

Progress Programming for MFG/PRO: Introduction

TRAINING GUIDE



70-2865A
PROGRESS V9.1
Database: Train eB2
September 2003

This document contains proprietary information that is protected by copyright. No part of this document may be photocopied, reproduced, or translated without the prior written consent of QAD Inc. The information contained in this document is subject to change without notice.

QAD Inc. provides this material as is and makes no warranty of any kind, expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. QAD Inc. shall not be liable for errors contained herein or for incidental or consequential damages (including lost profits) in connection with the furnishing, performance, or use of this material whether based on warranty, contract, or other legal theory.

Some states do not allow the exclusion of implied warranties or the limitation or exclusion of liability for incidental or consequential damages, so the above limitations and exclusion may not be applicable.

PROGRESS® is a registered trademark of Progress Software Corporation.

QAD® and MFG/PRO® are registered trademarks of QAD Inc. QAD eQ and the QAD logo are trademarks of QAD Inc. MFGx.net and Supply Visualization are service marks of QAD Inc.

All other products and company names are used for identification purposes only and may be trademarks of their respective owners.

© Copyright 2003 by QAD Inc. All Rights Reserved.

70-2865A

QAD Inc.

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

LearningServices@qad.com
<http://www.qad.com/services/learn/>

Contents

ABOUT THIS COURSE	11
Course Description	12
Who Should Attend This Course	12
Prerequisites	12
Approximate Length of Course	12
Configuration and Setup for This Course	12
Minimum Requirements	12
Instructor Setup	13
General Training Facilities Information	14
Course Overview	15
 SECTION 1 DATABASE AND ENVIRONMENT	 17
 CHAPTER 1 DATABASE AND PROGRAMMING	 19
4GL: 4th Generation Programming Language	21
Query/Results	22
Client Networking	23
Oracle DataServer	24
File Types	25
Standard File Types	26
Database File Types	28
Database Structures	29
Physical Structure	30
Multi-Volume Overview	31

Multi-Volume Components	32
Disk Layout Example	33
MFG/PRO Directory Structure	34
Logical Structure	35
Progress Schema	40
Tables	43
Records	45
Fields	47
Data Types: Default Display Formats	49
Indexes	50
Triggers	52
Sequences	53
Validation	54
Progress Components	55
Database Manager	56
Procedure Editor	57
MFG/PRO Databases	58
MFG/PRO Naming Conventions	60
Tables	63
Indexes	64
Database Relationships	66
File Relationships Manual	67
File Relationship Symbols	68
Procedure Editor Access: MFG/PRO Character Client	70
Procedure Editor Access: MFG/PRO Character Window	71
Menus	72
Data Dictionary: Character Window	75
Data Dictionary Menus	76
Recent Progress Versions	84
What Is a Relational Database?	85
What Is a Database File?	87
File Relationships Example	88
Indexing	89
Compound Indexes	90
Qualifying Field and Table Names	91

Data Types	92
Default Initial Values	93
Progress Editor	94
Progress Key Functions	95
The Data Administration Tool	96
Connecting to a Database	97
Selecting a Working Database	98
Data Dictionary	99
Create a Table	100
Create a Field	101
Create an Index	102
Sequences	103
Dictionary Reports	104
Dumping/Loading Data and Defs	105
Lab 1	106
 SECTION 2 PROCEDURAL CODING	 109
 CHAPTER 2 WORKING WITH RECORDS	 111
Syntax Elements	112
Data Buffers	113
Data Movement	114
INSERT Statement	115
CREATE and UPDATE	116
DISPLAY and SET	117
PROMPT-FOR and ASSIGN	118
FIND Statement	119
Record Buffers	121
DELETE Statement	122
PAUSE Statement	123
USERID Function	124
Lab 2	125
Lab 2 Review	126

CHAPTER 3 BLOCKS 127

Blocks	128
Programming Impact	129
How Blocks Work	130
Block Properties	131
DO Blocks	132
REPEAT Blocks	134
FOR Blocks	136
Nested Blocks vs. Joins	137
Nested Blocks	138
Inner Joins	139
UNDO Processing	140
NO-ERROR Option	141
AVAILABLE Function	142
IF...THEN...ELSE Statements	143
NAMED Blocks	144
Lab 3	145
Lab 3 Review	146

