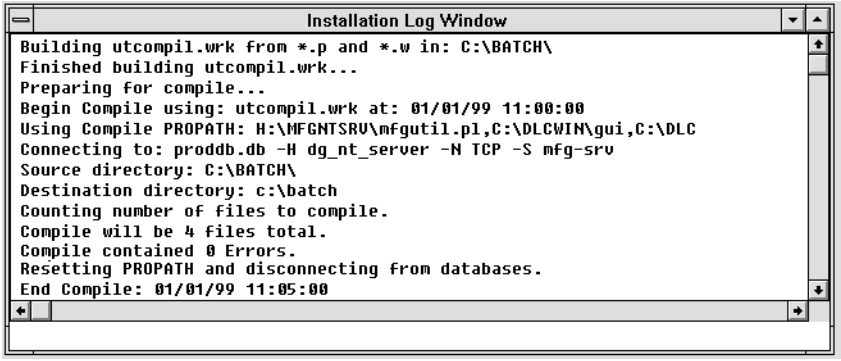
b Choose the MFG/PRO default option or the flat destination option. With the default option, your compiled .r code is placed in a series of two-letter sub-directories based on the first two letters of the program name. Also, if you compile triggers, they appear under a .\triggers sub-directory. With the flat destination option, all .r code is placed directly under the destination directory, with no language code directory appended.

6 The compile process must connect to an active database. In the Use this database set field, select a database set to run the compile against. If you have configured a compile database set, you should select that set; otherwise, select any database set that already has database servers started for it.

7 Choose Compile.

8 During the compile, an Installation Log Window displays the current status.

**Fig. 4.2**
Installation Log Window




9 When the compile ends, choose Print or Close.

***Note*** MFG/UTIL saves the settings you entered in the Compile Procedures Files window so that you can reuse or modify them for the next compile.

# About the UNIX cron Utility

If you need to schedule your MFG/PRO batch jobs to run at a specified time, set it up in the UNIX cron utility. Keep in mind the following facts about cron. The cron utility:

- Executes commands at scheduled times and dates.
- Typically starts when the UNIX system boots up.
- Checks crontab files for regularly scheduled commands.
- Should be run continuously and executed only once (otherwise, scheduled jobs run for each cron process that is running).

You set up cron jobs in a text file that follows a set format. It is called a crontab file. Each authorized user may have a crontab file located in the cron/crontab directory. Typically, the path to the crontab file is:

```
/usr/spool/cron/crontab/filename
```

You can make changes to this file with a text editor or an administration utility, if you have one. Make the changes while logged in as root or another authorized user. The format of the crontab file is shown in Figure 4.3.

**Fig. 4.3** crontab File Example

| Minute (0-59) | Hour (0-23) | Day of Mo. (1-31) | Month (1-12) | Day of Week 0=Sunday) | Script |
|---|---|---|---|---|---|
| **30** | **23** | ***** | ***** | ***** | **/usr/scripts/backup** |
| **00** | **03** | **31** | ***** | ***** | **/usr/scripts/mo.end** |
| **15** | **22** | ***** | ***** | **1-5** | **/usr/scripts/daybatch** |
| **48** | **01** | ***** | ***** | **1,3,5** | **/usr/scripts/mwf** |

Some standard crontab commands are listed in Table 4.1 on page 72.

**Table 4.1**
Standard crontab Commands

| Command | Description |
|---|---|
| crontab < filename | Copy an instruction file to a crontab file |
| crontab -l | Display contents of your crontab file (if you are a valid user) |
| crontab -l > filename | Copy a crontab file to a file |
| crontab -r | Remove your crontab file (if you are a valid user) |

## Scheduling Batch Jobs with cron

The following steps outline how to schedule an MFG/PRO batch job through cron.

1. In MFG/PRO, set up batch IDs and output reports or processes to these batch IDs.
2. Create a PROGRESS program that runs MFG/PRO and specifies a file to use as input to that MFG/PRO session. This program should also specify a file to write output to.
3. Depending on the PROGRESS product you are using, you may need to compile this file first.
4. Create an input file containing keyboard-style commands that direct MFG/PRO to start the Batch Request Processor and run the desired batch ID. The input file can also start any MFG/PRO program.
5. Modify the crontab file to call the PROGRESS program at a specific day and time.

Figure 4.4 on page 73 depicts the components required to set up MFG/PRO batch jobs.

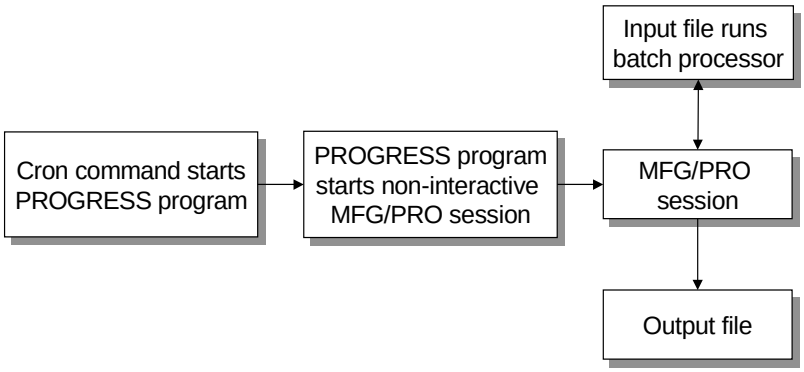



**Fig. 4.4**
Batch Job Components

## Creating the PROGRESS Program

Write a simple PROGRESS program that specifies an input file and runs MFG/PRO (at a minimum). You should also compile the program for best performance. Use the following example as a model.

Input from `InstallDir/input.in`.

Output to `InstallDir/output.out`.

Run `InstallDir/mf.p`.

Input `close`.

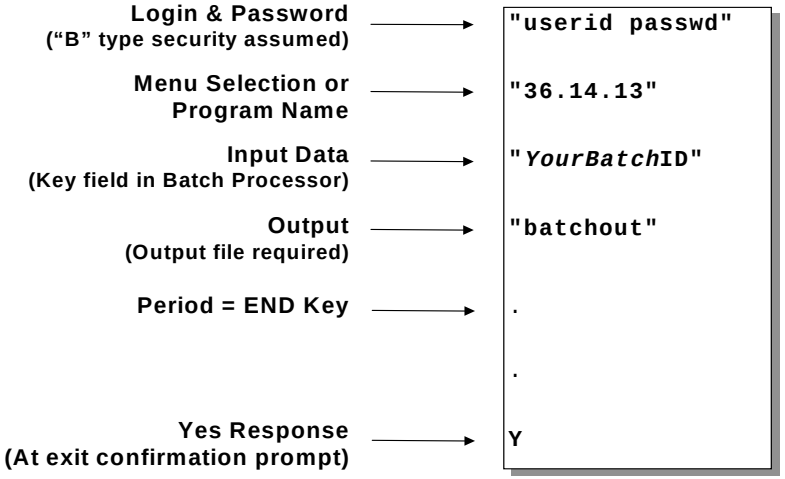
Output `close`.

## Creating the Input File

The input file contains the instructions for logging into MFG/PRO and starting Batch Request Processor (36.14.13) or any other MFG/PRO program. Create it in a text editor using Figure 4.5 on page 74 as a model.

The input file follows the same rules as CIM load files. For example, a carriage return represents the Go key (F1 in character and F2 in Windows) and a period on its own line represents an End key (F4 in character and Esc in Windows).

**Fig. 4.5**
Example Input File




## Setting Up cron to Run the Batch Program

Once you have the PROGRESS program created, you can reference it in the cron settings. The command to start a PROGRESS program depends on whether you are running in multiuser or single-user mode. Choose one of the following PROGRESS scripts to start a session.

mbpro – starts a multiuser session against a database (a server process must be running for the database)

bpro – starts a single-user session against a database

The cron setting must run the batch program against an entire database set, including the production, gui, mfghelp, and cfg databases.

For example, to start a batch program (batrun.p) against the Demonstration database (mfgdemo.db) in single-user mode (bpro), you can use the following command.

```
$DLC/bin/bpro /path/mfgdemo -db mfghelp -db gui -db cfg
-p /path/batrun.p
```

To use this via cron, edit the crontab file to contain this command plus the appropriate date and time instructions. For example, to run it every Saturday at 10:30 P.M., the crontab file would contain this entry:

```
30 22 * * 6 $DLC/bin/bpro /path/mfgdemo -db mfghelp -db
gui -db cfg -p /path/batrun.p
```

CHAPTER 5

# Performance Tuning

This chapter provides information about performance and performance tuning of MFG/PRO on Oracle. Both MFG/PRO and Oracle are complex software packages, and it is therefore crucial that expectations are set and the correct steps are taken to optimize system performance.

Achieving required performance requires decisions prior to installing the software and creating the database, and it requires tasks during the life cycle of the system.

## Relevant Tuning Areas

This chapter outlines the different tuning actions that can be performed in order to improve performance of an On-Line Transaction Processing (OLTP) application such as MFG/PRO. Each of the steps mentioned is a tuning action of its own, and the steps should generally be followed in the order provided. However, tuning on the MFG/PRO application will impact the Oracle RDBMS and vice versa.

See "Customizations" on page 97 for tuning of PROGRESS code.

Seen from an Oracle perspective, performance tuning includes, or actually starts with, application tuning. Application tuning is normally outside the possibilities given to an end-user, and the information about this in this chapter is meant for QAD partners or others who do application programming.

This chapter assumes some familiarity with Oracle database administration and basic Oracle performance tuning. Most suggestions apply to UNIX, NT, and open system platforms. However, some of the suggestions may have to be modified depending on the specific application and operating systems.

## Tuning Goals

For more on application tuning, see *Oracle7 Server Application Developer's Guide* and *Oracle7 Server Tuning Guide*.

There are several factors involved in achieving the best performance from your system such as the Oracle7 server configuration, the hardware, the operating system, and the application software. It is important to tune the application software to take advantage of the system resources. Due to the diversity of applications, covering all the aspects of application software tuning is beyond the scope of this chapter.

The general performance tuning procedure should follow these steps:

1. General tuning
2. Client/server tuning
3. Memory tuning
4. Disk I/O tuning
5. Resource contention tuning

Each of these is described below, with information about tuning on both the MFG/PRO side and the Oracle side for each step.

# General Tuning

General tuning includes various aspects of performance tuning. These are the tuning of SQL statements, coordinating access to manage locking behavior, and using Oracle's rule-base optimizer.

## Tuning SQL statements

SQL statements can be tuned using the `sql_trace` Oracle variable that can be set by user session or DBA. When the `sql_trace` resource is set to `true`, Oracle produces a trace file on the Oracle Server system. You can then format the trace file using `tkprof`. For example:

1. Set `SQL_TRACE` to `true`.
2. Run the application user session.

   An Oracle trace file is produced by default in $ORACLE_HOME/rdbms/trace on the Oracle Server system.
3. Process the trace file using:

   ```
   tkprof <name>.trc <name>.prf explain=
   <username>/<password>
   ```

   Specify the trace file name, an output file name, a user name, and password valid for MFG/PRO.

   ➧ See the *Oracle Server Tuning Guide*
4. The generated file, `<name>.prf`, contains all SQL statements processed. For each statement, an execution plan describing how Oracle accessed the data (e.g., through indexes) and a resource summary (e.g., physical and logical disk reads) is shown.

This tuning should be done by application developers only.

### Locking Behavior

See recommended parameter values in “Oracle Start-Up Parameters” on page 91.

When multiple users are requesting access to the same data—the same account in a general ledger—access must be coordinated. This is frequently done by a user locking a resource that they have modified or intend to modify so that no other user can lock or modify the same resource. The Oracle database engine automatically does most necessary locking. The init<SID>.ora parameter values of `dml_locks` and `enque_resources` should be set correctly.

### Optimizer

Two of the Oracle optimizer modes can be used with MFG/PRO–rule-based and cost-based.

- Under the rule-based mode:

  Oracle uses the SQL statements (primarily columns appearing in the WHERE clause) to determine whether to access the tables using a full table scan or an indexed lookup.

- Under the cost-based mode:

  Oracle uses statistics to determine how to access data in each table. These statistics must be generated against tables fully populated with production data to be of value.

QAD recommends using Oracle’s rule-based optimizer for MFG/PRO, because the WHERE clauses in MFG/PRO have been modified to trigger Oracle to use the most efficient lookup method for that table. Cost-based yields more varied and typically slower results. Tests have resulted in 20 items per minute under cost-based and up to 800 items per minute for the same lookup under rules-based.

## Client/Server Tuning

The PROGRESS Oracle DataServer communicates heavily with the Oracle server, and you should generally try to reduce the number of round trips between the two. If you are running the PROGRESS DataServer on a system separate from the Oracle server, such a round-trip is a network round-trip via SQL*Net.

# Memory Tuning

The Oracle server uses a large memory segment (typically 10 to several hundred megabytes) that must be tuned properly. This memory, the System Global Area (SGA), contains:

- Database buffers—copies of data blocks from the database files
- The shared pool, which consists of:
  - The shared SQL area, which contains information about all currently executing SQL statements. This is often referred to as the library cache.
  - The dictionary cache, which contains information about all currently used tables, columns, indexes, and so on.
  - The cursor area, which keeps information for each individual use of a SQL statement, such as rows of data accessed.
- The log buffer—copies of changed data blocks that are written to redo-log files.

It is essential that the size of the SGA is set appropriately for a given application, and it is most directly influenced by three Oracle start-up parameters.

See recommended parameter values in “Oracle Start-Up Parameters” on page 91.

- `db_block_buffers`

  Set the number of database blocks cached in the SGA.

- `shared_pool_size`

  Sets the amount of memory, in bytes, reserved for the various other caches.

- `log_buffer`

  Sets the amount of memory, in bytes, used for redo-log data blocks.

When tuning these, ensure the total size of the SGA does not exceed available memory in a system that is purely used as a database server. A reasonable total SGA size depends upon the total number of users supported. Use operating system utilities to ensure no paging or swapping of memory is taking place.

Compared to most applications, MFG/PRO requires more shared pool, due to the large number of tables and columns and the need to keep parsed SQL statements in the SGA.

## Optimizing the shared_pool_size

Optimizing the `shared_pool_size` consists of tuning the data dictionary hit ratio and library cache hit ratio. The data dictionary hit ratio should be above 90%, the library cache miss ratio should be less than 0.1.

## Optimizing the Buffer Cache (db_block_buffers)

### Buffer Hit Ratio

Ensure the buffer hit ratio is at least 70%. Otherwise, increase the `db_block_buffers` parameter.

## Tuning the log_buffers

Use the server manager to look at the redo log space requests. If space requests are greater than 0, increase the Oracle start-up parameter `log_buffers` by small increments until requests = 0.

### Reduce the Redo Log Buffer Latch Contention

Access to the redo-log buffer is controlled by two types of latches: the redo allocation latch and the redo copy latch. Look at the V$LATCH table for any contention (number of misses vs number of gets) for these latches. `log_simultaneous_copies` and `log_small_entry_max_size` Oracle start-up parameters control the redo latches contention. `log_simultaneous_copies` depends on the number of CPUs on the system. To correct the redo copy latch, increase the value of the `log_simultaneous_copies` parameter.

If the miss ratio or the immediate miss ratio is more than 1, decrease the `log_small_entry_max_size` parameter.

Excessive memory use in Oracle may cause some swapping or paging to occur. This can be monitored by `sar -w` or `vmstat -s` UNIX commands. If the system is swapping, you should decrease the memory usage by decreasing the `db_block_buffers`, `shared_pool_size`, `sort_area_size`, and `sort_area_retained_size` parameters.

### Tuning the Number of Rollback Segments

If any rollback segment ratio is more than 0.01, add a few more rollback segments.

Output

```
NAME                           WAITS         GETS     Ratio
------------------------------ ----------   ---------- --
--
SYSTEM                         0              269    .00000
R01                            0              304    .00000
R02                            0             2820    .00000
R03                            0              629    .00000
R04                            1              511    .00196
R05                            0              513    .00000
R06                            1              503    .00199
R07                            0              301    .00000
R08                            0              299    .00000
```

# Disk I/O Tuning

## General I/O Recommendations

Since all database systems store data on disks, it is essential that the disk layout is tuned properly. The following guidelines should be used:

- Distribution of load on Oracle database files should be as even as possible. One frequently used solution is to use disk-striping or similar hardware or operating system methods to achieve this. This is an administratively simple method, but it is very difficult to investigate problems, if the disk load is not evenly spread.
- In systems with high transaction rates, the Oracle redo-log files have high write activity, and should be placed on their own physical devices. They should not be put together with files with high random access such as Oracle database files or operating system paging space.

- Some operating systems offer RAID systems for increased security in case of hardware failures. RAID systems generally have slightly lower disk performance than individual disk systems, but can be used for Oracle database files. Oracle redo-log files should never be put on RAID systems.
- You can optionally use raw devices instead of file system files for your database and redo-log files. Raw devices can have performance benefits, but add complexity to the system administration.

## Identifying I/O Bottlenecks and Moving Tablespaces

You can use operating system utilities or Oracle statistics to display disk activity. Where uneven loads are identified, you can move some tablespaces from one disk to another:

1. Identify tablespaces with high disk activity.
2. Shut down the database.
3. Copy the operating system data files that correspond to the high disk-activity tablespaces to a new location.
4. Start the instance and mount the database.
5. Rename the data files in the Oracle system dictionary, using the `alter database rename file` command.
6. Open the database for users.

## Checkpoint Processing

Checkpoint processing is the activity that takes place in Oracle when a switch occurs between redo-log files. Oracle uses redo-log files to store information necessary for recovery. The redo-log files are used cyclically, and whenever a switch from one redo-log file to the next occurs, all database buffers are flushed from the buffer cache in the SGA to the database files. There are a number of Oracle start-up parameters that are associated with checkpoint processing, and two of these are of particular importance.

The `log_checkpoint_interval` parameter sets the number of 512 byte blocks in the redo-log file that are written before a checkpoint occurs. Setting this parameter to a larger value than redo-log file size makes the checkpoints occur only at log switches. This gives the best performance at the expense of potentially longer recovery times.

See “Oracle Start-Up Parameters” on page 91

During checkpoints, all database file headers must be updated. This is normally done by the redo-log writer background process, but setting the Oracle start-up parameter `checkpoint_process` to `true` creates a background process for this purpose. This causes better checkpoint performance when the database consists of many database files. Set it when more than 10 to 15 database files are used, which is true for MFG/PRO.

## Verifying Extent Counts

Each table or index in Oracle is stored in a segment, which has a number of extents. Extents are allocated when the table or index needs more space. Tables that grow rapidly are usually put into many extents, which can cause a negative impact on performance.

Such tables can be identified by means of data dictionary views (see the *Oracle Server Reference Manual*) and exported and imported to improve performance.

## Disk Subsystem Setup and Configuration

Databases require fast update and retrieval of information from the disk subsystem. The configuration of the disk subsystem can have a significant impact on database performance. If disks are configured poorly, the disk subsystem can quickly become a performance bottleneck.

## RAID Disks

A common way to optimize the disk subsystem is to use a RAID configuration to achieve the right mix of performance and availability. The following sections outline the different RAID configurations and the advantages and disadvantages of each type of configuration from the Oracle tuning perspective.

### RAID 0: Striping

RAID level 0 refers to striping data across multiple disks without any redundant information. Striping can be used to enhance performance in either a request-rate-intensive or transfer-rate-intensive environment. Unfortunately, striping reduces the level of data availability, since a disk failure causes the entire array to be inaccessible. RAID 0 is often used with mirroring for data availability. Check with your hardware vendor for RAID 0 mirroring options.

#### Advantages

- High performance
- No cost penalty–all storage is usable
- Works well for database applications

It is almost always better to use RAID 0 configuration for Oracle data files than single disks because RAID enables even I/O distribution among several disks.

#### Disadvantages

- Significantly reduced data availability unless mirroring is used

### RAID 1: Shadowing/Mirroring/Duplexing

RAID level 1 refers to maintaining duplicate sets of all data on separate disks. Of the RAID levels, level 1 provides the highest data availability since two complete copies of all information are maintained. In addition, read performance may be enhanced if the array controller allows simultaneous reads from both members of a mirrored pair. During writes, there is a minor performance penalty when compared to writing to a single disk. Higher availability is achieved if both disks in a mirror pair are duplexed (on separate I/O buses).

#### Advantages

- Excellent data availability
- Higher read performance than a single disk

Disadvantages

- Expensive–requires twice the desired disk space

### RAID 3: Striping and Parity

In RAID level 3, data is striped across a set of disks. In addition, parity is generated and stored on a dedicated disk. With RAID 3, data chunks are much smaller than the average I/O size and the disk spindles are synchronized to enhance throughput in transfer-rate-intensive environments. RAID 3 is well suited for CAD/CAM or imaging type applications. Since parity is used, a RAID 3 stripe set can withstand a single disk failure without losing data or access to data. However, RAID 3 is not recommended for large database applications such as MFG/PRO.

Advantages

- Good data availability
- High performance for transfer-rate-intensive applications
- Cost effective–only one extra disk is required for parity

Disadvantages

- Can satisfy only one I/O request at a time
- Poor small, random I/O performance
- Complicated
- Significant performance problems for databases

### RAID 5: Striping and Parity

In RAID level 5, both parity and data are striped across a set of disks. Data chunks are much larger than the average I/O size. Disks can satisfy requests independently, which provides high read performance in a request-rate-intensive environment. Since parity information is used, a RAID 5 stripe can withstand a single disk failure without losing data or access to data.

Unfortunately, the write performance of RAID 5 is poor. Each write requires four independent disk accesses. First, old data and parity are read off of separate disks. Next, the new parity is calculated. Finally, the

new data and parity are written to separate disks. Many array vendors use write caching to compensate for the poor write performance of RAID 5.

#### Advantages

- Average data availability
- Cost effective–only one extra disk is required

#### Disadvantages

- Poor write performance
- No performance gain in transfer-rate-intensive applications
- Complexity
- Requires special hardware
- Unsuitable for write-intensive databases

### RAID Stripe Size

See “Oracle Start-Up Parameters” on page 91

With RAID 0 and RAID 5 configurations, it is important to configure disks with an appropriate stripe size for optimum performance with Oracle. Oracle reads and writes from the disks in units that range from a single to multiple Oracle blocks (Oracle start-up parameter `db_block_size` and `db_file_multiblock_read_count`). If the stripe size is too large, I/O is not evenly distributed across the available disks; if the stripe size is too small, multiple physical I/O requests for every logical I/O occur, which adds overhead. For best performance, make the stripe size equal to multiples of the Oracle block size.

If software RAID 0 is used, the stripe size should never be smaller than the Oracle block size. Also, with software RAID 0, it is critical to ensure that Oracle block boundaries are aligned with stripe boundaries. Otherwise, Oracle blocks can span stripes, which requires multiple I/Os at the OS level to access a single block. This can degrade performance considerably, because most of the I/O overhead is due to the OS system call and disk arm setup instead of the actual data transfer time. With hardware RAID 0, the problem is minimized because the multiple I/O request is generated at the disk level, not the OS level.

Suggested stripe size for OLTP (On-Line Transaction Processing) applications such as MFG/PRO is 16k with 8k database block size (`db_block_size`). If the work load consists of batch jobs that require full table scans, use larger stripe sizes such as 32k or 64k.

The stripe size should not be smaller than the maximum size of an Oracle I/O request. The maximum size of an Oracle I/O request depends on the Oracle database block size (`db_block_size`) and the Oracle multi-block read count parameter (`db_file_multiblock_read_count`). For example, if the Oracle block size is 8k and the multi-block read count initialization parameter is set to 4k, the maximum size of an Oracle I/O request is 32k. The maximum I/O request size is limited to 64k in most operating systems.

### Non-RAID Disks

If you are not using RAID 0 or RAID 5 configurations, you must be careful in distributing the various Oracle files onto different disks and controllers to even out the I/O distribution and minimize contention at the disk and controller level.

## Oracle Files Distribution

When distributing Oracle files across the available disks, make sure that the I/O intensive files are distributed across the disks to minimize contention for a single disk. Here is a breakdown of the different Oracle files in the order of importance and how they should be distributed across the available disks.

### Online Redo-Log Files

The online redo-log files are usually the most I/O-intensive files in OLTP environments. Whenever there is a commit, a log entry is written to the log file, which bypasses the OS buffer cache. Also, unlike other types of I/O, the I/O to the log file must complete before Oracle can proceed. Therefore, log files should be placed on dedicated disks with their own controller to minimize contention. Since log reading and writing is sequential, it is not necessary to use RAID 0 disks for log files.

### Rollback Tablespace Files

The rollback data files are usually the second most I/O-intensive files in OLTP environments. Rollback data files conflict with online redo-log files. Every time there is an entry in the rollback segment, a log entry is written to the redo-log files. Therefore, the rollback data files should always be located on separate disks and controllers from redo-log files. Since rollback tablespaces are like regular tablespaces, they should be stored on RAID 0 disks for even I/O distribution.

### OLTP Data and Index Tablespace Files

Data and index tablespace files are usually not as I/O intensive if the SGA is large enough to accommodate the working set of the application. Data and index tablespace files should be on separate disks and controllers from each other, because they are usually accessed together. Both the data and index tablespace files should be stored on RAID 0 disks for even I/O distribution.

### Temporary Tablespace Files

The temporary tablespace is used for sort operations and complex join operations that do not fit in memory. With larger OLTP databases such as MFG/PRO, the temporary tablespace could be heavily used. Temporary tablespace files should be located on disks that are separate from the disks containing the index and data tablespaces.

## Disk I/O Tuning Example

As an example, if there are three controllers with two channels each and several types of disks, the following would be a good distribution of Oracle files for certain usage profiles:

Controller 1:

Channel 0 - Redo log file 1, Redo log file 3

Channel 1 - Redo log file 2, Redo log file 4

Controller 2:

Channel 0 - Rollback tablespace (RAID 0)

Channel 1 - Index tablespace (RAID 0)

Controller 3:

Channel 0 - Temporary tablespace (RAID 0)

Channel 1 - Data tablespace (RAID 0)

## Raw Devices vs File Systems

This topic has generated some confusion within the Oracle community. The general assumption that raw devices always give you a 10-15% performance boost is not always true. Disks and OS file system technologies are changing rapidly and there are cases where OS-level buffering could provide better performance than raw devices. Another factor to consider when deciding between raw devices and file systems is the administrative overhead of raw devices.

The difficulty in deciding between raw devices and file systems is that it is not possible to predict in advance whether your database will benefit from raw partitions. It is also not easy to convert database files that are stored on raw devices to a file system and vice versa. Therefore, you should tune other parts of your subsystem first and only move to raw devices when I/O is still a performance bottleneck.

## Database Block Size

During the standard Oracle installation procedure, it is possible to create a starter database. However, the database that is created during Oracle installation uses 2k database blocks on most platforms, which is not optimal for anything other than small databases. Unfortunately, once a database is created, you cannot change the database block size unless you re-create the database. To change a database block size:

1. Export your existing database.
2. Create a new database with a 4k or 8k `db_block_size`.
3. Import data back into database.

See the *Oracle7 Server Administrator's Guide*.

## Configuring Online Redo Logs

If online redo logs are not configured properly, two things can degrade performance. If the log files are too small, checkpoints occur too frequently, resulting in increased I/O. If the log files are too small and there are not enough log files, the log writer could end up waiting for checkpoints to complete, which makes the entire database wait until the checkpoint is complete.

A good starting point is three 5M online redo-log files, which can later be reconfigured if necessary. You can monitor the alert_<SID>.log to see if the log writer is forced to wait for checkpoints (indicating that more or larger redo-log files are needed). Set `log_checkpoint_interval` to a very large number to make sure checkpoints only happen during log switches.

## Configuring Rollback Segments

Rollback segments should be configured based on the number of users and the average amount of redo generated by a transaction. There should be enough rollback segments so that there are no waits for rollback segments.

See the *Oracle7 Server Administrator's Guide*.

The rollback segment extents should be sized so that each extent can contain the average amount of rollback generated by a transaction but small enough to ensure that rollback segments stay cached in memory. The optimal value should also be set for rollback segments so that rollback segments shrink infrequently and contain a minimum of five extents each.

## Configuring Temporary Tablespace

See the *Oracle Server Concepts Manual*.

The temporary tablespace is primarily used for sorts that do not fit in memory and should therefore be configured to optimize the performance of disk sorts. In most OLTP systems, the use of the temporary tablespace can be minimized by setting the `sort_area_size` high enough. This increases performance at the cost of higher memory utilization.

In Oracle 7.3, it is possible to optimize sort performance by enabling the `temporary` option for the temporary tablespace. This removes the overhead associated with allocating and de-allocating segments in the

temporary tablespace. If the database is an earlier version than Oracle 7.3, the `initial` and `next` parameters of the temporary tablespace should be set to correspond to multiples of the `sort_area_size` parameter within the Oracle start-up file.

### Tuning the Database Writer(s)

The database writer process (DBWR) writes all data changes to disk. There are three ways to modify DBWR performance:

- List I/O provides non-blocking writes.
- Asynchronous I/O enables DBWR to issue new writes without waiting for current writes to finish.
- Multiple DBWR processes are often useful on non-raw file systems (UFS).

To determine if DBWR is a bottleneck, run

```
select * from v$system_event
where event like 'buffer busy waits';
```

average_wait from the output should be close to zero (0). Experiment with the I/O tuning features to bring this close to zero. Check the *Oracle Installation and Configuration Guide* for availability of these features and potential conflicts on your hardware platform.

## Resource Contention Tuning

The most frequent resource contention is rollback segment headers and data blocks for tables with high insert rates. In systems with many CPUs, you may see contention for latches.

Resource contention tuning is discussed in detail in the *Oracle Server Tuning Guide*.

## Oracle Start-Up Parameters

The Oracle database engine is controlled via a large number of start-up parameters, often referred to as init<SID>.ora parameters, since they are stored in a file called init<SID>.ora. This section lists parameters that are relevant when Oracle is used as a database for MFG/PRO.

For full reference, consult the *Oracle Server Reference*.

## Notes on Parameter Usage and References

For each parameter, at least the following information is listed:

- Explanation.
- Typical or recommended value. A range of values is shown, if different values should be used for different numbers of users.

Many references are made to the *Oracle Server Tuning Guide*, which contains a wealth of information on performance-tuning Oracle and applications using Oracle.

## Parameters Controlling Disk I/O Behavior

**Table 5.1** init<SID>.ora I/O Parameters

| Parameter | Value |
|---|---|
| **Explanation** | |
| **db_block_size** | 4096, 8192 |
| Sets the logical database block size, which should be a multiple of the operating system block size. Must be set before database creation. The default value is often 2048, which is too small for many operating systems. | |
| **db_block_buffers** | 500-10,000 or more |
| Sets the number of database blocks that are cached in the Oracle System Global Area. The parameter should be set so that the cache hit ratio is between 95 and 98%. High performance impact. | |
| **db_file_multiblock_ read_count** | 8-16-32 |
| Sets the number of database blocks that are read sequentially at one time for full table scans.<br>With the rule-based optimizer, this parameter can be set to a large value to read as much table data in one read during a full table scan. With the cost-based optimizer, large values may influence selection of full table scan access over index based access. Potentially high performance impact. | |
| **db_writers** | 1 (see Explanation) |
| Sets the number of database writer processes.<br>If the operating system supports asynchronous I/O, the value should be 1. If set to a larger value, Oracle starts that number of database writer processes, which perform asynchronous writes. This parameter is not available on all operating systems. In Oracle Version 8, this parameter is replaced by a parameter called `dbwr_io_slaves`, available on all platforms. | |

| Parameter | Value |
|---|---|
| **Explanation** | |
| **log_checkpoint_interval** | Larger than largest redo-log file |

Sets the interval between checkpoints measured in units of redo-log file blocks, which normally is 512 bytes. Checkpoints always occur when a switch from one redo-log file to the next takes place. If this parameter is set to a low value, checkpoints occur more frequently, causing performance overhead, but ensuring a faster recovery time in case of system failures.

| | |
|---|---|
| **checkpoint_process** | true |

If set to `true`, a process is started to speed up checkpoint processing. Should be set to `true` if the database contains more than approximately 10 database files.

| | |
|---|---|
| **asynchronous I/O:**<br>**use_async_io** (HP)<br>**async_read, async_write** (SUN)<br>**disk_asynch_io** (Oracle8) | true (if supported by OS) |

If set, this parameter tells Oracle to use OS asynchronous I/O. The different parameters are available on different hardware platforms. Check your hardware-specific Oracle configuration manual. Should be set to `true` if the operating system supports asynchronous I/O. The `db_writers` should be 1. See parameter `db_writers`.

| | |
|---|---|
| **rollback_segments** | Comma-delimited list of rollback segment names. |

Lists the rollback segments brought online at database start-up. High performance impact.

## Parameters Controlling Memory and CPU Usage

**Table 5.2** init<SID>.ora CPU and Memory Parameters

| Parameter | Value |
|---|---|
| **Explanation** | |
| **shared_pool_size** | 10,000,000-100,000,000 or more |

Sets the size in bytes of the shared pool that Oracle uses to cache active SQL statements, dictionary information, etc. High performance impact.

| | |
|---|---|
| **cursor_space_for_time** | True, if memory allows it. |

Saves information about running SQL statements to improve performance. When set to `true`, the `shared_pool_size` should be sufficiently high or increased.

| Parameter | Value |
|---|---|
| **Explanation** | |
| **sort_area_size** | 262144 |
| Amount of sort memory that can be claimed by each user session to perform sorts. During operations like loading the data or creating indexes in single-user mode, this value can be set to a high value; e.g., 5M to get better performance. But make sure to reset it to the recommended value for normal production use. | |
| **sort_area_retained_size** | 262144 (same as sort_area_size) |
| Amount of sort memory retained for subsequent sorts. During operations like loading the data or creating indexes in single-user mode, this value can be set to a high value; e.g., 5M to get better performance. But make sure to reset it to the recommended value for normal production use. | |
| **sequence_cache_entries** | 525 |
| Number of sequences that can be cached into SGA. Default value of 10 to 30 is low, since MFG/PRO has over 510 sequence objects. | |
| **nls_sort** | Binary |
| Sort mechanism used by the database. The default is binary with the U.S. English. But the value of `nls_sort` is set to that particular language sort for some `nls_languages`, which degrades the performance. Set this value to `binary` if using `nls_language` parameter in init<SID>.ora. | |
| **Multi-CPU machines:** | |
| **cpu_count** | = number of CPUs |
| **db_block_lru_latches** | = larger of 1 or ¼ * cpu_count |
| **log_simultaneous_copies** | = 2 * cpu_count |
| **spin_count** | = 8,000 to 32,000 |
| Parameters used by Oracle in multi-CPU machines. These settings make better use of multiple CPUs with MFG/PRO. | |

## Configuration Parameters

The parameters in this section have no direct impact on performance, but are necessary to configure the Oracle database. Some of the parameters listed depend on the number of users connected and must therefore be modified if additional users are added. Because setting higher values for some of these parameters causes increased memory use by Oracle, there may be indirect performance impacts.

Table 5.3 init<SID>.ora Configuration Parameters

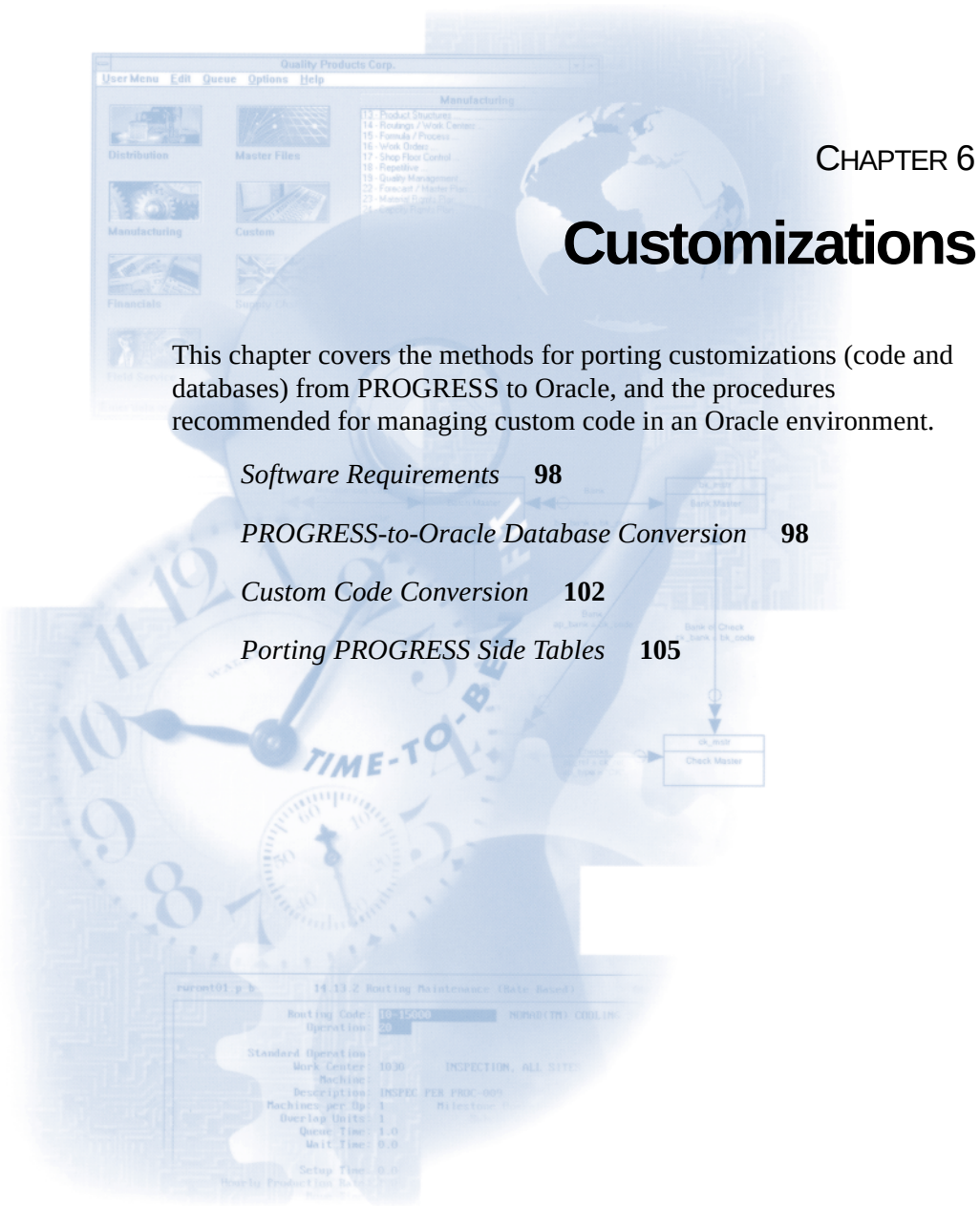
| Parameters<br>Explanation | Value |
|---|---|
| **optimizer_mode** | Rule |
| The type of optimizer mechanism used by Oracle. QAD recommends setting the optimizer_mode to `rule` because it works best for MFG/PRO. The default is `choose`. | |
| **open_cursors** | 525 |
| The maximum number of open cursors per session. The default value may not be sufficient for MFG/PRO. | |
| **processes** | Number of users plus 20. |
| The maximum number of Oracle processes, including background and shadow processes, that can be started. The default is normally inadequate. | |
| **dml_locks** | 6 to 7 times the number of users. |
| Sets the maximum number of DML locks that Oracle can handle. The Oracle default is 4 times the number of users. | |
| **enque_resources** | 50-100 more than the number of dml_locks. |
| Sets the maximum number of locks that Oracle can handle. | |

## Parameters for Tuning Preparations

Parameters listed in this section are of importance when system tuning is performed.

**Table 5.4**
init<SID>.ora
Tuning Parameters

| Parameters | Values |
|---|---|
| **Explanation** | |
| **sql_trace** | False |
| Set to enable system-wide tracing of SQL statements. It is not advisable to set `sql_trace` at the system level; instead it should be set individually for the sessions for which tracing is needed. High performance impact due to the overhead of writing to the trace files. | |
| **timed_statistics** | False in production systems only, true in other cases, in particular for tuning and testing. |
| Specifies that Oracle statistics should include timings.<br>Generally, this parameter should be `true` when system testing or tuning is done. A performance impact is caused by Oracle requesting system time frequently, and is minor (a few percent) on most operating systems.<br>When running a production system, the value should be `false`, unless it is known that the performance impact is minor or the actual performance impact is accepted. | |



CHAPTER 6

# Customizations

This chapter covers the methods for porting customizations (code and databases) from PROGRESS to Oracle, and the procedures recommended for managing custom code in an Oracle environment.

# Software Requirements

## Oracle

- Development license (Full Use)
- Oracle7 Server
- SQL*DBA, SQL*Plus, PRO*C
- SQL*Net (optional)

## PROGRESS

- Development license (PROVISION)
- DataServer for PROGRESS
- DataServer for Oracle
- PROGRESS networking (for remote DataServers only)

# PROGRESS-to-Oracle Database Conversion

## Overview

1. Create empty Oracle database.
2. Run the PROGRESS to Oracle conversion utility (protoora.p).
3. Refine Oracle database definition language (DDL).
4. Load new Oracle DDL.
5. Load data into new Oracle tables.