CHAPTER 4 TRANSACTIONS AND RECORD SCOPING 147

Scoping	148
Programming Impact	149
Sample Listing	150
Data Integrity	151
Transactions	152
Managing Transactions	153
Transactions and the .bi File	154
Development Objectives	155
Starting a Transaction	156
TRANSACTION Function	157
Transaction Scope	158
Development Objectives II	159
Transaction Scope	160
Subtransactions	161

Record Locking	163
Record Contention	165
Releasing Locks	166
Record Scope	167
Programming Impact	168
Types of Record Scope	169
Scope Reference	170
Block Scope Reference	171
Strong Scoped Records	172
Medium Scoped Records	173
Weak Scoped Records	174
Free References	175
 CHAPTER 5 PREDICATES, INDEXES, AND FIELD LISTS	 177
Predicates, Indexes, and Field Lists	178
WHERE Expression	179
Equality, Range, and Sort Matches	180
Operators—Logical	181
Operators—Comparison	182
BEGINS Operator	183
MATCHES Operator	184
CONTAINS Operators	185
Sorting with BY Option	186
USE-INDEX Option	187
Indexing	188
Simple WHERE Clause	189
Compound WHERE Clause 1	190
Compound WHERE Clause 2	191
Choosing a Single Index	192
XREF Output	193
Word Index	194
Unique Index - Equality Matches	195
Most Active Equality Matches	196
Most Active Range Matches	197

Most Active Sort Matches	198
Alphabetic Index Selection	199
Primary Index	200
PRESELECT Phase	201
Bracketing	202
Indexing Guidelines	203
Field Lists	204
Field List Benefits	206
Lab 4	207
Lab 4 Review	208

CHAPTER 6 VARIABLES, MESSAGES, AND CONVERSIONS 209

DEFINE VARIABLE Statement	210
NO-UNDO	211
Screen Input: INPUT	212
STATUS Statement	213
Key Translation Functions	214
Message Statement	215
ALERT-BOX Phrase	216
STRING, SUBSTRING Functions	217
TRIM Function	218
CASE Statement	219
IF...THEN...ELSE Function	220
Editing Phrase	221
FRAME-FIELD, FRAME-VALUE	222

CHAPTER 7 VALIDATION AND HELP 223

Validation Overview	224
Field-Level Validation	225
Session-Defined Validation	226
VALIDATE Statement	227
VALIDATE() Method	228
CAN-FIND Function	229
CAN-DO Function	230

Field-Level Help	231
APPLHELP.P	234
Manipulating the PROPATH	235
Lab 5	236
Lab 5 Review	237
CHAPTER 8 FRAMES.....	239
Frames in PROGRESS	240
Frame Characteristics	241
Allocation: Top-Down Compile	242
Default Frame Appearance	243
Appearance: Frame-Phrase	244
Frame Scope	247
Determining Down Frames	250
Frame Scope	251
Controlling Frames	252
Creating Named Down Frames	256
Frame Flashing	257
FORM Statement	258
DEFINE FRAME Statement	259
FORMAT Phrase	260
Frame Phrase	261
Lab 6	262
Lab 6 Review	263
CHAPTER 9 REPORTING AND INPUT/OUTPUT.....	265
Aggregate Phrase	266
BREAK Group	267
ACCUMULATE Statement	268
ACCUM Function	269
Input/Output	270
OUTPUT TO Statement	271
Redirecting Output	272
DISPLAY Statement	273

EXPORT Statement	279
PUT Statement	282
Headers and Footers	287
Output to Multiple Streams	288
INPUT Statement	289
IMPORT Statement	290
Operating System Commands	291
Lab 7	293
Lab 7 Review	294
Code Structures	295
4GL Functions	296
User-Defined Functions	297
Include Files	300
Progress Procedures	301
RUN Statement	302
Internal Procedures	303
Internal Procedure Limitations	304
RETURN Statement	305
External Procedures	306
Passing Parameters	307
DEFINE PARAMETER Statement	308
DEFINE VARIABLE Statement	309
Persistent Procedures	310
MFG/PRO Code Details	311
Maintenance Program Template	312
Inquiry Program Template	320
Report Program Template	324

CHAPTER 1

About This Course

Course Description

This course provides an overview to using the Progress 4GL programming language up through Progress version 9. The material is selected and presented to highlight the basics of using simple Progress Queries and how MFG/PRO applications function. Fundamental database layout, data movement, and basic Progress transaction and record management are covered. Tools and language to aid in reporting are also covered. The course includes hands-on labs and sample programs to review.