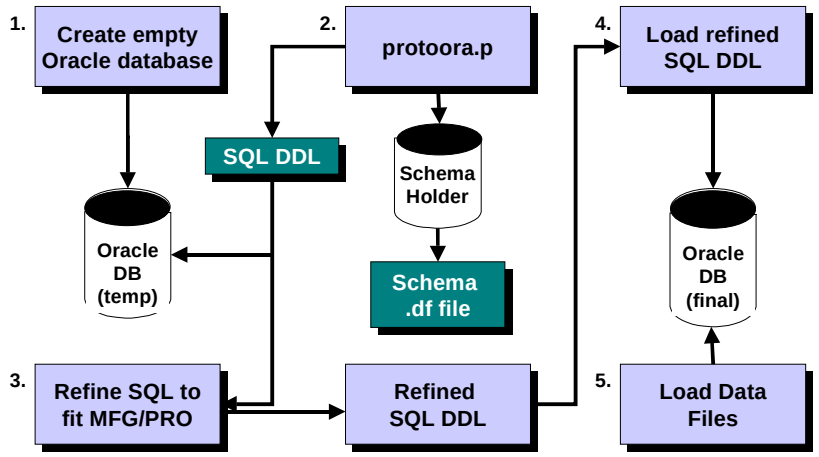



**Fig. 6.1** Database Conversion Flow Diagram

## Create Empty Oracle Database

- Use SQL*DBA to create an empty Oracle database
  - Small, need just enough room to hold table schema objects
  - Temporary, will be discarded
  - Single tablespace is sufficient (System)
  - Use the crdb1<SID>.sql script supplied by QAD as a model
- Start Oracle instance and mount database
- Start SQL*Net (if required)
  - tnsnames.ora
  - listener.ora
  - lsnrctl

### Use an Existing Oracle Database

- Create a new (temporary) Oracle user
- Assign default tablespace to TEMP
- Grant DBA privileges

- Do not run Oracle catalog scripts
- Drop user when done

## PROGRESS-to-Oracle Utility (protoora.p)

- Reads PROGRESS database schema
- Creates and runs SQL script to create Oracle schema objects
- Adds schema objects for extended 4GL functionality
  - FIND PREV/LAST
  - Cursor repositioning
  - Case-insensitive indexes
  - RECID() function
- “Pulls” the newly created Oracle schema back from Oracle
- Creates a new PROGRESS schema holder based on information from the Oracle schema

## Refine Oracle Data Definition Language (DDL)

- Newly created Oracle database should not be used to hold data
- All schema objects are placed in the default tablespace
- Widths for VARCHAR2 columns are not always appropriate
- Use QAD’s sizing utility and Oracle’s sizing formulas to adjust DDL
- Use the PROGRESS Data Administration utility to dump the schema holder to a .df file
- Locate the SQL script that contains the DDL for the Oracle schema ($ORACLE_SID.sql)
- Save both the .df and the .sql files, since they will serve as the baseline for changes

### Integrate the New .df File into MFG/PRO

- Edit the new .df file
- Delete ADD DATABASE statements
- Edit FOREIGN-OWNER statements to the correct Oracle owner value

- Load .df file into a COPY of the standard MFG/PRO schema holder
- Use PROGRESS Data Administration utility or
- Use dfloadx script from the MFG/PRO Oracle sub-directory

## Load New Oracle DDL

Three options:

- Add new schema objects to the existing instance
  - Add to existing MFG/PRO database
  - Add to new database
- Create a new Oracle database to hold schema objects

Decision depends on individual configurations

- Are there resources available to support additional instances?

## Load Data into Oracle

- Dump data from custom PROGRESS database to .d files
- Connect to new schema holder and Oracle database
  - Can connect to schema holder in R/O or single-user mode
- Load data using the Data Administration utility or
- Generate new MFG/PRO dump/load routines
  - Use utmkdl.p to create routines for the new tables
  - For large amounts of data, consider dropping the indexes during load
  - Cut and paste DROP/ADD INDEX from main sql script
  - Consider keeping the `progress_recid` index defined
  - Duplicate unique indexes will abort index creations
  - Add the indexes back after load is complete

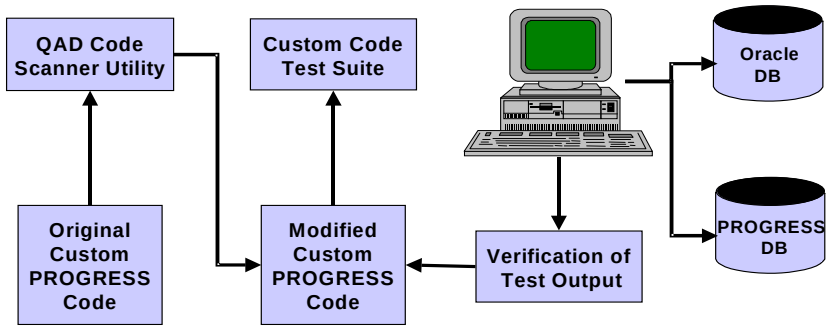
- Loading data via the DataServer correctly populates all columns
  - Populates shadow columns using uppercase characters
  - Uses an Oracle sequence to populate `progress_recid` column
  - Integrity constraints contained in the schema holder are followed
- SQL*Loader scripts would have to follow DataServer rules for loading data

# Custom Code Conversion

## Overview

- Run QAD's code scanner utility against custom code
- Evaluate and make appropriate source code changes
- Test custom code

**Fig. 6.2**
Custom Code Conversion Flow Diagram




## QAD's Code Scanner Utility

### Checks:

- SHARE-LOCK usage
  - ERROR for tight delete loops
  - WARNING for other FOR EACH loops

- New record availability to subprograms and display statements
  - Requires call to RECID() function to generate the SQL INSERT statement
- DO loops using NUM-DBS and NUM-ALIAS
  - Must check the DB-TYPE of the databases as they are processed
  - PROGRESS operations are not valid against foreign databases
  - Foreign databases must be processed as pairs (schema holder/ foreign database)
- Use of MATCHES function in WHERE clauses
  - PROGRESS wildcard characters are not supported by the Oracle7 server
  - Use an outer function such as STRING() to force client-side resolution
  - This can cause performance degradation
- Use of the NOT operator in WHERE clauses
  - Causes full table scans
  - Do not use “WHERE NOT logical_var”
  - Use “WHERE logical_var = false” instead
- Concatenation of literal strings in WHERE clauses
- “literal_string_one” + “literal_string_two” concatenates the strings correctly
- “literal_string_one” + char_var tries to arithmetically add the values together

### Does Not Check:

- WHERE Clauses
  - Absence of a WHERE clause will cause full table scans
  - Not a problem for small tables (e.g., control files)
  - Large problem for tables such as tr_hist

QAD may incorporate this check as a warning in future versions of the scanner.

## Utility Scanner Output

- Creates new copies of the source code programs
- Anticipated problem code is commented out
- Suggested code changes are inserted
- All modifications are flagged with /*ORACLE*/ comment strings

## Source Code Changes

- Evaluate suggested source code changes
  - Not all suggested changes will be appropriate
  - May have side-effects, especially in subprograms
- Make modifications to a working copy of your source code
  - Keep it separate from the code scanner copy
- If custom code is based on PROGRESS Version 6, there may be other changes required for PROGRESS Version 7 or Version 8
  - Shared streams
  - Frame stack order

## Test Custom Code

- Develop test suite
- Create concurrent PROGRESS and Oracle test environments
- Execute test suite in both environments
- Compare results for correctness and consistency
- Correct problems and repeat test suite

## MFG/PRO Access to External Oracle Data

- Use PROGRESS utility to create a schema holder for the Oracle database
- Schema holder will not support extended 4GL functionality:
  - FIND PREV/LAST
  - Case-insensitive indexes
  - Cursor repositioning

    - RECID() function
- Extended 4GL functionality can be added to a schema holder
    - Semi-automatically
    - Manually
- Documented in *DataServer for Oracle Guide* from PROGRESS Software
    - DataServer Tutorial chapter

### External Access to MFG/PRO Data

- QAD strongly suggests READ-ONLY access to MFG/PRO tables
    - Database integrity constraints are kept in the schema holder
    - By-passing the schema holder can lead to an inconsistent database
- Connect MFG/PRO database using normal Oracle connection methods
    - Internally on server machine
    - Using SQL*Net from client machines
- Must understand relationship between PROGRESS and Oracle schema objects:
    - Shadow columns and indexes
    - PROGRESS array element names
    - Oracle index components
    - Table relationships
- Can define read-only views to simplify end user access
    - Hide complexity from end user

## Porting PROGRESS Side Tables

There are several steps involved in porting side databases from PROGRESS to Oracle and integrating them with the standard MFG/PRO tables. Briefly, they are:

1. Create an empty PROGRESS database containing only the database definitions.
2. Create a temporary Oracle user to hold the new Oracle schema objects.
3. Run the PROGRESS utility protoora to create a new PROGRESS schema holder and the new Oracle schema objects.
4. Incorporate the new schema holder into the full MFG/PRO schema holder.
5. Incorporate the new Oracle schema objects into the full MFG/PRO Oracle schema.
6. Load data into the Oracle tables.
7. Compile programs.

## Requirements

- PROGRESS Database
- PROGRESS DataServer for Oracle
- Full 4GL access
- PROGRESS Networking (depending on client/server configuration)
- PROGRESS DataServer for Oracle Guide
- Oracle development license
- SQL*Net (depending on client/server configuration)
- Oracle DBA
- C Compiler (to link DataServer in Unix environments)

## Create Empty PROGRESS Database

Create a PROGRESS database to contain the schema objects to be added to Oracle. These schema objects include new tables, indexes, validation expressions, and sequences. The database does not need to contain any data. Do not include any standard MFG/PRO tables.

## Create Temporary Oracle User to Hold New Schema

Create an Oracle user in an existing Oracle database to be used temporarily to hold the new schema objects. This user should have the DEFAULT and TEMPORARY tablespaces set to a location other than the SYSTEM. The new Oracle user should have DBA privileges granted. Do not run the `catdbsyn.sql` script for this new user. Assuming that there is a tablespace called TEMP, you can use the following Oracle Server Manager commands to create a new user called TEMPUSER:

```
SVRMGR> CONNECT INTERNAL

SVRMGR> CREATE USER TEMPUSER IDENTIFIED BY TEMPUSER;

SVRMGR> ALTER USER TEMPUSER DEFAULT TABLESPACE TEMP
   TEMPORARY TABLESPACE TEMP;

SVRMGR> GRANT DBA TO TEMPUSER;
```

## Run the PROGRESS-to-Oracle Conversion

Start a PROGRESS session against the custom PROGRESS database created in step 1 above, in READ ONLY mode. Go to the Data Dictionary and select “DataServer - Oracle Utilities - Schema Migration Tools - Migrate DB to Oracle.” This brings up a screen that prompts you for information needed to migrate the database to Oracle. The following example uses the PROGRESS database new_pro to create a new schema holder called new_ora with the internal name of oradb. The codepage is specified as ibm850, although it could have other values (such as iso8859-1) depending on language values.

**Table 6.1** Migration Utility Entry Values

| | |
|---|---|
| Name of the original PROGRESS database: | new_pro |
| Other connect parameters for PROGRESS Db: | |
| Enter name of schema holder Database: | new_ora |
| Enter logical name for Oracle Database: | oradb |
| Enter the Oracle owner’s user name: | tempuser |
| Enter Oracle user’s password: | tempuser |
| Enter other connect parameters: | |
| Enter codepage for schema image: | ibm850 |

```
Create extended 4GL capable objects:                YES
Dump and load data:                                 NO
```

The utility loads the new Oracle schema objects into the Oracle database (owned by tempuser) and creates a schema holder database called new_ora. It also produces a SQL script called `new_ora.sql`, which contains the Oracle schema object definitions.

Exit out of the PROGRESS session, then restart PROGRESS against the schema holder. Go to the Data Dictionary and dump the database definitions to a .df file (`oradb.df` in the example above) and keep it with the `new_ora.sql` file. These two files are used in the next steps.

## Incorporate .df File into MFG/PRO Schema Holder

Edit the `oradb.df` file by removing the ADD DATABASE statement at the beginning of the file and the trailer information at the end of the file, leaving just the ADD SEQUENCE/TABLE/COLUMN/INDEX statements. Change the FOREIGN-OWNER values from tmpuser to the owner of the main MFG/PRO schema objects (by default qad), which can be found in a .df dump of the standard MFG/PRO oraempty.db schema holder.

These definitions must now be added to the standard MFG/PRO schema holder. There are a number of ways to do this, but the most straightforward method is to dump the qaddb.df file from the production oraempty.db schema holder. Add the new tables, sequences, etc., from above to the end of the qaddb.df file (just above the trailer information). Create a new empty PROGRESS database and load these definitions into it from the Data Dictionary. This is your new schema holder for your customized configuration, effectively replacing the standard oraempty.db schema holder. Do not overwrite the standard oraempty.db schema holder.

The above steps are for the schema holder located on the UNIX host. Modify the Windows client schema holder in the same manner as the UNIX schema holder.

After this step is complete, you are finished with the Oracle user created in step 2. The user can now be deleted from the Oracle database with the Server Manager command:

```
SVRMGR> CONNECT INTERNAL
SVRMGR> DROP USER TEMPUSER CASCADE;
```

## Incorporate Oracle Objects into MFG/PRO Schema

You should create two new tablespaces in Oracle to hold the new schema objects. The first should be used to hold new tables and the second for the indexes. They should be large enough to hold the data and indexes, with room to expand if necessary. As a general guideline, you can allocate approximately three times the space required to hold the data in the PROGRESS database. The actual amount of space required depends on the characteristics of the data. You can use the `crdb2<SID>.sql` script as a template for the CREATE TABLESPACE statements.

Load the new Oracle schema objects contained in the `new_ora.sql` file via SQL*Plus, making sure that you connect to the database as the correct Oracle user (by default qad). The command would be similar to:

```
sqlplus qad/qad < new_ora.sql
```

## Load Data into Oracle Tables

Using PROGRESS, connect to the custom PROGRESS database containing the data to be transferred into Oracle. From the Data Dictionary, dump the data for each table into .d files. Then start another PROGRESS session connected to the new custom schema holder. Connect to the Oracle database by either providing the information on the command line (or .pf file) or via the Data Dictionary. Use the Data Dictionary routines to load the previously created .d files into the new Oracle tables.

There are two problem areas that are likely to arise during the data loads. The first is duplicate unique key errors. The most likely cause of this error is because PROGRESS will allow duplicates to occur within unique keys if the duplicated key component has the UNKNOWN (?) value. Since Oracle does not allow this, the duplicate rows will not be inserted into the table. You need to identify these rows and ensure uniqueness before they can be inserted.

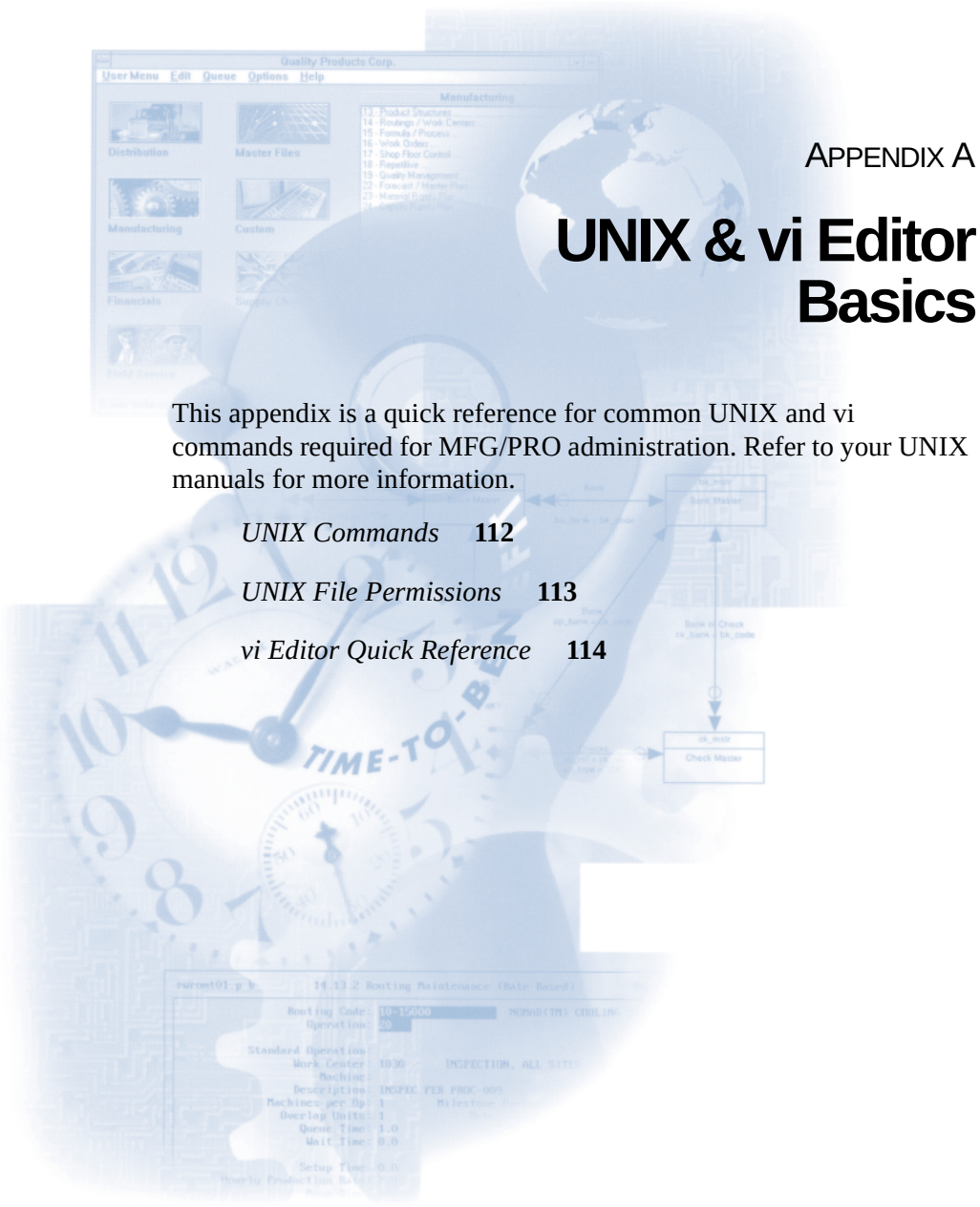
The second area is due to insufficient storage for character columns. Oracle does not have a character data type that is truly variable in size. The VARCHAR2 data type is variable from zero bytes to the maximum defined for the column (in the `new_ora.sql`) file. You can dynamically increase the column width by using the SQL*Plus command:

```
SQLPLUS> ALTER TABLE <table_name> MODIFY COLUMN
<column_name> VARCHAR2(xx);
```

where xx is the new width of the column. Identify the rows that did not load because of the width violation and load them again.

## Compile Programs

You now compile your MFG/PRO programs (both host and GUI versions) against the new schema holders.



APPENDIX A

# UNIX & vi Editor Basics

This appendix is a quick reference for common UNIX and vi commands required for MFG/PRO administration. Refer to your UNIX manuals for more information.

# UNIX Commands

**Table A.1**
Basic UNIX Commands

| **Directories** | |
|---|---|
| Display directory path | pwd |
| Change directory | cd |
| Change back one directory | cd .. |
| List files in directory | ls |
| List files (long format) | ls-l |
| List files (all) | ls-a |
| **File Permissions** | |
| Change permission | chmod *Permission Value* |
| Change group | chgrp *GroupName FileName* |
| Change owner | chown *UserID FileName* |

**Table A.2**
Other UNIX Commands

| | |
|---|---|
| Copy file | cp *OldName NewName* |
| Open vi editor | vi *FileName* |
| Search for lines in a file containing a given text string | grep *TextString FileName* |
| Search for files whose names contain a given text string | ls \| grep '*TextString*' |
| Display processes | ps |
| Display all processes (long format) | ps -ef |

# UNIX File Permissions

In order to manage file permissions, you must know how to read the UNIX file permission display. UNIX displays file permissions when you use the `ls -l` command:

```
-rwxr--r-- 1 pml  users  154  Nov 4 10:18  testfile
permissions owner group  size date & time filename
```

Permissions are defined for three user classes: owner, group, and others, as shown in below Table A.3.

**Table A.3** UNIX User Classes

| - | rwx | r-- | r-- |
|---|---|---|---|
| **Displays "d" if directory** | **Owner Permission (usually the creator of the file)** | **Group Permission (defined by system manager)** | **Other Permission (any other user)** |

For each class, there are three types of permission:

- Read (r)
- Write (w)
- Execute (x)

Each permission type has a value, used to change permissions.

**Table A.4** UNIX Permission Type Values

| Symbol | Type | Value |
|---|---|---|
| r | Read | 4 |
| w | Write | 2 |
| x | Execute | 1 |
| - | None | 0 |

To change permissions, you type the `chmod` command and three numbers, one for each class of user: the owner, group, and others. The permission number is the sum of the individual permission values. For example, you may want to change permissions as shown below.

**Table A.5**
UNIX Permission Numbers

| Class | Permission |
|---|---|
| Owner | Read (4) + Write (2) + Execute (1) |
| Group | Read (4) + Write (2) |
| Others | No permission |

The command you would enter is:

```
chmod 760
```

# vi Editor Quick Reference

**Table A.6**
Basic vi Editor Commands

## Basic Navigation

(arrow keys unavailable in update mode)

| | |
|---|---|
| `h` | Left |
| `j` | Down |
| `k` | Up |
| `l` | Right |

**Table A.7**
Last Line Mode Commands for vi

## Last Line Mode (:)

| | |
|---|---|
| **Search Functions** | |
| `/exp` | Search down to expression |
| `?exp` | Search up to expression |
| **Move and Insert Text** | |
| `:3,8d` | Delete lines 3 through 8 |
| `:4,9m 12` | Move lines 4 through 9 to line 12 |
| `:2,5t 13` | Copy lines 2 through 5 to line 13 |
| `:5,9w file` | Save lines 5 through 9 to file name |
| **Save Files and Exit** | |
| `:q` | Quit–no changes to file |
| `:q!` | Quit–do not save changes |
| `:wq` | Quit–save changes |
| `:wq!` | Force quit–save changes |
| **Search and Replace** | |

| | |
|---|---|
| | :1,22s/oldtext/newtext/g |
| **Control Edit Session** | |
| :set nu | Turn line numbers on |
| :set nonu | Turn line numbers off |
| :set all | Show all settings |
| :set list | Display invisible characters |
| :set wm=5 | Set margin 5 spaces from the right |

## Command Mode (Esc)

**Table A.8** Command Mode Commands for vi

| | |
|---|---|
| **Screen/Line Movements** | |
| 0 | Go to start of line |
| $ | Go to end of line |
| % | Go to matching brace/paren |
| G | Go to last line |
| 3G | Go to line 3 |
| ^J | Go down one line |
| ^P | Go up one line |
| **Word Movement** | |
| W | Go forward 1 word |
| 3W | Go forward 3 words |
| B | Go back 1 word |
| B3 | Go back 3 words |
| **Deleting Text** | |
| dw | Delete word |
| 3dw | Delete next 3 words |
| dd | Delete line |
| 3dd | Delete line and the next 2 lines |
| d0 | Delete to the beginning of line |
| dG | Delete to the end of line |
| x | Delete character |
| **Copy/Insert Text** | |
| Y | Copy line |
| 4y | Copy line and next 4 lines |

| | |
|---|---|
| P | Insert copied line below |
| p | Insert copied line above |
| **Cancel and Edit Functions** | |
| u | Undo last change |
| . | Redo last change |

## Command Mode (Esc)

| | |
|---|---|
| **Word Processing Functions** | |
| J | Join this and next line |
| 4J | Join this and next 4 lines |

**Table A.9**
Input Mode Commands for vi

## Input Mode

| | |
|---|---|
| **Add/Append Text** | |
| a | Append after cursor |
| A | Append to end of line |
| i | Insert before cursor |
| I | Insert at beginning of line |
| 5i | Insert text 5 times |
| **Add New Lines** | |
| o | Open new line below |
| O | Open new line above |
| **Change Text** | |
| cw | Change word |
| 3cw | Change next 3 words |
| C | Change line |

APPENDIX B

# MFG/PRO on Oracle Script Files

This appendix contains sample installation and initialization scripts used by Oracle for MFG/PRO on Oracle.

## crdb1085.sql

```
##
## crdb1O85F.sql - Create database O85F


## * Set terminal output and command echoing on; log output
of script
## *
#set termout on
#set echo on
spool /dr1/mfgpro/8.5fora/crdb1O85.lst

## * Start the <SID> instance (ORACLE_SID) here must be
set to <SID>
## *
connect internal
startup nomount pfile=/dr1/mfgpro/8.5fora/initO85F.ora

## * Create the <dbname> database.
## * SYSTEM tablespace configuration guidelines:
## *   General-Purpose ORACLE RDBMS          5Mb
## *   Additional dictionary for applications 10-50Mb
## * Redo Log File configuration guidelines:
## *   Use 3+ redo log files to relieve cannot allocate
new log... waits
## *  Use 100Kb per redo log file per connection to reduce
checkpoints
## *

create database "O85F"
    maxinstances 1
    maxlogfiles  16
```

```
    maxdatafiles 200
    character set "WE8ISO8859P1"
    datafile
   '/dr1/mfgpro/8.5fora/oradata/O85F/sys01O85F.dbf'
size    10M
    logfile
    '/dr4/oradata/O85F/log01O85F.dbf'      size    1M,
    '/dr5/oradata/O85F/log02O85F.dbf'      size    1M;

disconnect
spool off
```

# crdb2085.sql

```
##
## crdb2O85F.sql - Create database RBS and tablespaces
## * This script takes care off all commands necessary to
create
## * an OFA compliant database after the CREATE DATABASE
command has
## * succeeded.



## * Set terminal output and command echoing on; log output
of script
## *
#set termout on
#set echo on
spool /dr1/mfgpro/8.5fora/crdb2O85.lst


## * Database should already be started
#startup nomount pfile=/dr1/mfgpro/8.5fora/initO85F.ora


connect internal


## * install data dictionary views:
@/dr1/oracle/product/7.3.2.2/rdbms/admin/catalog.sql


## * Create additional rollback segment in SYSTEM before
creating tablespace.
## *
connect internal


create rollback segment r0 tablespace system
storage (initial 16k next 16k minextents 2 maxextents 20);


## * Use ALTER ROLLBACK SEGMENT ONLINE to put r0 online
```

```
without shutting
## * down and restarting the database.
## *
alter rollback segment r0 online;



## * Create a tablespace for rollback segments.
## * Rollback segment configuration guidelines:
## *   1 rollback segments for every 4 concurrent xactions.
## *    No more than 50 rollback segments.
## *    All rollback segments the same size.
## *    Between 2 and 4 homogeneously-sized extents per
rollback segment.
## * Attempt to keep rollback segments to 4 extents.
## *
create tablespace RBS datafile
     '/dr1/oradata/O85F/rbs01O85F.dbf'        size    8M
default storage (
     initial            128k
     next               128k
     pctincrease        0
     minextents         2
);


## * Create a tablespace for temporary segments.
## * Temporary tablespace configuration guidelines:
## *   Initial and next extent sizes = k * SORT_AREA_SIZE,
k in 1,2,3,etc.
## *
create tablespace TEMP datafile
      '/dr4/oradata/O85F/tmp01O85F.dbf'        size    20M
default storage (
      initial      256k
```

```
      next         256k
   pctincrease  0
);

## * Create a tablespace for database tools.
## *
create tablespace TOOLS datafile
     '/dr5/oradata/O85F/mgr01O85F.dbf'       size   15M;

## * Create a tablespace for miscellaneous database user
activity.
## *
create tablespace USERS datafile
      '/dr1/oradata/O85F/usr01O85F.dbf'        size   1M;

create tablespace MFGHELP datafile
     '/dr4/oradata/O85F/hlp01O85F.dbf'       size   140M;
create tablespace MFGHELP_IDX datafile
     '/dr5/oradata/O85F/hid01O85F.dbf'       size   167M;

create tablespace GUI datafile
      '/dr1/oradata/O85F/gui01O85F.dbf'        size   5M;
create tablespace GUI_IDX datafile
      '/dr4/oradata/O85F/gid01O85F.dbf'        size   5M;

create tablespace GLRPWRTR datafile
      '/dr5/oradata/O85F/glr01O85F.dbf'        size   4M;
create tablespace GLRPWRTR_IDX datafile
     '/dr1/oradata/O85F/rid01O85F.dbf'       size   12M;

create tablespace INTRASTAT datafile
      '/dr4/oradata/O85F/mfg13O85F.dbf'        size   2M;
create tablespace INTRASTAT_IDX datafile
```

```
      '/dr5/oradata/O85F/idx13O85F.dbf'        size   3M;

create tablespace TRANSACTION datafile
      '/dr1/oradata/O85F/mfg01O85F.dbf'       size   25M;
create tablespace TRANSACTION_IDX datafile
      '/dr4/oradata/O85F/idx01O85F.dbf'       size   35M;

create tablespace HISTORY datafile
      '/dr5/oradata/O85F/mfg02O85F.dbf'        size   6M;
create tablespace HISTORY_IDX datafile
      '/dr1/oradata/O85F/idx02O85F.dbf'        size   6M;

create tablespace STATIC datafile
      '/dr4/oradata/O85F/mfg03O85F.dbf'       size   15M;
create tablespace STATIC_IDX datafile
      '/dr5/oradata/O85F/idx03O85F.dbf'       size   30M;

create tablespace CONTROL datafile
      '/dr1/oradata/O85F/mfg04O85F.dbf'        size   2M;
create tablespace CONTROL_IDX datafile
      '/dr4/oradata/O85F/idx04O85F.dbf'        size   2M;

create tablespace REFERENCE datafile
      '/dr5/oradata/O85F/mfg05O85F.dbf'        size   5M;
create tablespace REFERENCE_IDX datafile
      '/dr1/oradata/O85F/idx05O85F.dbf'        size   6M;

create tablespace WORKFILE datafile
      '/dr4/oradata/O85F/mfg06O85F.dbf'        size   5M;
create tablespace WORKFILE_IDX datafile
      '/dr5/oradata/O85F/idx06O85F.dbf'        size   5M;

create tablespace GLTDET datafile
```

```
      '/dr1/oradata/O85F/mfg07O85F.dbf'        size   2M;
create tablespace GLTDET_IDX datafile
      '/dr4/oradata/O85F/idx07O85F.dbf'        size   5M;

create tablespace GLTRHIST datafile
     '/dr5/oradata/O85F/mfg08O85F.dbf'        size   10M;
create tablespace GLTRHIST_IDX datafile
     '/dr1/oradata/O85F/idx08O85F.dbf'        size   20M;

create tablespace TRHIST datafile
     '/dr4/oradata/O85F/mfg09O85F.dbf'        size   15M;
create tablespace TRHIST_IDX datafile
     '/dr5/oradata/O85F/idx09O85F.dbf'        size   15M;

create tablespace TRGLDET datafile
      '/dr1/oradata/O85F/mfg10O85F.dbf'        size   4M;
create tablespace TRGLDET_IDX datafile
      '/dr4/oradata/O85F/idx10O85F.dbf'        size   6M;

create tablespace MRPDET datafile
      '/dr5/oradata/O85F/mfg11O85F.dbf'        size   5M;
create tablespace MRPDET_IDX datafile
      '/dr1/oradata/O85F/idx11O85F.dbf'        size   8M;

create tablespace QADWKFL datafile
      '/dr4/oradata/O85F/mfg12O85F.dbf'        size   5M;
create tablespace QADWKFL_IDX datafile
      '/dr5/oradata/O85F/idx12O85F.dbf'        size   5M;

create tablespace COMPCONF datafile
      '/dr1/oradata/O85F/cct01O85F.dbf'        size   5M;
create tablespace COMPCONF_IDX datafile
      '/dr4/oradata/O85F/cci01O85F.dbf'        size   5M;
```

```
## * Create rollback segments.
## *
create rollback segment r01 tablespace rbs
                            storage ( optimal 1M );
create rollback segment r02 tablespace rbs
                            storage ( optimal 1M );
create rollback segment r03 tablespace rbs
                            storage ( optimal 1M );
create rollback segment r04 tablespace rbs
                            storage ( optimal 1M );
create rollback segment r05 tablespace rbs
                            storage ( optimal 1M );
create rollback segment r06 tablespace rbs
                            storage ( optimal 1M );
create rollback segment r07 tablespace rbs
                            storage ( optimal 1M );
create rollback segment r08 tablespace rbs
                            storage ( optimal 1M );


## * Use ALTER ROLLBACK SEGMENT ONLINE to put rollback
segments online
## * without shutting down and restarting the database.
Only put one
## * of the rollback segments online at this time so that
it will always
## * be the one used.  When the user shuts down the
database and starts
## * it up with initSID.ora, all four will be brought
online.
## *
alter rollback segment r01 online;
alter rollback segment r02 online;
alter rollback segment r03 online;
```

```
alter rollback segment r04 online;
alter rollback segment r05 online;
alter rollback segment r06 online;
alter rollback segment r07 online;
alter rollback segment r08 online;


## * Since we've created and brought online 2 more rollback
segments,
## * we no longer need the second rollback segment in the
SYSTEM tablespace.


alter rollback segment r0 offline;
drop rollback segment r0;


## * Alter SYS and SYSTEM users.
## *
alter user sys temporary tablespace temp;
#revoke resource from system;
#revoke resource on system from system;
#grant resource on tools to system;
alter user system default tablespace tools temporary
tablespace temp;


## * For each DBA user, run DBA synonyms SQL script.  Don't
forget that EACH
## * DBA USER created in the future needs dba_syn.sql run
from its account.
## *
connect system/manager
@/dr1/oracle/product/7.3.2.2/rdbms/admin/catdbsyn.sql
create user qad identified by qad;
grant dba to qad;
alter user qad default tablespace tools temporary
tablespace temp;
connect qad/qad
```

```
@/dr1/oracle/product/7.3.2.2/rdbms/admin/catdbsyn.sql

spool off
```

## ohpempty.sql

```
spool ohpempty.lst;


DROP SEQUENCE flhd_det_SEQ;

CREATE SEQUENCE flhd_det_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;

DROP TABLE flhd_det;

CREATE TABLE flhd_det(

      U##flhd_call_pg varchar2 (30),

      flhd_call_pg varchar2 (30),

      flhd_text varchar2 (256),

      U##flhd_field varchar2 (80),

      flhd_field varchar2 (80),

      U##flhd_lang varchar2 (8),

      flhd_lang varchar2 (8),

      flhd_line number,

      flhd_user1 varchar2 (80),

      flhd_user2 varchar2 (80),

      U##flhd_type varchar2 (80),

      flhd_type varchar2 (80),

      flhd__qad01 varchar2 (80),

      progress_recid number null

) TABLESPACE mfghelp STORAGE (initial 10K next 1M
pctincrease 0);

CREATE UNIQUE INDEX flhd_det##progress_recid on flhd_det
(progress_recid) TABLESPACE mfghelp_idx STORAGE (initial
10K next 1M pctincrease 0);

CREATE UNIQUE INDEX flhd_det##flhd_type on flhd_det
(U##flhd_lang, U##flhd_field, U##flhd_call_pg,
U##flhd_type, flhd_line) TABLESPACE mfghelp_idx STORAGE
(initial 10K next 1M pctincrease 0);

DROP SEQUENCE flhk_mstr_SEQ;

CREATE SEQUENCE flhk_mstr_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;

DROP TABLE flhk_mstr;
```

```
CREATE TABLE flhk_mstr(
      U##flhk_lang varchar2 (8) NULL,
      flhk_lang varchar2 (8) NULL,
      U##flhk_keyword varchar2 (30) NULL,
      flhk_keyword varchar2 (30) NULL,
      flhk_text##1 varchar2 (80) NULL,
      flhk_text##2 varchar2 (80) NULL,
      flhk_text##3 varchar2 (80) NULL,
      flhk_text##4 varchar2 (80) NULL,
      flhk_text##5 varchar2 (80) NULL,
      flhk_text##6 varchar2 (80) NULL,
      flhk_text##7 varchar2 (80) NULL,
      flhk_text##8 varchar2 (80) NULL,
      flhk_text##9 varchar2 (80) NULL,
      flhk_text##10 varchar2 (80) NULL,
      flhk_text##11 varchar2 (80) NULL,
      flhk_text##12 varchar2 (80) NULL,
      flhk_text##13 varchar2 (80) NULL,
      flhk_text##14 varchar2 (80) NULL,
      flhk_text##15 varchar2 (80) NULL,
      flhk__qad01 varchar2 (80),
      flhk_user1 varchar2 (80),
      flhk_user2 varchar2 (80),
      progress_recid number null
) TABLESPACE mfghelp;
CREATE UNIQUE INDEX flhk_mstr##progress_recid on
flhk_mstr (progress_recid) TABLESPACE mfghelp_idx;
CREATE UNIQUE INDEX flhk_mstr##flhk_keyword on flhk_mstr
(U##flhk_keyword, U##flhk_lang) TABLESPACE mfghelp_idx;
CREATE UNIQUE INDEX flhk_mstr##flhk_lang on flhk_mstr
(U##flhk_lang, U##flhk_keyword) TABLESPACE mfghelp_idx;
DROP SEQUENCE flhm_mst_SEQ;
CREATE SEQUENCE flhm_mst_SEQ START WITH 1 INCREMENT BY
```

```
CACHE 75;
DROP TABLE flhm_mst;
CREATE TABLE flhm_mst(
      U##flhm_lang varchar2 (8),
      flhm_lang varchar2 (8),
      U##flhm_field varchar2 (80),
      flhm_field varchar2 (80),
      U##flhm_call_pg varchar2 (30),
      flhm_call_pg varchar2 (30),
      flhm_status varchar2 (80),
      U##flhm_lnk_fld varchar2 (80),
      flhm_lnk_fld varchar2 (80),
      U##flhm_lnk_pgm varchar2 (30),
      flhm_lnk_pgm varchar2 (30),
      flhm_sub number,
      flhm_label varchar2 (80),
      flhm_col_label varchar2 (80),
      flhm_class varchar2 (80),
      flhm_type varchar2 (30),
      flhm_format varchar2 (80),
      flhm_len number,
      flhm_validate varchar2 (80),
      flhm_default varchar2 (80),
      flhm_priority number,
      flhm_ll_code number,
      flhm_user1 varchar2 (80),
      flhm_user2 varchar2 (80),
      U##flhm_key_words varchar2 (80),
      flhm_key_words varchar2 (80),
      flhm__qad01 varchar2 (80),
      flhm__qad02 varchar2 (80),
      progress_recid number null
) TABLESPACE mfghelp STORAGE (initial 10K next 512K
```

```
pctincrease 0);

CREATE UNIQUE INDEX flhm_mst##progress_recid on flhm_mst
(progress_recid) TABLESPACE mfghelp_idx STORAGE (initial
10K next 512K pctincrease 0);

CREATE UNIQUE INDEX flhm_mst##flhm_call_pg on flhm_mst
(U##flhm_lang, U##flhm_call_pg, U##flhm_field) TABLESPACE
mfghelp_idx STORAGE (initial 10K next 512K pctincrease 0);

CREATE UNIQUE INDEX flhm_mst##flhm_key_words on flhm_mst
(U##flhm_lang, U##flhm_key_words, U##flhm_field,
U##flhm_call_pg) TABLESPACE mfghelp_idx STORAGE (initial
10K next 512K pctincrease 0);

CREATE UNIQUE INDEX flhm_mst##flhm_lang on flhm_mst
(U##flhm_lang, U##flhm_field, U##flhm_call_pg) TABLESPACE
mfghelp_idx STORAGE (initial 10K next 512K pctincrease 0);

CREATE UNIQUE INDEX flhm_mst##flhm_lnk on flhm_mst
(U##flhm_lang, U##flhm_lnk_fld, U##flhm_lnk_pgm,
U##flhm_field, U##flhm_call_pg) TABLESPACE mfghelp_idx
STORAGE (initial 10K next 512K pctincrease 0);

DROP SEQUENCE xd_mstr_SEQ;

CREATE SEQUENCE xd_mstr_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;

DROP TABLE xd_mstr;

CREATE TABLE xd_mstr(

      U##xd_exec varchar2 (30),

      xd_exec varchar2 (30),

      xd_msg_nbr number,

      xd_user1 varchar2 (80),

      xd_user2 varchar2 (80),

      xd__qad01 varchar2 (80),

      progress_recid number null

) TABLESPACE mfghelp STORAGE (initial 10K next 256K
pctincrease 0);

CREATE UNIQUE INDEX xd_mstr##progress_recid on xd_mstr
(progress_recid) TABLESPACE mfghelp_idx STORAGE (initial
10K next 256K pctincrease 0);

CREATE UNIQUE INDEX xd_mstr##xd_exec on xd_mstr
(U##xd_exec, xd_msg_nbr) TABLESPACE mfghelp_idx STORAGE
(initial 10K next 256K pctincrease 0);

CREATE INDEX xd_mstr##xd_nbr on xd_mstr (xd_msg_nbr,
U##xd_exec, progress_recid) TABLESPACE mfghelp_idx
```

```
STORAGE (initial 10K next 256K pctincrease 0);

DROP SEQUENCE xf_mstr_SEQ;

CREATE SEQUENCE xf_mstr_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;

DROP TABLE xf_mstr;

CREATE TABLE xf_mstr(
      U##xf_pgm varchar2 (30) NULL,
      xf_pgm varchar2 (30) NULL,
      xf_dbname varchar2 (30),
      U##xf_file varchar2 (30) NULL,
      xf_file varchar2 (30) NULL,
      U##xf_field varchar2 (30) NULL,
      xf_field varchar2 (30) NULL,
      xf_create number NULL,
      xf_delete number NULL,
      xf_search number NULL,
      xf_access number NULL,
      xf_update number NULL,
      xf_user1 varchar2 (80),
      xf_user2 varchar2 (80),
      xf__qad01 varchar2 (80),
      progress_recid number null
) TABLESPACE mfghelp STORAGE (initial 10K next 1M
pctincrease 0);

CREATE UNIQUE INDEX xf_mstr##progress_recid on xf_mstr
(progress_recid) TABLESPACE mfghelp_idx STORAGE (initial
10K next 1M pctincrease 0);

CREATE INDEX xf_mstr##xf_field on xf_mstr (U##xf_field,

U##xf_file, U##xf_pgm, progress_recid) TABLESPACE

mfghelp_idx STORAGE (initial 10K next 1M pctincrease 0);

CREATE UNIQUE INDEX xf_mstr##xf_pgm on xf_mstr (U##xf_pgm,

U##xf_file, U##xf_field) TABLESPACE mfghelp_idx STORAGE

(initial 10K next 1M pctincrease 0);

DROP SEQUENCE xmi_mstr_SEQ;
```

```
CREATE SEQUENCE xmi_mstr_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;
DROP TABLE xmi_mstr;
CREATE TABLE xmi_mstr(
      U##xmi_exec varchar2 (30) NULL,
      xmi_exec varchar2 (30) NULL,
      U##xmi_index varchar2 (30) NULL,
      xmi_index varchar2 (30) NULL,
      xmi_count number,
      xmi__qad01 varchar2 (80),
      xmi_user1 varchar2 (80),
      xmi_user2 varchar2 (80),
      U##xmi_file varchar2 (30) NULL,
      xmi_file varchar2 (30) NULL,
      progress_recid number null
) TABLESPACE mfghelp;
CREATE UNIQUE INDEX xmi_mstr##progress_recid on xmi_mstr
(progress_recid) TABLESPACE mfghelp_idx;
CREATE UNIQUE INDEX xmi_mstr##xmi_exec on xmi_mstr
(U##xmi_exec, U##xmi_file, U##xmi_index) TABLESPACE
mfghelp_idx;
CREATE UNIQUE INDEX xmi_mstr##xmi_file on xmi_mstr
(U##xmi_file, U##xmi_index, U##xmi_exec) TABLESPACE
mfghelp_idx;
DROP SEQUENCE xm_mstr_SEQ;
CREATE SEQUENCE xm_mstr_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;
DROP TABLE xm_mstr;
CREATE TABLE xm_mstr(
      U##xm_exec varchar2 (30),
      xm_exec varchar2 (30),
      U##xm_file varchar2 (30),
      xm_file varchar2 (30),
```

```
        U##xm_field varchar2 (30),
        xm_field varchar2 (30),
        xm_create number,
        xm_delete number,
        xm_search number,
        xm_access number,
        xm_update number,
        xm_user1 varchar2 (80),
        xm_user2 varchar2 (80),
        xm__qad01 varchar2 (80),
        progress_recid number null
) TABLESPACE mfghelp STORAGE (initial 10K next 1M
pctincrease 0);
CREATE UNIQUE INDEX xm_mstr##progress_recid on xm_mstr
(progress_recid) TABLESPACE mfghelp_idx STORAGE (initial
10K next 1M pctincrease 0);
CREATE UNIQUE INDEX xm_mstr##xm_exec on xm_mstr
(U##xm_exec, U##xm_file, U##xm_field) TABLESPACE
mfghelp_idx STORAGE (initial 10K next 1M pctincrease 0);
CREATE INDEX xm_mstr##xm_field on xm_mstr (U##xm_field,
U##xm_file, U##xm_exec, progress_recid) TABLESPACE
mfghelp_idx STORAGE (initial 10K next 1M pctincrease 0);
DROP SEQUENCE xpi_mstr_SEQ;
CREATE SEQUENCE xpi_mstr_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;
DROP TABLE xpi_mstr;
CREATE TABLE xpi_mstr(
        U##xpi_pgm varchar2 (30) NULL,
        xpi_pgm varchar2 (30) NULL,
        U##xpi_index varchar2 (30) NULL,
        xpi_index varchar2 (30) NULL,
        xpi_count number,
        xpi__qad01 varchar2 (80),
```

```
      xpi_user1 varchar2 (80),
      xpi_user2 varchar2 (80),
      U##xpi_file varchar2 (30) NULL,
      xpi_file varchar2 (30) NULL,
      progress_recid number null
) TABLESPACE mfghelp;
CREATE UNIQUE INDEX xpi_mstr##progress_recid on xpi_mstr
(progress_recid) TABLESPACE mfghelp_idx;
CREATE UNIQUE INDEX xpi_mstr##xpi_file on xpi_mstr
(U##xpi_file, U##xpi_index, U##xpi_pgm) TABLESPACE
mfghelp_idx;
CREATE UNIQUE INDEX xpi_mstr##xpi_pgm on xpi_mstr
(U##xpi_pgm, U##xpi_file, U##xpi_index) TABLESPACE
mfghelp_idx;
DROP SEQUENCE xp_mstr_SEQ;
CREATE SEQUENCE xp_mstr_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;
DROP TABLE xp_mstr;
CREATE TABLE xp_mstr(
      U##xp_pgm varchar2 (30) NULL,
      xp_pgm varchar2 (30) NULL,
      xp_order number NULL,
      xp_type varchar2 (8) NULL,
      U##xp_parent varchar2 (30) NULL,
      xp_parent varchar2 (30) NULL,
      xp_line number NULL,
      U##xp_component varchar2 (80) NULL,
      xp_component varchar2 (80) NULL,
      xp_perm number NULL,
      xp_args varchar2 (1280),
      xp_user1 varchar2 (80),
      xp_user2 varchar2 (80),
      xp__qad01 varchar2 (80),
```

```
      xp__qad02 varchar2 (80),
      progress_recid number null
) TABLESPACE mfghelp STORAGE (initial 10K next 512K
pctincrease 0);
CREATE UNIQUE INDEX xp_mstr##progress_recid on xp_mstr
(progress_recid) TABLESPACE mfghelp_idx STORAGE (initial
10K next 512K pctincrease 0);
CREATE INDEX xp_mstr##xp_component on xp_mstr
(U##xp_component, U##xp_pgm, U##xp_parent, xp_order,
progress_recid) TABLESPACE mfghelp_idx STORAGE (initial
10K next 512K pctincrease 0);
CREATE INDEX xp_mstr##xp_perm on xp_mstr (xp_perm,
U##xp_pgm, U##xp_component, progress_recid) TABLESPACE
mfghelp_idx STORAGE (initial 10K next 512K pctincrease 0);
CREATE UNIQUE INDEX xp_mstr##xp_pgm on xp_mstr (U##xp_pgm,
xp_order, U##xp_parent, U##xp_component) TABLESPACE
mfghelp_idx STORAGE (initial 10K next 512K pctincrease 0);

spool off;
```