At the conclusion of the course, students should have a general understanding of how to use the Progress 4GL programming language, how it relates to use within MFG/PRO, and how to obtain further information and training if needed.

Who Should Attend This Course

- Beginning programmers who plan to customize reports within MFG/PRO
- Project managers who want to understand the fundamentals of MFG/PRO as it pertains to Progress
- Members of MFG/PRO implementation teams

Prerequisites

- Some knowledge of MFG/PRO, as it is used in your business
- Programming experience a plus, but not required

Approximate Length of Course

- This course is designed to be taught in three days

Configuration and Setup for This Course

Minimum Requirements

Software

- PROGRESS Version 9.1b + installed.

Additional features which will enhance the course:

- Pentium 200 with 64 MB RAM

- Floppy diskette per student to save work created during class.

Instructor Setup

In NT System | Environment folder set the DLC env var to c:\dlc

In the same folder, add c:\dlc\bin to the path.

Set the default procedure editor font:

Select Options | Default Font, go to font #16 and select Edit... set to 18 point Arial.

Each day:

In DOS window, start a server from the working directory (c:\temp\progress):

```
_mprosrv c:\progress\wrk\reachv9
```

Launch the client session:

In the procedure editor:

```
connect reachv9
```

Set the propath using setpropath.p. For example:

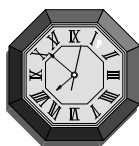
```
Propath = "c:\temp\progress\code, c:\temp\progress\labs," + propath.
```

The Instructor workstation also needs the ability to display the PowerPoint presentation slides to the class.

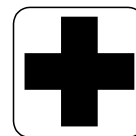
Facilities



Telephone/Fax



Class Hours



Emergency



Messages



Breaks



Restrooms



Parking



Smoking

PR-IN-010

General Training Facilities Information

- Telephone or fax
- Messages
- Restrooms
- Class hours: start and finish time, and punctuality
- Breaks: frequency, approximate times
- Parking considerations; carpooling
- Emergency procedures, location of first aid, contact person for assistance
- Exit locations, building hours
- Location of approved smoking areas

Course Overview

- Database and Environment
- Procedural Coding
- Working With Records
- Blocks
- Transactions and Record Scoping
- Predicates, Indexes & Field Lists
- Variables, Messages & Conversions
- Validation & Help
- Frames
- Reporting & Input/Output

PR-IN-015

Course Overview

SECTION 1

Database and Environment

CHAPTER 1

Database and Programming

Agenda

Part I: Database & Environment

Part II: Procedural Coding

- Working with Records
- Blocks
- Transactions and Record Scoping
- Predicates, Indexes, & Field Lists
- Variables, Messages & Conversions
- Validation & Help
- Frames
- Reporting & Input/Output

PR-PT2-010

Agenda

4GL - 4th Generation Programming Language

- Used to create MFG/PRO Application
- Retrieve and display data
- Assign and update data
- Format screen displays and definitions
- Character and GUI development
- Integrated transaction control and record locking

PR-PT1-010

4GL: 4th Generation Programming Language

Progress 4GL is the programming language that MFG/PRO is written in. There are more than 9,000 programs within MFG/PRO.

Query / Results

- Data access tool and search engine
- Develop reports and inquiries
- Fully utilizes relational databases
- Used by MFG/PRO for browse creation
- Limited screen formatting

PR-PT1-020

Query/Results

Query/Results is the Progress tool to create queries and reports.

- One license per user required because MFG/PRO uses the query engine to create the browses

Client Networking

- Remote access to database and AppServers
- Required for client/server connections
- Windows client connection to Unix database
- Requires TCP/IP for database connections
- Uses hosts –H and services –S files

PR-PT1-030

Client Networking

Client Networking is a free product that may need to be specifically ordered.