## oraempty.sql (partial)

```
spool oraempty.sql

DROP SEQUENCE abd_det_SEQ;
CREATE SEQUENCE abd_det_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;
DROP TABLE abd_det;
CREATE TABLE abd_det(
      U##abd_book varchar2 (80),
      abd_book varchar2 (80),
      U##abd_asset varchar2 (30),
      abd_asset varchar2 (30),
      abd_type varchar2 (30),
      abd_active number,
      abd_cost number (38,10),
      abd_curr_cost number (35,10),
      abd_ex_rate number (38,10),
      abd_ent_ex number (20,10),
      abd_life_yr number NULL,
      abd_life_mnth number NULL,
      abd_rem_yr number NULL,
      abd_rem_mnth number NULL,
      abd_method varchar2 (80),
      abd_salvage number (38,10),
      abd_dtd number (38,10),
      abd_ytd number (38,10),
      abd_pd_depr number (35,10),
      abd_last_depr date NULL,
      abd_expense number (38,10),
      abd_bonus number (38,10),
      abd_credit number (38,10),
      abd_cr_amt number (38,10),
```

```
        abd_conv varchar2 (80),
        abd_db_pct number (38,10),
        abd_uint1 number,
        abd_uint2 number,
        abd_uint3 number,
        abd_udec1 number (38,10),
        abd_udec2 number (38,10),
        abd_udec3 number (38,10),
        abd_prior_ast number (38,10),
        abd_user1 varchar2 (80),
        abd_user2 varchar2 (80),
        abd__qad01 varchar2 (80),
        abd__qad02 varchar2 (80),
        abd_rep_pct number (38,10),
        abd_date date,
        abd_periods number,
        abd_ytd_total number (38,10),
        abd_first_mnth number (38,10),
        abd_rt_date date,
        abd__qad03 varchar2 (80),
        abd_sched_depr number (38,10) NULL,
        abd_udec4 number (38,10) NULL,
        progress_recid number null
) TABLESPACE transaction;
CREATE UNIQUE INDEX abd_det##progress_recid on abd_det
(progress_recid) TABLESPACE transaction_idx;
CREATE UNIQUE INDEX abd_det##abd_active on abd_det
(abd_active, U##abd_asset, U##abd_book) TABLESPACE
transaction_idx;
CREATE UNIQUE INDEX abd_det##abd_asset on abd_det
(U##abd_asset, U##abd_book) TABLESPACE transaction_idx;
DROP SEQUENCE abs_mstr_SEQ;
.....
```

```
.....
DROP SEQUENCE ytd_det_SEQ;
CREATE SEQUENCE ytd_det_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;
DROP TABLE ytd_det;
CREATE TABLE ytd_det(
      U##ytd_addr varchar2 (80),
      ytd_addr varchar2 (80),
      ytd_year number,
      U##ytd_earn varchar2 (30),
      ytd_earn varchar2 (30),
      U##ytd_ded varchar2 (80),
      ytd_ded varchar2 (80),
      ytd_amt number (38,10),
      ytd_type varchar2 (30),
      ytd_units number (38,10),
      U##ytd_wk_loc varchar2 (30),
      ytd_wk_loc varchar2 (30),
      U##ytd_state varchar2 (30),
      ytd_state varchar2 (30),
      U##ytd_bank varchar2 (30),
      ytd_bank varchar2 (30),
      ytd_user1 varchar2 (80),
      ytd_user2 varchar2 (80),
      progress_recid number null
) TABLESPACE transaction;
CREATE UNIQUE INDEX ytd_det##progress_recid on ytd_det
(progress_recid) TABLESPACE transaction_idx;
CREATE INDEX ytd_det##ytd_earn on ytd_det (U##ytd_addr,
U##ytd_state, U##ytd_earn, progress_recid) TABLESPACE
transaction_idx;
CREATE UNIQUE INDEX ytd_det##ytd_index on ytd_det
(U##ytd_addr, ytd_year, U##ytd_bank, U##ytd_wk_loc,
U##ytd_ded, U##ytd_earn) TABLESPACE transaction_idx;
```

```
DROP SEQUENCE cmt_sq01;

CREATE SEQUENCE cmt_sq01  START WITH 1 INCREMENT BY 1
MAXVALUE 99999999 MINVALUE 0 CYCLE  CACHE 75;

DROP SEQUENCE glt_sq01;

CREATE SEQUENCE glt_sq01  START WITH 1 INCREMENT BY 1
MAXVALUE 999999 MINVALUE 0 CYCLE  CACHE 75;

DROP SEQUENCE glt_sq02;

CREATE SEQUENCE glt_sq02  START WITH 1 INCREMENT BY 1
MAXVALUE 999999 MINVALUE 0 CYCLE  CACHE 75;

DROP SEQUENCE glt_sq03;

CREATE SEQUENCE glt_sq03  START WITH 1 INCREMENT BY 1
MAXVALUE 999999 MINVALUE 0 CYCLE  CACHE 75;

DROP SEQUENCE glt_sq04;

CREATE SEQUENCE glt_sq04  START WITH 1 INCREMENT BY 1
MAXVALUE 999999 MINVALUE 0 CYCLE  CACHE 75;

DROP SEQUENCE glt_sq05;

CREATE SEQUENCE glt_sq05  START WITH 1 INCREMENT BY 1
MAXVALUE 999999 MINVALUE 0 CYCLE  CACHE 75;

DROP SEQUENCE glt_sq06;

CREATE SEQUENCE glt_sq06  START WITH 1 INCREMENT BY 1
MAXVALUE 999999 MINVALUE 0 CYCLE  CACHE 75;

DROP SEQUENCE glt_sq07;

CREATE SEQUENCE glt_sq07  START WITH 1 INCREMENT BY 1
MAXVALUE 999999 MINVALUE 0 CYCLE  CACHE 75;

DROP SEQUENCE ieh_sq01;

CREATE SEQUENCE ieh_sq01  START WITH 1 INCREMENT BY 1
MAXVALUE 99999999 MINVALUE 0 CYCLE  CACHE 75;

DROP SEQUENCE op_sq01;

CREATE SEQUENCE op_sq01  START WITH 1 INCREMENT BY 1
MAXVALUE 99999999 MINVALUE 0 CYCLE  CACHE 75;

DROP SEQUENCE rcc_sq01;

CREATE SEQUENCE rcc_sq01  START WITH 1 INCREMENT BY 1
MAXVALUE 99999999 MINVALUE 0 CYCLE  CACHE 75;

DROP SEQUENCE rcc_sq02;

CREATE SEQUENCE rcc_sq02  START WITH 1 INCREMENT BY 1
MAXVALUE 99999999 MINVALUE 0 CYCLE  CACHE 75;

DROP SEQUENCE te_sq01;
```

```
CREATE SEQUENCE te_sq01  START WITH 1 INCREMENT BY 1
MAXVALUE 1000000 MINVALUE 1 CYCLE  CACHE 75;

DROP SEQUENCE tr_sq01;

CREATE SEQUENCE tr_sq01  START WITH 1 INCREMENT BY 1
MAXVALUE 99999999 MINVALUE 0 CYCLE  CACHE 75;

DROP SEQUENCE woc_sq01;

CREATE SEQUENCE woc_sq01  START WITH 1 INCREMENT BY 1
MAXVALUE 99999999 MINVALUE 0 CYCLE  CACHE 75;


spool off


init085.ora

#

# initO85F.ora - oracle instance parameter file




# include database configuration parameters

ifile = /dr1/mfgpro/8.5fora/config.O85F


open_cursors = 384

rollback_segments            =

(r01,r02,r03,r04,r05,r06,r07,r08)


# NLS Parameters

  NLS_LANGUAGE = "AMERICAN"

  NLS_TERRITORY = "AMERICA"

  NLS_NUMERIC_CHARACTERS = "."


# tuning parameters


  db_files = 50


  db_file_multiblock_read_count = 8              # SMALL
```

```
# db_file_multiblock_read_count = 16              # MEDIUM
# db_file_multiblock_read_count = 32               # LARGE

  db_block_buffers = 200                           # SMALL
# db_block_buffers = 550                          # MEDIUM
# db_block_buffers = 3200                          # LARGE

  shared_pool_size = 3500000                       # SMALL
# shared_pool_size = 6000000                      # MEDIUM
# shared_pool_size = 9000000                       # LARGE

  log_checkpoint_interval = 10000

  processes = 50                                   # SMALL
# processes = 100                                 # MEDIUM
# processes = 200                                  # LARGE

  dml_locks = 100                                  # SMALL
# dml_locks = 200                                 # MEDIUM
# dml_locks = 500                                  # LARGE

  log_buffer = 8192                                # SMALL
# log_buffer = 32768                              # MEDIUM
# log_buffer = 163840                              # LARGE

  sequence_cache_entries = 10                      # SMALL
# sequence_cache_entries = 30                     # MEDIUM
# sequence_cache_entries = 100                     # LARGE

  sequence_cache_hash_buckets = 1                  # SMALL
# sequence_cache_hash_buckets = 23                # MEDIUM
# sequence_cache_hash_buckets = 89                 # LARGE
```

```
# audit_trail = true          # if you want auditing
# timed_statistics = true    # if you want timed statistics


  max_dump_file_size = 10240 # limit trace file size to 5M
ea
# log_archive_start = true   # if you want auto archiving


  mts_dispatchers="ipc,1"
  mts_max_dispatchers=10
  mts_servers=1
  mts_max_servers=10
  mts_service=O85F
  mts_listener_address="(ADDRESS=(PROTOCOL=ipc)
    (KEY=O85F))"
```

## client.demo

```
#!/bin/sh
# Script to start multi-user session of MFG/PRO


# tokens:
# &DLC = Progress Directory
# &CLIENT-DB-CONNECT = command line to connect to each db
in dbset


# SCCS: @(#)client.tpl1.1 04/02/97 13:00:50 >


stty intr '^c'
DLC=${DLC:-/dr1/dlc81a};export DLC
PATH=$PATH:$DLC;export PATH
ORACLE_HOME=${ORACLE_HOME}; export ORACLE_HOME
ORACLE_SID=${ORACLE_SID}; export ORACLE_SID
PROMSGS=$DLC/promsgs;export PROMSGS
PROTERMCAP=$DLC/protermcap;export PROTERMCAP
PS1='$$ ';export PS1
PROPATH=${PROPATH:-.,/dr1/mfgpro/8.5fora,/dr1/mfgpro/
8.5fora/us/bbi};export PROPATH


NLS_NUMERIC_CHARACTERS="."; export
NLS_NUMERIC_CHARACTERS
NLS_LANG="AMERICAN_AMERICA.WE8ISO8859P1";export NLS_LANG


#
# Set terminal type.
#
if [ ${TERM:-NULL} = NULL ]
then
    echo
    echo "Please enter your terminal type: \c"
```

```
    read TERM
    export TERM
fi


#
# Start MFG/PRO.
#
cd  # change to home directory


# exec $DLC/bin/_progres &DB etc
exec $DLC/bin/_progres \
  /dr1/mfgpro/8.5fora/demo85f -RO -znotrim -trig triggers
\
-db O85F -dt ORACLE -U qad -P qad -c 250 -Dsrv
qt_no_lookahead \
-cpstream ibm850 -cpinternal iso8859-1 -cprcodein
iso8859-1 -cpcoll basic  -Bt 350 -D 100 -mmax 3000 -nb 200
-s 48 -p mf.p;
```

## client.tpl

```
#!/bin/sh
# Script to start multi-user session of MFG/PRO

# tokens:
# &DLC = Progress Directory
# &CLIENT-DB-CONNECT = command line to connect to each db
in dbset

# SCCS: @(#)client.tpl1.1 04/02/97 13:00:50 >

stty intr '^c'
DLC=${DLC:-&DLC};export DLC
PATH=$PATH:$DLC;export PATH
ORACLE_HOME=${ORACLE_HOME}; export ORACLE_HOME
ORACLE_SID=${ORACLE_SID}; export ORACLE_SID
PROMSGS=$DLC/promsgs;export PROMSGS
PROTERMCAP=$DLC/protermcap;export PROTERMCAP
PS1='$$ ';export PS1
PROPATH=${PROPATH:-&SCRIPT_PROPATH};export PROPATH

NLS_NUMERIC_CHARACTERS=".,"; export
NLS_NUMERIC_CHARACTERS
NLS_LANG=&NLS_LANG;export NLS_LANG

#
# Set terminal type.
#
if [ ${TERM:-NULL} = NULL ]
then
    echo
    echo "Please enter your terminal type: \c"
```

```
    read TERM
    export TERM
fi

#
# Start MFG/PRO.
#
cd  # change to home directory

# exec $DLC/bin/_progres &DB etc
&CLIENT-DB-CONNECT
```

## guitrunc

```
TRUNCATE TABLE BRWF_DET;
TRUNCATE TABLE BRWT_DET;
TRUNCATE TABLE BRW_MSTR;
TRUNCATE TABLE DRL_MSTR;
TRUNCATE TABLE FLC_MSTR;
TRUNCATE TABLE GRFD_DET;
TRUNCATE TABLE MNDS_DET;
TRUNCATE TABLE MNTS_DET;
TRUNCATE TABLE OBCD_DET;
TRUNCATE TABLE OBCL_DET;
TRUNCATE TABLE OBCV_DET;
TRUNCATE TABLE OBC_MSTR;
TRUNCATE TABLE PWCD_DET;
TRUNCATE TABLE PWC_MSTR;
TRUNCATE TABLE TBRD_DET;
TRUNCATE TABLE TBR_MSTR;
TRUNCATE TABLE UIP_MSTR;
TRUNCATE TABLE USS_MSTR;
TRUNCATE TABLE UTD_DET;
TRUNCATE TABLE VUE_MSTR;
TRUNCATE TABLE VUF_DET;
TRUNCATE TABLE VWJ_DET;
```

## ora085up.sql

```
spool ora85up.log

ALTER TABLE ad_mstr MODIFY (
       ad_timezone varchar2 (80));

ALTER TABLE tr_hist MODIFY (
       tr_rmks varchar2 (80));

ALTER TABLE tzod_det MODIFY (
       U##tzod_tzone varchar2 (80),
       tzod_tzone varchar2 (80));

ALTER SEQUENCE op_sq01 MAXVALUE 99999999;

ALTER TABLE mnts_det MODIFY (
       mnts_label varchar2 (255));

spool off
```