- Multi-platform connectivity
 - Connection between unlike operating platforms

Oracle DataServer

- Connectivity to Oracle databases
- Schema holder connection for MFG/PRO

PR-PT1-040

Oracle DataServer

DataServer product is used to connect to the Oracle database through the schema holder.

File Types

- **Standard** – input/output and application files
- **Database** – Directly related to a specific database

PR-PT1-050

File Types

Standard

Files that are not required or necessary for the database, but are necessary for applications to run.

Database

Usually identified by the database name, they are the files that physically hold the database and its definitions.

Standard File Types

Progress Procedure (source)	.p
Include Files	.i
Compiled Procedures (object)	.r
Output	.prn
Validation	.v
Trigger	.t

PR-PT1-060

Standard File Types

.p	Uncompiled code
.i	Incomplete code segments used in multiple files
.r	Compiled code
.prn	MFG/PRO report output file
.v	Field testing and validation
.t	Used for database integrity checking/Triggers

Standard File Types

OBCM	.w
Data (Input)	.d
Database Schema Definition	.df
Bulk Load File Description	.fd
Results Table Join	.qc (.qc7)
Procedure Library	.pl

PR-PT1-070

Standard File Types, cont.

.w	Object-Based Component Model
.d	Import/export files containing specific table data
.df	Commands for creating a database; the database structure itself
.fd	Table load instructions; field descriptions within a table
.qc	Contains the table relationships within the database
.pl	GUI button images

Database File Types

Database Header	.db
Database Extent	.dx
Before Image	.bi (bx)
After Image	.ai (ax)
Log	.lg
Lock	.lk
License	.lic
Structure Description	.st

PR-PT1-080

Database File Types

.db	Single volume database file; or roadmap to multi-volume database files and information
.dx	Database extent number (dbname.d5)
.bi	The before-image file or extent number file
.ai	The after-image file or extent number file
.lg	Reporting log of database activity
.lk	Existence of this file indicates the database in use; updateable mode
.lic	Keeps track of number of user log-ins
.st	A map for creating a database

Database Structures

- **Physical** – Structural Design
- **Logical** – Referential Design

PR-PT1-090

Database Structures

Physical

Where the files are located

Logical

Where the data is located

Physical Structure

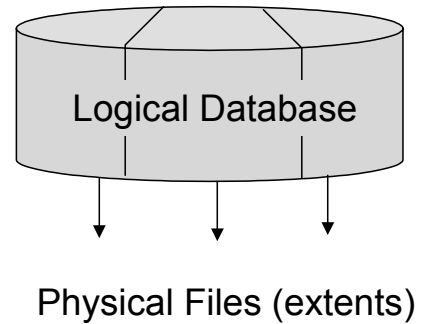
- Disk Layout
- Database File Locations
- MFG/PRO Application File Layout

PR-PT1-100

Physical Structure

Multi-Volume Overview

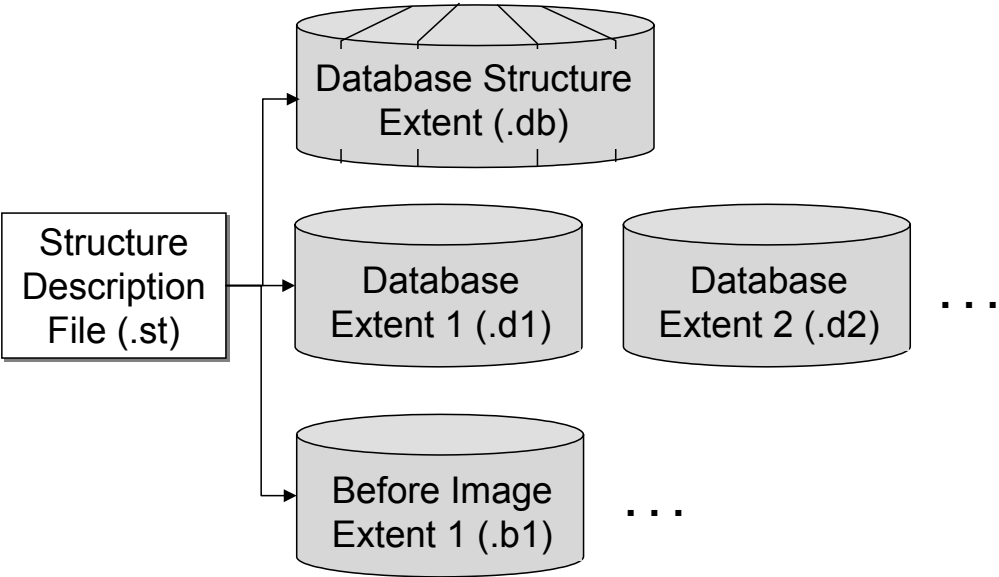
- One logical database comprised of several physical files
- Physical files placed across several physical disk drives for maximize I/O
- Likewise for After Image and Before Image files