## ofcempty.sql

```
spool ocfempty.lst

DROP SEQUENCE brd_det_SEQ;
CREATE SEQUENCE brd_det_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;
DROP TABLE brd_det;
CREATE TABLE brd_det(
      U##brd_name varchar2 (80) NULL,
      brd_name varchar2 (80) NULL,
      brd_desc varchar2 (80) NULL,
      U##brd_trig_table varchar2 (80) NULL,
      brd_trig_table varchar2 (80) NULL,
      U##brd_trig_field varchar2 (80) NULL,
      brd_trig_field varchar2 (80) NULL,
      brd_nbr number NULL,
      U##brd_val_table varchar2 (80) NULL,
      brd_val_table varchar2 (80) NULL,
      U##brd_val_field varchar2 (80) NULL,
      brd_val_field varchar2 (80) NULL,
      brd_msg_nbr number NULL,
      brd_repl_var1 varchar2 (80) NULL,
      brd_repl_var2 varchar2 (80) NULL,
      brd_repl_var3 varchar2 (80) NULL,
      brd_concat varchar2 (80) NULL,
      brd_commit_only number NULL,
      brd_dialog_button number NULL,
      brd_key_field number NULL,
      brd_userid varchar2 (80) NULL,
      brd_mod_date date NULL,
      brd_modified number NULL,
      brd__qadc01 varchar2 (80) NULL,
```

```
        brd__qadc02 varchar2 (80) NULL,
        brd__qadc03 varchar2 (80) NULL,
        brd__qadc05 varchar2 (80) NULL,
        brd__qadi01 number NULL,
        brd__qadi02 number NULL,
        brd__chr01 varchar2 (80) NULL,
        brd__chr02 varchar2 (80) NULL,
        brd__chr03 varchar2 (80) NULL,
        brd__chr04 varchar2 (80) NULL,
        brd__chr05 varchar2 (80) NULL,
        brd__qadc04 varchar2 (80) NULL,
        brd__dte01 date,
        brd__dte02 date,
        brd__log01 number NULL,
        brd__log02 number NULL,
        brd_no_blank number NULL,
        brd_gen_code_val number NULL,
        brd_pswd_protect number NULL,
        brd_lng_val number NULL,
        brd_com_sev number NULL,
        brd_leave_sev number NULL,
        brd_lngd_dataset varchar2 (30) NULL,
        brd_gen_code_fld varchar2 (80) NULL,
        progress_recid number null
) TABLESPACE compconf;

CREATE UNIQUE INDEX brd_det##progress_recid on brd_det
(progress_recid) TABLESPACE compconf_idx;

CREATE UNIQUE INDEX brd_det##brd_det on brd_det
(U##brd_name, U##brd_trig_table, U##brd_trig_field,
brd_nbr) TABLESPACE compconf_idx;

CREATE INDEX brd_det##brd_field on brd_det (U##brd_name,
U##brd_val_table, U##brd_val_field, brd_nbr,
progress_recid) TABLESPACE compconf_idx;

DROP SEQUENCE br_mst_SEQ;
```

```
CREATE SEQUENCE br_mst_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;
DROP TABLE br_mst;
CREATE TABLE br_mst(
      U##br_name varchar2 (80) NULL,
      br_name varchar2 (80) NULL,
      br_desc varchar2 (80) NULL,
      br_db varchar2 (80) NULL,
      br_table varchar2 (80) NULL,
      br_userid varchar2 (80) NULL,
      br_mod_date date NULL,
      br_modified number NULL,
      br__qadc01 varchar2 (80) NULL,
      br__qadc02 varchar2 (80) NULL,
      br__qadc03 varchar2 (80) NULL,
      br__qadc04 varchar2 (80) NULL,
      br__qadc05 varchar2 (80) NULL,
      br__qadi01 number NULL,
      br__qadi02 number NULL,
      br__chr01 varchar2 (80) NULL,
      br__chr02 varchar2 (80) NULL,
      br__chr03 varchar2 (80) NULL,
      br__chr04 varchar2 (80) NULL,
      br__chr05 varchar2 (80) NULL,
      br__dte01 date,
      br__dte02 date,
      br__log01 number NULL,
      br__log02 number NULL,
      br_config number NULL,
      progress_recid number null
) TABLESPACE compconf;
CREATE UNIQUE INDEX br_mst##progress_recid on br_mst
(progress_recid) TABLESPACE compconf_idx;
```

```
CREATE UNIQUE INDEX br_mst##br_mst on br_mst (U##br_name)
TABLESPACE compconf_idx;
DROP SEQUENCE con_mst_SEQ;
CREATE SEQUENCE con_mst_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;
DROP TABLE con_mst;
CREATE TABLE con_mst(
      U##con_name varchar2 (80) NULL,
      con_name varchar2 (80) NULL,
      con_desc varchar2 (80) NULL,
      con_super number NULL,
      con_label varchar2 (80) NULL,
      con_table varchar2 (80) NULL,
      con_userid varchar2 (80) NULL,
      con_mod_date date NULL,
      con_modified number NULL,
      con__qadc01 varchar2 (80) NULL,
      con__qadc02 varchar2 (80) NULL,
      con__qadc03 varchar2 (80) NULL,
      con__qadc04 varchar2 (80) NULL,
      con__qadc05 varchar2 (80) NULL,
      con__qadi01 number NULL,
      con__qadi02 number NULL,
      con__qadl01 number NULL,
      con__qadl02 number NULL,
      con__chr01 varchar2 (80) NULL,
      con__chr02 varchar2 (80) NULL,
      con__chr03 varchar2 (80) NULL,
      con__chr04 varchar2 (80) NULL,
      con__chr05 varchar2 (80) NULL,
      con__dte01 date,
      con__log02 number NULL,
      con__log01 number NULL,
```

```
      con__dte02 date,
      con_config number NULL,
      progress_recid number null
) TABLESPACE compconf;
CREATE UNIQUE INDEX con_mst##progress_recid on con_mst
(progress_recid) TABLESPACE compconf_idx;
CREATE UNIQUE INDEX con_mst##con_mst on con_mst
(U##con_name) TABLESPACE compconf_idx;
DROP SEQUENCE flc_det_SEQ;
CREATE SEQUENCE flc_det_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;
DROP TABLE flc_det;
CREATE TABLE flc_det(
      U##flc_folder varchar2 (80) NULL,
      flc_folder varchar2 (80) NULL,
      U##flc_table varchar2 (30) NULL,
      flc_table varchar2 (30) NULL,
      U##flc_name varchar2 (30) NULL,
      flc_name varchar2 (30) NULL,
      flc_view_as varchar2 (30) NULL,
      flc_format varchar2 (80) NULL,
      flc_label varchar2 (80) NULL,
      flc_row number NULL,
      flc_column number NULL,
      flc_width number NULL,
      flc_height number NULL,
      flc_access varchar2 (30) NULL,
      flc_tab_order number NULL,
      flc_list varchar2 (255) NULL,
      flc_radio_horiz number NULL,
      flc_sorted_items number NULL,
      flc_multi_select number NULL,
      flc_v_scrlbar number NULL,
```

```
      flc_h_scrlbar number NULL,
      flc_return_ins number NULL,
      flc_no_labels number NULL,
      flc_userid varchar2 (80) NULL,
      flc_mod_date date NULL,
      flc_modified number NULL,
      flc__qadc02 varchar2 (80) NULL,
      flc__qadc03 varchar2 (80) NULL,
      flc__qadc04 varchar2 (80) NULL,
      flc__qadc05 varchar2 (80) NULL,
      flc__qadi01 number NULL,
      flc__qadi02 number NULL,
      flc__chr01 varchar2 (80) NULL,
      flc__chr02 varchar2 (80) NULL,
      flc__qadc01 varchar2 (80) NULL,
      flc__chr03 varchar2 (80) NULL,
      flc__chr04 varchar2 (80) NULL,
      flc__chr05 varchar2 (80) NULL,
      flc__dte01 date,
      flc__dte02 date,
      flc__log01 number NULL,
      flc__log02 number NULL,
      flc_std_bts varchar2 (255) NULL,
      flc_std_trigs varchar2 (255) NULL,
      flc_cstm_bts varchar2 (255) NULL,
      flc_cstm_trigs varchar2 (255) NULL,
      flc_rect_flds varchar2 (255) NULL,
      flc_dest_ctrl varchar2 (30) NULL,
      flc_dest_label varchar2 (80) NULL,
      progress_recid number null
) TABLESPACE compconf;
CREATE UNIQUE INDEX flc_det##progress_recid on flc_det
(progress_recid) TABLESPACE compconf_idx;
```

```
CREATE UNIQUE INDEX flc_det##flc_index on flc_det
(U##flc_folder, U##flc_table, U##flc_name) TABLESPACE
compconf_idx;
DROP SEQUENCE fldr_mst_SEQ;
CREATE SEQUENCE fldr_mst_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;
DROP TABLE fldr_mst;
CREATE TABLE fldr_mst(
      U##fldr_name varchar2 (80) NULL,
      fldr_name varchar2 (80) NULL,
      fldr_desc varchar2 (80) NULL,
      fldr_label varchar2 (80) NULL,
      fldr_title varchar2 (80) NULL,
      fldr_table varchar2 (30) NULL,
      fldr_userid varchar2 (80) NULL,
      fldr_mod_date date NULL,
      fldr_modified number NULL,
      fldr__qadc01 varchar2 (80) NULL,
      fldr__qadc02 varchar2 (80) NULL,
      fldr__qadc03 varchar2 (80) NULL,
      fldr__qadc04 varchar2 (80) NULL,
      fldr__qadc05 varchar2 (80) NULL,
      fldr__qadi01 number NULL,
      fldr__qadi02 number NULL,
      fldr__chr01 varchar2 (80) NULL,
      fldr__chr02 varchar2 (80) NULL,
      fldr__chr03 varchar2 (80) NULL,
      fldr__chr04 varchar2 (80) NULL,
      fldr__chr05 varchar2 (80) NULL,
      fldr__dte01 date,
      fldr__dte02 date,
      fldr__log02 number NULL,
      fldr__log01 number NULL,
```

```
      fldr_config number NULL,
      progress_recid number null
) TABLESPACE compconf;
CREATE UNIQUE INDEX fldr_mst##progress_recid on fldr_mst
(progress_recid) TABLESPACE compconf_idx;
CREATE UNIQUE INDEX fldr_mst##fldr_mst on fldr_mst
(U##fldr_name) TABLESPACE compconf_idx;
DROP SEQUENCE grp_det_SEQ;
CREATE SEQUENCE grp_det_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;
DROP TABLE grp_det;
CREATE TABLE grp_det(
      U##grp_con_name varchar2 (80) NULL,
      grp_con_name varchar2 (80) NULL,
      grp_nbr number NULL,
      grp_kf_name varchar2 (80) NULL,
      grp_nav_name varchar2 (80) NULL,
      grp_pbrw_name varchar2 (80) NULL,
      U##grp_br_name varchar2 (80) NULL,
      grp_br_name varchar2 (80) NULL,
      grp_create number NULL,
      grp_update number NULL,
      grp_delete number NULL,
      grp_folders varchar2 (80) NULL,
      grp_sub_name varchar2 (80) NULL,
      grp_sub_label varchar2 (80) NULL,
      grp_userid varchar2 (80) NULL,
      grp_mod_date date NULL,
      grp_modified number NULL,
      grp__qadc01 varchar2 (80) NULL,
      grp__qadc02 varchar2 (80) NULL,
      grp__qadc03 varchar2 (80) NULL,
      grp__qadc04 varchar2 (80) NULL,
```

```
        grp__qadc05 varchar2 (80) NULL,
        grp__qadi01 number NULL,
        grp__qadi02 number NULL,
        grp__chr01 varchar2 (80) NULL,
        grp__chr03 varchar2 (80) NULL,
        grp__chr04 varchar2 (80) NULL,
        grp__chr05 varchar2 (80) NULL,
        grp__dte01 date,
        grp__dte02 date,
        grp__log01 number NULL,
        grp__log02 number NULL,
        grp__chr02 varchar2 (80) NULL,
        progress_recid number null
) TABLESPACE compconf;
CREATE UNIQUE INDEX grp_det##progress_recid on grp_det
(progress_recid) TABLESPACE compconf_idx;
CREATE INDEX grp_det##grp_br_name on grp_det
(U##grp_br_name, grp_nbr, progress_recid) TABLESPACE
compconf_idx;
CREATE UNIQUE INDEX grp_det##grp_det on grp_det
(U##grp_con_name, grp_nbr) TABLESPACE compconf_idx;
DROP SEQUENCE lnk_det_SEQ;
CREATE SEQUENCE lnk_det_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;
DROP TABLE lnk_det;
CREATE TABLE lnk_det(
      U##lnk_con_name varchar2 (80) NULL,
      lnk_con_name varchar2 (80) NULL,
      U##lnk_to_con varchar2 (80) NULL,
      lnk_to_con varchar2 (80) NULL,
      lnk_to_group number NULL,
      U##lnk_to_br_name varchar2 (80) NULL,
      lnk_to_br_name varchar2 (80) NULL,
```

```
      U##lnk_to_table varchar2 (30) NULL,
      lnk_to_table varchar2 (30) NULL,
      U##lnk_to_field varchar2 (80) NULL,
      lnk_to_field varchar2 (80) NULL,
      lnk_from_con varchar2 (80) NULL,
      lnk_from_group number NULL,
      lnk_from_br_name varchar2 (80) NULL,
      lnk_from_table varchar2 (30) NULL,
      lnk_from_field varchar2 (80) NULL,
      lnk_value varchar2 (80) NULL,
      lnk_disable number NULL,
      lnk_userid varchar2 (80) NULL,
      lnk_mod_date date NULL,
      lnk_modified number NULL,
      lnk__qadc01 varchar2 (80) NULL,
      lnk__qadc02 varchar2 (80) NULL,
      lnk__qadc03 varchar2 (80) NULL,
      lnk__qadc04 varchar2 (80) NULL,
      lnk__qadc05 varchar2 (80) NULL,
      lnk__qadi01 number NULL,
      lnk__qadi02 number NULL,
      lnk__chr01 varchar2 (80) NULL,
      lnk__chr02 varchar2 (80) NULL,
      lnk__chr03 varchar2 (80) NULL,
      lnk__chr04 varchar2 (80) NULL,
      lnk__dte01 date,
      lnk__dte02 date,
      lnk__log01 number NULL,
      lnk__log02 number NULL,
      lnk__chr05 varchar2 (80) NULL,
      progress_recid number null
) TABLESPACE compconf;
CREATE UNIQUE INDEX lnk_det##progress_recid on lnk_det
```

```
(progress_recid) TABLESPACE compconf_idx;
CREATE UNIQUE INDEX lnk_det##lnk_det on lnk_det
(U##lnk_con_name, U##lnk_to_con, lnk_to_group,
U##lnk_to_br_name, U##lnk_to_table, U##lnk_to_field)
TABLESPACE compconf_idx;
DROP SEQUENCE nav_mst_SEQ;
CREATE SEQUENCE nav_mst_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;
DROP TABLE nav_mst;
CREATE TABLE nav_mst(
      U##nav_name varchar2 (80) NULL,
      nav_name varchar2 (80) NULL,
      nav_desc varchar2 (80) NULL,
      nav_table varchar2 (80) NULL,
      nav_index varchar2 (80) NULL,
      nav_field1 varchar2 (80) NULL,
      nav_field2 varchar2 (80) NULL,
      nav_field3 varchar2 (80) NULL,
      nav_field4 varchar2 (80) NULL,
      nav_field5 varchar2 (80) NULL,
      nav_comparand1 varchar2 (80) NULL,
      nav_comparand2 varchar2 (80) NULL,
      nav_comparand3 varchar2 (80) NULL,
      nav_comparand4 varchar2 (80) NULL,
      nav_comparand5 varchar2 (80) NULL,
      nav_where varchar2 (255) NULL,
      nav_userid varchar2 (80) NULL,
      nav_mod_date date NULL,
      nav_modified number NULL,
      nav__qadc01 varchar2 (80) NULL,
      nav__qadc02 varchar2 (80) NULL,
      nav__qadc03 varchar2 (80) NULL,
      nav__qadc04 varchar2 (80) NULL,
```

```
      nav__qadc05 varchar2 (80) NULL,
      nav__qadi02 number NULL,
      nav__chr01 varchar2 (80) NULL,
      nav__chr02 varchar2 (80) NULL,
      nav__chr03 varchar2 (80) NULL,
      nav__chr04 varchar2 (80) NULL,
      nav__chr05 varchar2 (80) NULL,
      nav__dte01 date,
      nav__dte02 date,
      nav__qadi01 number NULL,
      nav__log01 number NULL,
      nav__log02 number NULL,
      nav_config number NULL,
      progress_recid number null
) TABLESPACE compconf;
CREATE UNIQUE INDEX nav_mst##progress_recid on nav_mst
(progress_recid) TABLESPACE compconf_idx;
CREATE UNIQUE INDEX nav_mst##nav_mst on nav_mst
(U##nav_name) TABLESPACE compconf_idx;
DROP SEQUENCE proc_det_SEQ;
CREATE SEQUENCE proc_det_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;
DROP TABLE proc_det;
CREATE TABLE proc_det(
      U##proc_name varchar2 (30) NULL,
      proc_name varchar2 (30) NULL,
      U##proc_proc_name varchar2 (80) NULL,
      proc_proc_name varchar2 (80) NULL,
      proc_userid varchar2 (80) NULL,
      proc_mod_date date NULL,
      proc_modified number NULL,
      proc__qadc01 varchar2 (80) NULL,
      proc__qadc02 varchar2 (80) NULL,
```

```
        proc__qadc03 varchar2 (80) NULL,
        proc__qadc04 varchar2 (80) NULL,
        proc__qadc05 varchar2 (80) NULL,
        proc__qadi01 number NULL,
        proc__qadi02 number NULL,
        proc__chr01 varchar2 (80) NULL,
        proc__chr02 varchar2 (80) NULL,
        proc__chr03 varchar2 (80) NULL,
        proc__chr04 varchar2 (80) NULL,
        proc__chr05 varchar2 (80) NULL,
        proc__dte01 date,
        proc__dte02 date,
        proc__log01 number NULL,
        proc__log02 number NULL,
        progress_recid number null
) TABLESPACE compconf;
CREATE UNIQUE INDEX proc_det##progress_recid on proc_det
(progress_recid) TABLESPACE compconf_idx;
CREATE UNIQUE INDEX proc_det##proc_det on proc_det
(U##proc_name, U##proc_proc_name) TABLESPACE
compconf_idx;
DROP SEQUENCE prol_det_SEQ;
CREATE SEQUENCE prol_det_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;
DROP TABLE prol_det;
CREATE TABLE prol_det(
        U##prol_name varchar2 (30) NULL,
        prol_name varchar2 (30) NULL,
        U##prol_proc_name varchar2 (80) NULL,
        prol_proc_name varchar2 (80) NULL,
        prol_line_nbr number NULL,
        prol_text varchar2 (255) NULL,
        prol__qadc01 varchar2 (80) NULL,
```

```
      prol__qadc02 varchar2 (80) NULL,
      prol__qadc03 varchar2 (80) NULL,
      prol__qadc04 varchar2 (80) NULL,
      prol__qadc05 varchar2 (80) NULL,
      prol__qadi01 number NULL,
      prol__qadi02 number NULL,
      prol__chr01 varchar2 (80) NULL,
      prol__chr02 varchar2 (80) NULL,
      prol__chr03 varchar2 (80) NULL,
      prol__chr04 varchar2 (80) NULL,
      prol__chr05 varchar2 (80) NULL,
      prol__dte01 date,
      prol__dte02 date,
      prol__log01 number NULL,
      prol__log02 number NULL,
      progress_recid number null
) TABLESPACE compconf;
CREATE UNIQUE INDEX prol_det##progress_recid on prol_det
(progress_recid) TABLESPACE compconf_idx;
CREATE UNIQUE INDEX prol_det##prol_det on prol_det
(U##prol_name, U##prol_proc_name, prol_line_nbr)
TABLESPACE compconf_idx;
DROP SEQUENCE var_det_SEQ;
CREATE SEQUENCE var_det_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;
DROP TABLE var_det;
CREATE TABLE var_det(
      U##var_component varchar2 (80) NULL,
      var_component varchar2 (80) NULL,
      U##var_usage varchar2 (8) NULL,
      var_usage varchar2 (8) NULL,
      U##var_name varchar2 (30) NULL,
      var_name varchar2 (30) NULL,
```

```
        var_type varchar2 (30) NULL,
        var_decimals number NULL,
        var_extents number NULL,
        var_initial varchar2 (80) NULL,
        var_no_undo number NULL,
        var_format varchar2 (80) NULL,
        var_userid varchar2 (80) NULL,
        var_mod_date date NULL,
        var_modified number NULL,
        var__qadc01 varchar2 (80) NULL,
        var__qadc02 varchar2 (80) NULL,
        var__qadc03 varchar2 (80) NULL,
        var__qadc04 varchar2 (80) NULL,
        var__qadc05 varchar2 (80) NULL,
        var__qadi01 number NULL,
        var__qadi02 number NULL,
        var__chr01 varchar2 (80) NULL,
        var__chr02 varchar2 (80) NULL,
        var__chr03 varchar2 (80) NULL,
        var__chr04 varchar2 (80) NULL,
        var__dte01 date,
        var__dte02 date,
        var__log01 number NULL,
        var__log02 number NULL,
        var__chr05 varchar2 (80) NULL,
        progress_recid number null
) TABLESPACE compconf;

CREATE UNIQUE INDEX var_det##progress_recid on var_det
(progress_recid) TABLESPACE compconf_idx;

CREATE UNIQUE INDEX var_det##var_index on var_det
(U##var_component, U##var_usage, U##var_name) TABLESPACE
compconf_idx;


spool off
```

## ogui.sql

```
spool ogui.lst


DROP SEQUENCE brwf_det_SEQ;
CREATE SEQUENCE brwf_det_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;
DROP TABLE brwf_det;
CREATE TABLE brwf_det(
      U##brw_name varchar2 (30) NULL,
      brw_name varchar2 (30) NULL,
      brwf_seq number NULL,
      U##brwf_field varchar2 (30) NULL,
      brwf_field varchar2 (30) NULL,
      brwf_datatype varchar2 (30) NULL,
      brwf_format varchar2 (30) NULL,
      brwf_label varchar2 (80),
      brwf_col_label varchar2 (80),
      brwf_expression varchar2 (80),
      brwf_table varchar2 (30) NULL,
      brwf_select number NULL,
      brwf_sort number NULL,
      brwf_userid varchar2 (80),
      brwf_mod_date date,
      brwf_user1 varchar2 (80),
      brwf_user2 varchar2 (80),
      brwf__qad01 varchar2 (80),
      brwf__qad02 varchar2 (80),
      progress_recid number null
) TABLESPACE gui;
CREATE UNIQUE INDEX brwf_det##progress_recid on brwf_det
(progress_recid) TABLESPACE gui_idx;
CREATE UNIQUE INDEX brwf_det##brwf_det on brwf_det
(U##brw_name, brwf_seq) TABLESPACE gui_idx;
```

```
CREATE INDEX brwf_det##brwf_field on brwf_det
(U##brwf_field, progress_recid) TABLESPACE gui_idx;

DROP SEQUENCE brwt_det_SEQ;

CREATE SEQUENCE brwt_det_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;

DROP TABLE brwt_det;

CREATE TABLE brwt_det(

      U##brw_name varchar2 (30) NULL,

      brw_name varchar2 (30) NULL,

      brwt_seq number NULL,

      U##brwt_table varchar2 (30) NULL,

      brwt_table varchar2 (30) NULL,

      brwt_join varchar2 (255),

      brwt_where varchar2 (255),

      brwt_userid varchar2 (80),

      brwt_mod_date date,

      brwt_user1 varchar2 (80),

      brwt_user2 varchar2 (80),

      brwt__qad01 varchar2 (80),

      brwt__qad02 varchar2 (80),

      progress_recid number null

) TABLESPACE gui;

CREATE UNIQUE INDEX brwt_det##progress_recid on brwt_det
(progress_recid) TABLESPACE gui_idx;

CREATE UNIQUE INDEX brwt_det##brwt_det on brwt_det
(U##brw_name, brwt_seq) TABLESPACE gui_idx;

CREATE INDEX brwt_det##brwt_table on brwt_det
(U##brwt_table, progress_recid) TABLESPACE gui_idx;

DROP SEQUENCE brw_mstr_SEQ;

CREATE SEQUENCE brw_mstr_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;

DROP TABLE brw_mstr;

CREATE TABLE brw_mstr(

      U##brw_name varchar2 (30) NULL,

      brw_name varchar2 (30) NULL,
```

```
      U##brw_desc varchar2 (80),
      brw_desc varchar2 (80),
      U##brw_view varchar2 (30) NULL,
      brw_view varchar2 (30) NULL,
      brw_cansee varchar2 (80),
      brw_filter varchar2 (255),
      brw_userid varchar2 (80),
      brw_mod_date date,
      brw_user1 varchar2 (80),
      brw_user2 varchar2 (80),
      brw__qad01 varchar2 (80),
      brw__qad02 varchar2 (80),
      brw_pwr_brw number,
      brw_lu_brw number,
      progress_recid number null
) TABLESPACE gui;

CREATE UNIQUE INDEX brw_mstr##progress_recid on brw_mstr
(progress_recid) TABLESPACE gui_idx;

CREATE UNIQUE INDEX brw_mstr##brw_desc on brw_mstr
(U##brw_desc, U##brw_name) TABLESPACE gui_idx;

CREATE UNIQUE INDEX brw_mstr##brw_mstr on brw_mstr
(U##brw_name) TABLESPACE gui_idx;

CREATE UNIQUE INDEX brw_mstr##brw_view on brw_mstr
(U##brw_view, U##brw_name) TABLESPACE gui_idx;

DROP SEQUENCE drl_mstr_SEQ;

CREATE SEQUENCE drl_mstr_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;

DROP TABLE drl_mstr;

CREATE TABLE drl_mstr(
      U##drl_field varchar2 (30) NULL,
      drl_field varchar2 (30) NULL,
      U##drl_call_pgm varchar2 (30) NULL,
      drl_call_pgm varchar2 (30) NULL,
      U##drl_exec varchar2 (30) NULL,
```

```
      drl_exec varchar2 (30) NULL,
      drl_desc varchar2 (80) NULL,
      drl_userid varchar2 (80),
      drl_mod_date date,
      drl_user1 varchar2 (80),
      drl_user2 varchar2 (80),
      drl__qad01 varchar2 (80),
      drl__qad02 varchar2 (80),
      progress_recid number null
) TABLESPACE gui;

CREATE UNIQUE INDEX drl_mstr##progress_recid on drl_mstr
(progress_recid) TABLESPACE gui_idx;

CREATE INDEX drl_mstr##drl_call_pgm on drl_mstr
(U##drl_call_pgm, U##drl_exec, U##drl_field,
progress_recid) TABLESPACE gui_idx;

CREATE INDEX drl_mstr##drl_exec on drl_mstr (U##drl_exec,
U##drl_call_pgm, U##drl_field, progress_recid) TABLESPACE
gui_idx;

CREATE UNIQUE INDEX drl_mstr##drl_field on drl_mstr
(U##drl_field, U##drl_call_pgm, U##drl_exec) TABLESPACE
gui_idx;

DROP SEQUENCE flc_mstr_SEQ;

CREATE SEQUENCE flc_mstr_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;

DROP TABLE flc_mstr;

CREATE TABLE flc_mstr(
      U##flc_field varchar2 (30),
      flc_field varchar2 (30),
      flc_desc varchar2 (80),
      flc_exec varchar2 (30),
      flc_down number,
      U##flc_call_pgm varchar2 (30),
      flc_call_pgm varchar2 (30),
      flc_user1 varchar2 (80),
      flc_user2 varchar2 (80),
      flc_userid varchar2 (80),
```

```
      flc_mod_date date,
      flc__qad01 varchar2 (80),
      flc__qad02 varchar2 (80),
      progress_recid number null
) TABLESPACE gui;
CREATE UNIQUE INDEX flc_mstr##progress_recid on flc_mstr
(progress_recid) TABLESPACE gui_idx;
CREATE UNIQUE INDEX flc_mstr##flc_mstr on flc_mstr
(U##flc_call_pgm, U##flc_field) TABLESPACE gui_idx;
DROP SEQUENCE grfd_det_SEQ;
CREATE SEQUENCE grfd_det_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;
DROP TABLE grfd_det;
CREATE TABLE grfd_det(
      U##grfd_userid varchar2 (80),
      grfd_userid varchar2 (80),
      grfd_seq number NULL,
      grfd_xlabel varchar2 (80),
      grfd_occur number,
      grfd_cnt##1 number,
      grfd_cnt##2 number,
      grfd_cnt##3 number,
      grfd_cnt##4 number,
      grfd_cnt##5 number,
      grfd_raw##1 number (26,10),
      grfd_raw##2 number (26,10),
      grfd_raw##3 number (26,10),
      grfd_raw##4 number (26,10),
      grfd_raw##5 number (26,10),
      grfd_tot##1 number (30,10),
      grfd_tot##2 number (30,10),
      grfd_tot##3 number (30,10),
      grfd_tot##4 number (30,10),
```

```
        grfd_tot##5 number (30,10),
        grfd_min##1 number (30,10),
        grfd_min##2 number (30,10),
        grfd_min##3 number (30,10),
        grfd_min##4 number (30,10),
        grfd_min##5 number (30,10),
        grfd_max##1 number (30,10),
        grfd_max##2 number (30,10),
        grfd_max##3 number (30,10),
        grfd_max##4 number (30,10),
        grfd_max##5 number (30,10),
        grfd_user1 varchar2 (80),
        grfd_user2 varchar2 (80),
        grfd_mod_date date,
        grfd__qad01 varchar2 (80),
        grfd__qad02 varchar2 (80),
        progress_recid number null
) TABLESPACE gui;

CREATE UNIQUE INDEX grfd_det##progress_recid on grfd_det
(progress_recid) TABLESPACE gui_idx;

CREATE UNIQUE INDEX grfd_det##grfd_usrseq on grfd_det
(U##grfd_userid, grfd_seq) TABLESPACE gui_idx;

DROP SEQUENCE mnds_det_SEQ;

CREATE SEQUENCE mnds_det_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;

DROP TABLE mnds_det;

CREATE TABLE mnds_det(
        U##mnds_exec varchar2 (30),
        mnds_exec varchar2 (30),
        U##mnds_exec_sub varchar2 (30),
        mnds_exec_sub varchar2 (30),
        mnds_label varchar2 (80),
        mnds_userid varchar2 (80),
        mnds_mod_date date,
```

```
      mnds_user1 varchar2 (80),
      mnds_user2 varchar2 (80),
      mnds__qad01 varchar2 (80),
      mnds__qad02 varchar2 (80),
      progress_recid number null
) TABLESPACE gui;
CREATE UNIQUE INDEX mnds_det##progress_recid on mnds_det
(progress_recid) TABLESPACE gui_idx;
CREATE UNIQUE INDEX mnds_det##mnds_exec on mnds_det
(U##mnds_exec) TABLESPACE gui_idx;
CREATE INDEX mnds_det##mnds_exec_sub on mnds_det
(U##mnds_exec_sub, progress_recid) TABLESPACE gui_idx;
DROP SEQUENCE mnts_det_SEQ;
CREATE SEQUENCE mnts_det_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;
DROP TABLE mnts_det;
CREATE TABLE mnts_det(
      U##mnts_exec varchar2 (80),
      mnts_exec varchar2 (80),
      U##mnts_exec_sub varchar2 (30),
      mnts_exec_sub varchar2 (30),
      mnts_label varchar2 (255),
      U##mnts_lang varchar2 (8),
      mnts_lang varchar2 (8),
      mnts_userid varchar2 (80),
      mnts_mod_date date,
      mnts_user1 varchar2 (80),
      mnts_user2 varchar2 (80),
      mnts__qad01 varchar2 (80),
      mnts__qad02 varchar2 (80),
      progress_recid number null
) TABLESPACE gui;
CREATE UNIQUE INDEX mnts_det##progress_recid on mnts_det
```

```
(progress_recid) TABLESPACE gui_idx;

CREATE UNIQUE INDEX mnts_det##mnts_exec on mnts_det
(U##mnts_exec, U##mnts_lang) TABLESPACE gui_idx;

CREATE INDEX mnts_det##mnts_exec_sub on mnts_det
(U##mnts_exec_sub, U##mnts_lang, progress_recid)
TABLESPACE gui_idx;

DROP SEQUENCE obcd_det_SEQ;

CREATE SEQUENCE obcd_det_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;

DROP TABLE obcd_det;

CREATE TABLE obcd_det(
      U##obcd_cntr_id varchar2 (30) NULL,
      obcd_cntr_id varchar2 (30) NULL,
      U##obcd_frm_grp varchar2 (30) NULL,
      obcd_frm_grp varchar2 (30) NULL,
      U##obcd_userid varchar2 (30) NULL,
      obcd_userid varchar2 (30) NULL,
      U##obcd_frm_id varchar2 (30) NULL,
      obcd_frm_id varchar2 (30) NULL,
      U##obcd_lang varchar2 (8),
      obcd_lang varchar2 (8),
      obcd_frm_title varchar2 (80),
      obcd_frm_label varchar2 (80),
      obcd_frm_secure number NULL,
      obcd_frm_type number NULL,
      obcd_frm_order number NULL,
      obcd_fld_none varchar2 (255),
      obcd_fld_read varchar2 (255),
      obcd_default number,
      obcd_mod_date date,
      obcd_mod_user varchar2 (80),
      obcd_user1 varchar2 (80),
      obcd_user2 varchar2 (80),
      obcd__qad01 varchar2 (80),
```

```
      obcd__qad02 varchar2 (80),
      progress_recid number null
) TABLESPACE gui;
CREATE UNIQUE INDEX obcd_det##progress_recid on obcd_det
(progress_recid) TABLESPACE gui_idx;
CREATE UNIQUE INDEX obcd_det##obcd_det on obcd_det
(U##obcd_cntr_id, U##obcd_frm_grp, U##obcd_userid,
U##obcd_frm_id, U##obcd_lang) TABLESPACE gui_idx;
CREATE UNIQUE INDEX obcd_det##obcd_userid on obcd_det
(U##obcd_userid, U##obcd_cntr_id, U##obcd_frm_grp,
obcd_frm_order, U##obcd_frm_id, U##obcd_lang) TABLESPACE
gui_idx;
DROP SEQUENCE obcl_det_SEQ;
CREATE SEQUENCE obcl_det_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;
DROP TABLE obcl_det;
CREATE TABLE obcl_det(
      U##obcl_cntr_id varchar2 (30) NULL,
      obcl_cntr_id varchar2 (30) NULL,
      U##obcl_userid varchar2 (80),
      obcl_userid varchar2 (80),
      U##obcl_lang varchar2 (8),
      obcl_lang varchar2 (8),
      U##obcl_to_group varchar2 (30),
      obcl_to_group varchar2 (30),
      U##obcl_to_process varchar2 (30),
      obcl_to_process varchar2 (30),
      U##obcl_to_field varchar2 (30),
      obcl_to_field varchar2 (30),
      obcl_fr_group varchar2 (30),
      obcl_fr_process varchar2 (30),
      obcl_fr_field varchar2 (30),
      obcl_value varchar2 (30),
      obcl_disable number,
      obcl_mod_date date,
```

```
        obcl_user1 varchar2 (80),
        obcl_user2 varchar2 (80),
        obcl__qad01 varchar2 (80),
        obcl__qad02 varchar2 (80),
        progress_recid number null
) TABLESPACE gui;

CREATE UNIQUE INDEX obcl_det##progress_recid on obcl_det
(progress_recid) TABLESPACE gui_idx;

CREATE INDEX obcl_det##obcl_det on obcl_det
(U##obcl_cntr_id, U##obcl_userid, U##obcl_lang,
U##obcl_to_group, U##obcl_to_process, U##obcl_to_field,
progress_recid) TABLESPACE gui_idx;

DROP SEQUENCE obcv_det_SEQ;

CREATE SEQUENCE obcv_det_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;

DROP TABLE obcv_det;

CREATE TABLE obcv_det(
        U##obcv_cntr_id varchar2 (30) NULL,
        obcv_cntr_id varchar2 (30) NULL,
        U##obcv_userid varchar2 (80),
        obcv_userid varchar2 (80),
        U##obcv_lang varchar2 (8),
        obcv_lang varchar2 (8),
        U##obcv_to_group varchar2 (30),
        obcv_to_group varchar2 (30),
        U##obcv_to_process varchar2 (30),
        obcv_to_process varchar2 (30),
        U##obcv_to_field varchar2 (30),
        obcv_to_field varchar2 (30),
        obcv_value varchar2 (30),
        obcv_disable number,
        obcv_mod_date date,
        obcv_user1 varchar2 (80),
        obcv_user2 varchar2 (80),
```

```
      obcv__qad01 varchar2 (80),
      obcv__qad02 varchar2 (80),
      progress_recid number null
) TABLESPACE gui;

CREATE UNIQUE INDEX obcv_det##progress_recid on obcv_det
(progress_recid) TABLESPACE gui_idx;

CREATE INDEX obcv_det##obcv_det on obcv_det
(U##obcv_cntr_id, U##obcv_userid, U##obcv_lang,
U##obcv_to_group, U##obcv_to_process, U##obcv_to_field,
progress_recid) TABLESPACE gui_idx;

DROP SEQUENCE obc_mstr_SEQ;

CREATE SEQUENCE obc_mstr_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;

DROP TABLE obc_mstr;

CREATE TABLE obc_mstr(
      U##obc_cntr_id varchar2 (30) NULL,
      obc_cntr_id varchar2 (30) NULL,
      U##obc_userid varchar2 (30),
      obc_userid varchar2 (30),
      U##obc_lang varchar2 (8),
      obc_lang varchar2 (8),
      obc_security number NULL,
      obc_config number NULL,
      obc_fld_labels number,
      obc_default number,
      obc_group varchar2 (30),
      obc_new_window number,
      obc_par_group varchar2 (30),
      obc_tabs number,
      obc_process varchar2 (30),
      obc_process_access varchar2 (30) NULL,
      obc_key_frame varchar2 (30),
      obc_data_frms varchar2 (255),
      obc_recfind varchar2 (30),
```

```
        obc_query_list varchar2 (255),
        obc_method1 varchar2 (30),
        obc_method2 varchar2 (30),
        obc_method3 varchar2 (30),
        obc_method4 varchar2 (30),
        obc_method5 varchar2 (30),
        obc_mod_date date,
        obc_mod_user varchar2 (80),
        obc_user1 varchar2 (80),
        obc_user2 varchar2 (80),
        obc__qad01 varchar2 (80),
        obc__qad02 varchar2 (80),
        obc__qad03 varchar2 (80),
        obc__qad04 varchar2 (80),
        progress_recid number null
) TABLESPACE gui;

CREATE UNIQUE INDEX obc_mstr##progress_recid on obc_mstr
(progress_recid) TABLESPACE gui_idx;

CREATE UNIQUE INDEX obc_mstr##obc_mstr on obc_mstr
(U##obc_cntr_id, U##obc_userid, U##obc_lang) TABLESPACE
gui_idx;

CREATE UNIQUE INDEX obc_mstr##obc_userid on obc_mstr
(U##obc_userid, U##obc_cntr_id, U##obc_lang) TABLESPACE
gui_idx;

DROP SEQUENCE pwcd_det_SEQ;

CREATE SEQUENCE pwcd_det_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;

DROP TABLE pwcd_det;

CREATE TABLE pwcd_det(
        U##pwcd_mnu_id varchar2 (30) NULL,
        pwcd_mnu_id varchar2 (30) NULL,
        U##pwcd_pgm_id varchar2 (30) NULL,
        pwcd_pgm_id varchar2 (30) NULL,
        U##pwcd_userid varchar2 (80) NULL,
        pwcd_userid varchar2 (80) NULL,
```

```
      U##pwcd_frm_id varchar2 (30) NULL,
      pwcd_frm_id varchar2 (30) NULL,
      pwcd_frm_desc varchar2 (80),
      pwcd_frm_title varchar2 (80),
      pwcd_frm_label varchar2 (80),
      pwcd_frm_secure number NULL,
      pwcd_frm_type number NULL,
      pwcd_frm_order number NULL,
      pwcd_fld_none varchar2 (132),
      pwcd_fld_read varchar2 (132),
      U##pwcd_lang varchar2 (8),
      pwcd_lang varchar2 (8),
      pwcd_mod_date date,
      pwcd_user1 varchar2 (80),
      pwcd_user2 varchar2 (80),
      pwcd__qad01 varchar2 (80),
      pwcd__qad02 varchar2 (80),
      progress_recid number null
) TABLESPACE gui;

CREATE UNIQUE INDEX pwcd_det##progress_recid on pwcd_det
(progress_recid) TABLESPACE gui_idx;

CREATE INDEX pwcd_det##pwcd_det on pwcd_det
(U##pwcd_mnu_id, U##pwcd_pgm_id, U##pwcd_userid,
U##pwcd_frm_id, U##pwcd_lang, progress_recid) TABLESPACE
gui_idx;

CREATE INDEX pwcd_det##pwcd_userid on pwcd_det
(U##pwcd_userid, U##pwcd_mnu_id, U##pwcd_pgm_id,
pwcd_frm_order, U##pwcd_frm_id, U##pwcd_lang,
progress_recid) TABLESPACE gui_idx;

DROP SEQUENCE pwc_mstr_SEQ;

CREATE SEQUENCE pwc_mstr_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;

DROP TABLE pwc_mstr;

CREATE TABLE pwc_mstr(
      U##pwc_mnu_id varchar2 (30) NULL,
      pwc_mnu_id varchar2 (30) NULL,
```

```
        U##pwc_pgm_id varchar2 (30) NULL,
        pwc_pgm_id varchar2 (30) NULL,
        U##pwc_userid varchar2 (80),
        pwc_userid varchar2 (80),
        pwc_security number NULL,
        pwc_config number NULL,
        pwc_browse varchar2 (30),
        pwc_fld_labels number,
        U##pwc_lang varchar2 (8),
        pwc_lang varchar2 (8),
        pwc_mod_date date,
        pwc_user1 varchar2 (80),
        pwc_user2 varchar2 (80),
        pwc__qad01 varchar2 (80),
        pwc__qad02 varchar2 (80),
        progress_recid number null
) TABLESPACE gui;

CREATE UNIQUE INDEX pwc_mstr##progress_recid on pwc_mstr
(progress_recid) TABLESPACE gui_idx;

CREATE INDEX pwc_mstr##pwc_mstr on pwc_mstr
(U##pwc_mnu_id, U##pwc_pgm_id, U##pwc_userid,
U##pwc_lang, progress_recid) TABLESPACE gui_idx;

CREATE INDEX pwc_mstr##pwc_userid on pwc_mstr
(U##pwc_userid, U##pwc_mnu_id, U##pwc_pgm_id,
U##pwc_lang, progress_recid) TABLESPACE gui_idx;

DROP SEQUENCE tbrd_det_SEQ;

CREATE SEQUENCE tbrd_det_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;

DROP TABLE tbrd_det;

CREATE TABLE tbrd_det(
        U##tbrd_from_table varchar2 (30) NULL,
        tbrd_from_table varchar2 (30) NULL,
        U##tbrd_to_table varchar2 (30) NULL,
        tbrd_to_table varchar2 (30) NULL,
        tbrd_nbr number NULL,
```

```
      tbrd_reversed number NULL,
      tbrd_userid varchar2 (80),
      tbrd_mod_date date,
      tbrd_user1 varchar2 (80),
      tbrd_user2 varchar2 (80),
      tbrd__qad01 varchar2 (80),
      tbrd__qad02 varchar2 (80),
      progress_recid number null
) TABLESPACE gui;

CREATE UNIQUE INDEX tbrd_det##progress_recid on tbrd_det
(progress_recid) TABLESPACE gui_idx;

CREATE UNIQUE INDEX tbrd_det##tbrd_from_table on tbrd_det
(U##tbrd_from_table, U##tbrd_to_table, tbrd_nbr)
TABLESPACE gui_idx;

DROP SEQUENCE tbr_mstr_SEQ;

CREATE SEQUENCE tbr_mstr_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;

DROP TABLE tbr_mstr;

CREATE TABLE tbr_mstr(
      U##tbr_from_table varchar2 (30) NULL,
      tbr_from_table varchar2 (30) NULL,
      U##tbr_to_table varchar2 (30) NULL,
      tbr_to_table varchar2 (30) NULL,
      tbr_nbr number NULL,
      tbr_desc varchar2 (80),
      tbr_where varchar2 (80) NULL,
      tbr_of number NULL,
      tbr_userid varchar2 (80),
      tbr_mod_date date,
      tbr_user1 varchar2 (80),
      tbr_user2 varchar2 (80),
      tbr__qad01 varchar2 (80),
      tbr__qad02 varchar2 (80),
      tbr_results number,
```

```
      progress_recid number null
) TABLESPACE gui;
CREATE UNIQUE INDEX tbr_mstr##progress_recid on tbr_mstr
(progress_recid) TABLESPACE gui_idx;
CREATE UNIQUE INDEX tbr_mstr##tbr_from_table on tbr_mstr
(U##tbr_from_table, U##tbr_to_table, tbr_nbr) TABLESPACE
gui_idx;
DROP SEQUENCE uip_mstr_SEQ;
CREATE SEQUENCE uip_mstr_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;
DROP TABLE uip_mstr;
CREATE TABLE uip_mstr(
      U##uip_userid varchar2 (80),
      uip_userid varchar2 (80),
      uip_style varchar2 (8),
      uip_sys_tool number NULL,
      uip_usr_tool number NULL,
      uip_nav_win number NULL,
      uip_auto_go number,
      uip_bg_color number,
      uip_hypertext_help number,
      uip_mod_date date,
      uip_user1 varchar2 (80),
      uip_user2 varchar2 (80),
      uip__qad01 varchar2 (80),
      uip__qad02 varchar2 (80),
      progress_recid number null
) TABLESPACE gui;
CREATE UNIQUE INDEX uip_mstr##progress_recid on uip_mstr
(progress_recid) TABLESPACE gui_idx;
CREATE UNIQUE INDEX uip_mstr##uip_userid on uip_mstr
(U##uip_userid) TABLESPACE gui_idx;
DROP SEQUENCE uss_mstr_SEQ;
CREATE SEQUENCE uss_mstr_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;
```

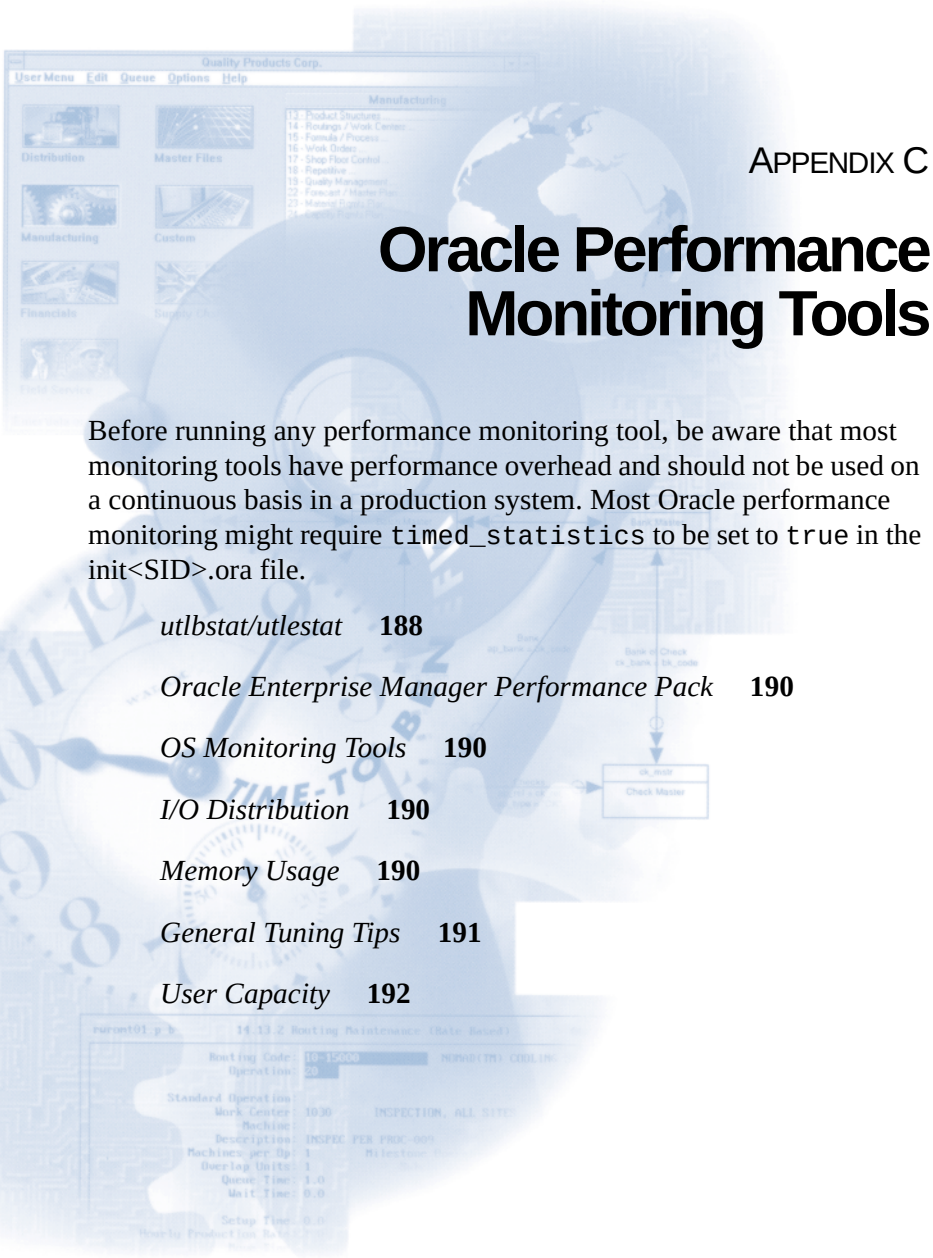
```
DROP TABLE uss_mstr;

CREATE TABLE uss_mstr(
      U##uss_userid varchar2 (80),
      uss_userid varchar2 (80),
      U##uss_style varchar2 (8),
      uss_style varchar2 (8),
      U##uss_menuinfo varchar2 (255),
      uss_menuinfo varchar2 (255),
      uss_logicals number,
      uss_mod_date date,
      uss__qad01 varchar2 (80),
      uss__qad02 varchar2 (80),
      progress_recid number null
) TABLESPACE gui;

CREATE UNIQUE INDEX uss_mstr##progress_recid on uss_mstr
(progress_recid) TABLESPACE gui_idx;

CREATE INDEX uss_mstr##uss_main on uss_mstr
(U##uss_userid, U##uss_style, U##uss_menuinfo,
progress_recid) TABLESPACE gui_idx;

DROP SEQUENCE utd_det_SEQ;

CREATE SEQUENCE utd_det_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;

DROP TABLE utd_det;

CREATE TABLE utd_det(
      U##utd_userid varchar2 (80),
      utd_userid varchar2 (80),
      U##utd_lang varchar2 (8),
      utd_lang varchar2 (8),
      U##utd_program varchar2 (80),
      utd_program varchar2 (80),
      utd_exec1 varchar2 (30),
      utd_exec2 varchar2 (30),
      utd_exec3 varchar2 (30),
      utd_exec4 varchar2 (30),
```

```
        utd_exec5 varchar2 (30),
        utd_exec6 varchar2 (30),
        utd_exec7 varchar2 (30),
        utd_exec8 varchar2 (30),
        utd_pgmlabel1 varchar2 (80),
        utd_pgmlabel2 varchar2 (80),
        utd_pgmlabel3 varchar2 (80),
        utd_pgmlabel4 varchar2 (80),
        utd_pgmlabel5 varchar2 (80),
        utd_pgmlabel6 varchar2 (80),
        utd_pgmlabel7 varchar2 (80),
        utd_pgmlabel8 varchar2 (80),
        utd_image1 varchar2 (80),
        utd_image2 varchar2 (80),
        utd_image3 varchar2 (80),
        utd_image4 varchar2 (80),
        utd_image5 varchar2 (80),
        utd_image6 varchar2 (80),
        utd_image7 varchar2 (80),
        utd_image8 varchar2 (80),
        utd_user1 varchar2 (80),
        utd_user2 varchar2 (80),
        utd_mod_date date,
        utd__qad01 varchar2 (80),
        utd__qad02 varchar2 (80),
        progress_recid number null
    ) TABLESPACE gui;

    CREATE UNIQUE INDEX utd_det##progress_recid on utd_det
    (progress_recid) TABLESPACE gui_idx;

    CREATE UNIQUE INDEX utd_det##utd_user_prog on utd_det
    (U##utd_userid, U##utd_lang, U##utd_program) TABLESPACE
    gui_idx;

    DROP SEQUENCE vue_mstr_SEQ;

    CREATE SEQUENCE vue_mstr_SEQ START WITH 1 INCREMENT BY 1
```

```
CACHE 75;

DROP TABLE vue_mstr;

CREATE TABLE vue_mstr(
      U##vue_name varchar2 (30) NULL,
      vue_name varchar2 (30) NULL,
      U##vue_desc varchar2 (80),
      vue_desc varchar2 (80),
      U##vue_table varchar2 (30) NULL,
      vue_table varchar2 (30) NULL,
      vue_cansee varchar2 (80),
      vue_filter varchar2 (80),
      vue_userid varchar2 (80),
      vue_mod_date date,
      vue_user1 varchar2 (80),
      vue_user2 varchar2 (80),
      vue__qad01 varchar2 (80),
      vue__qad02 varchar2 (80),
      progress_recid number null
) TABLESPACE gui;

CREATE UNIQUE INDEX vue_mstr##progress_recid on vue_mstr
(progress_recid) TABLESPACE gui_idx;

CREATE INDEX vue_mstr##vue_desc on vue_mstr (U##vue_desc,
U##vue_table, progress_recid) TABLESPACE gui_idx;

CREATE UNIQUE INDEX vue_mstr##vue_mstr on vue_mstr
(U##vue_name) TABLESPACE gui_idx;

CREATE INDEX vue_mstr##vue_table on vue_mstr
(U##vue_table, U##vue_name, progress_recid) TABLESPACE
gui_idx;

DROP SEQUENCE vuf_det_SEQ;

CREATE SEQUENCE vuf_det_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;

DROP TABLE vuf_det;

CREATE TABLE vuf_det(
      U##vue_name varchar2 (30) NULL,
      vue_name varchar2 (30) NULL,
```

```
      U##vuf_field varchar2 (80) NULL,
      vuf_field varchar2 (80) NULL,
      vuf_datatype varchar2 (30) NULL,
      vuf_format varchar2 (30) NULL,
      vuf_label varchar2 (80),
      vuf_col_label varchar2 (80),
      vuf_cansee varchar2 (80),
      U##vuf_table varchar2 (30) NULL,
      vuf_table varchar2 (30) NULL,
      vuf_expression varchar2 (512),
      vuf_userid varchar2 (80),
      vuf_mod_date date,
      vuf_user1 varchar2 (80),
      vuf_user2 varchar2 (80),
      vuf__qad01 varchar2 (80),
      vuf__qad02 varchar2 (80),
      progress_recid number null
) TABLESPACE gui;

CREATE UNIQUE INDEX vuf_det##progress_recid on vuf_det
(progress_recid) TABLESPACE gui_idx;

CREATE UNIQUE INDEX vuf_det##vuf_det on vuf_det
(U##vue_name, U##vuf_table, U##vuf_field) TABLESPACE
gui_idx;

DROP SEQUENCE vwj_det_SEQ;

CREATE SEQUENCE vwj_det_SEQ START WITH 1 INCREMENT BY 1
CACHE 75;

DROP TABLE vwj_det;

CREATE TABLE vwj_det(
      U##vue_name varchar2 (30) NULL,
      vue_name varchar2 (30) NULL,
      vwj_seq number NULL,
      vwj_table varchar2 (30) NULL,
      vwj_join varchar2 (255),
      vwj_userid varchar2 (80),
```

```
      vwj_mod_date date,
      vwj_user1 varchar2 (80),
      vwj_user2 varchar2 (80),
      vwj__qad01 varchar2 (80),
      vwj__qad02 varchar2 (80),
      progress_recid number null
) TABLESPACE gui;

CREATE UNIQUE INDEX vwj_det##progress_recid on vwj_det
(progress_recid) TABLESPACE gui_idx;

CREATE UNIQUE INDEX vwj_det##vwj_det on vwj_det
(U##vue_name, vwj_seq) TABLESPACE gui_idx;


spool off
```



APPENDIX C

# Oracle Performance Monitoring Tools

Before running any performance monitoring tool, be aware that most monitoring tools have performance overhead and should not be used on a continuous basis in a production system. Most Oracle performance monitoring might require `timed_statistics` to be set to `true` in the init<SID>.ora file.

## utlbstat/utlestat

The utlbstat/utlestat monitoring tool consists of two scripts that reside in the $ORACLE_HOME/rdbms/admin directory. You run `utlbstat.sql` within server manager to start the monitoring process, and then `utlestat.sql` to end it.

When `utlestat.sql` is executed, it generates a file called `report.txt` that contains performance statistics during the time that utlbstat/utlestat was executed. utlbstat/utlestat should be run once the system is in a stable state to display performance constraints.

Statistics to examine in the utlbstat/utlestat report include:

- Library Cache Statistics and Data Dictionary Cache Statistics

  Examine the cache hit ratio and the total number of reloads. If the cache hit ratios are low, or if the total reloads is greater than 0, you should increase the `shared_pool_size` parameter.

- Consistent Gets, DB Block gets, Physical Reads, and Logical Reads

  Using these statistics, it is possible to work out the database block buffers cache hit ratio as following:

  Logical reads = Consistent Gets + DB Block Gets

  Hit Ratio = 1 – (Physical Reads / Logical Reads)

  If the hit ratio is lower than 95%, increase the `db_block_buffers` parameter, as long as there is enough physical memory.

- Table scans (long tables)

  This statistic contains the number of long tables that are accessed using full table scans instead of indexes. If this parameter is large, check to make sure that the correct indexes are present, and that the optimizer statistics are up to date. Also, try decreasing the `db_multi_block_read_count` parameter so that the optimizer does not favor full table scans.

- Table Fetch by Continued Row

  If this value is large, it indicates that there are rows that are chained to another block. Check to see which tables have chained rows, and rebuild them to avoid row chaining.

- Average Length of Dirty Buffer Write Queue

  If this statistic is greater than 0, it indicates disk contention, and the disk subsystem needs to be tuned.

- Rollback Segment Statistics

  The important information from the rollback statistics is the `trans_tbl_waits` and `undo_bytes_written`. If there are `trans_tbl_waits`, you should increase the number of rollback segments. You can use the `undo_bytes_written` statistic to calculate the average amount of rollback generated by a transaction during the monitoring period.

- Sorts (disk)

  Disk sorts degrade performance, since they take more time and require more resources than sorting in memory. You can reduce the number of disk sorts by increasing the `sort_area_size` parameter. However, do not increase this too much because you can run out of physical memory if there are a lot of user sessions.

- Redo Log Space Requests

  Redo log space requests indicates how many times a user process waited for space in the redo log buffer. Try increasing the init<SID>.ora `log_buffer` parameter so that zero redo log space requests are made.

- Redo Small Copies

  This displays the total number of redo entries with fewer bytes than specified by the init<SID>.ora `log_small_entry_max_size` parameter. These entries are written in the redo buffer under the protection of the redo allocation latch. If Redo Small Copies / Redo Entries is more than 10%, decrease `log_small_entry_max_size` to a size smaller than the average redo entry size. This reduces the number of redo entries copied on the redo allocation latch and allows more redo entries to be copied on the redo copy latch.

- File I/O Statistics

  File I/O should be spread evenly across multiple disk drives. Follow the guidelines mentioned in “Oracle Files Distribution” on page 87 to configure database files. Set `db_multi_block_read_count` to increase the number of blocks read during a single read. Increasing this parameter reduces I/O when full table scans are performed.

# Oracle Enterprise Manager Performance Pack

The Performance Manager, which is part of the performance pack of Oracle Enterprise Manager, is a tool that can display information in Oracle dynamic performance tables in graphical format. It is useful to run Performance Manager during testing to display data such as cache hit ratio, number of active sessions, I/O rates, disk sorts, and so on. You may also find it useful to run Oracle Trace and Oracle Expert to gather statistics and generate tuning recommendations during benchmark tests or on production databases.

# OS Monitoring Tools

Most hardware vendors provide their own tools to monitor CPU and disk usage, which provide useful information on the source of constraints. Here are some things to examine using the operating system monitoring tools.

## CPU Utilization

See *Oracle7 for Unix Performance Tuning Tips.*

On a well-tuned system, most of the CPU utilization should be in user time, not system time. If the ratio of system time is greater than 30% of overall CPU utilization, look for disk contention, latch contention, and any other type of resource contention.

# I/O Distribution

Vendor-supplied performance monitoring tools can provide information on I/O distribution among various disks. Look for disks with high I/O and long disk queues compared to other drives, and redistribute Oracle files to even out the I/O.

# Memory Usage

See *Oracle7 for Unix Performance Tuning Tips*

Examine memory usage to ensure that paging and swapping are not taking place because you are running out of physical memory. Size your

SGA and shadow processes so that everything can run in physical memory without resorting to virtual memory.

# General Tuning Tips

## Optimizer Statistics

It is important that statistics on tables are up to date for the optimizer to make intelligent decisions in executing queries. If the data distribution in tables has changed significantly, it is important to recompile statistics on tables. When generating statistics for tables, it is always better to use the `compute` option. However, for large tables, using `estimate` with a sample size of 10 to 20% gives a reasonably good approximation of the statistics for the table. For tables with indexed columns and skewed distributions, use the `histogram` feature of Oracle 7.3 to generate histograms for tables.

## alert<SID>.log

The alert<SID>.log file contains different types of useful tuning information. One is a record of online redo log activity. The frequency of log switches and the duration and frequency of checkpoints can be deduced by examining this file. If log switches and checkpoints are happening too frequently, increase the size of the online redo logs. Monitoring the alert<SID>.log file lets you catch abnormal events and errors.

## Rebuilding Indexes

Oracle indexes can grow unnecessarily tall if rows are inserted in increasing indexed column order. Taller indexes take longer to scan which can degrade performance. Check the height of commonly used indexes to determine if the indexes are unnecessarily tall. If indexes grow in height even though the table data has been purged, drop and rebuild the index. The `rebuild` option of the `alter index` command can be used to re-create the index from the existing index.

### Free List Contention

Monitor the V$WAITSTAT view to see if there are any waits for the class `free list`. If the number of waits is greater than 1% of the total number of requests (db block gets + consistent gets), re-create tables with larger values for `freelists`.

### Initrans Contention

Monitor the V$LOCK view to see if there are sessions waiting for TX locks in mode 4. If this is happening, re-create frequently updated tables with higher values for `initrans`. Set `initrans` to 10 or less.

## User Capacity

Several tuning areas depend on the number of connections. There are Oracle parameters necessary to allow a specified number of connections to the database engine, and there may be areas of operating system tuning required based on your configuration.

### Program Global Area (PGA) Size

Oracle allocates the PGA when you connect, creating a session. The PGA is a memory area containing data and control information for a single connection. PGA must be available at connect time for each user; therefore, the amount of free server memory limits the number of concurrent connections. Each user's PGA can allocate anywhere from 400k to 2M of server memory, depending upon the application. PGA size is affected by:

- `open_links`
- `db_files`
- `log_files`

### Processes

The `processes` init<SID>.ora parameter must be adjusted. For Windows NT this parameter specifies the number of threads that Oracle can create. This number must also include the Oracle service and background threads. The number of Oracle service threads is two and the

number of background threads is typically six. Therefore, the number must be at least the maximum concurrent connections (concurrent connections cannot exceed 500) plus two service threads and six background threads.

## Dynamic Space Management

Make sure that all database objects such as tables, indexes, and rollback segments are sized appropriately so that dynamic space allocation does not take place. You can tell when dynamic space allocation/deallocation is taking place by examining the `recursive calls` statistics in the utlbstat/utlestat report and checking to see if the number of extents are increasing in database segments.

## Programmatic Interface

The Oracle Call Interface (OCI) provides better granularity and control than PRO*. Use deferred linking and parsing. These options reduce network trips to the database in a client/server environment.