PR-PT1-110

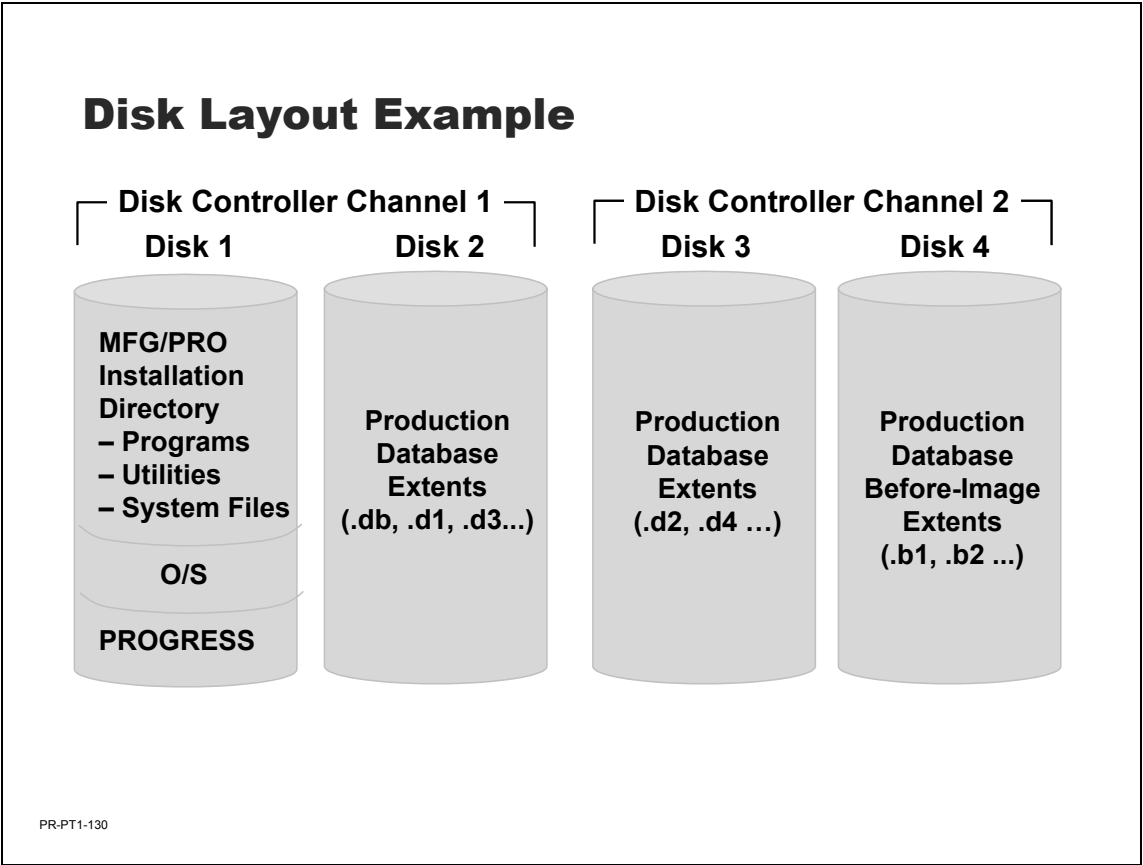
Multi-Volume Overview

Multi-Volume Components



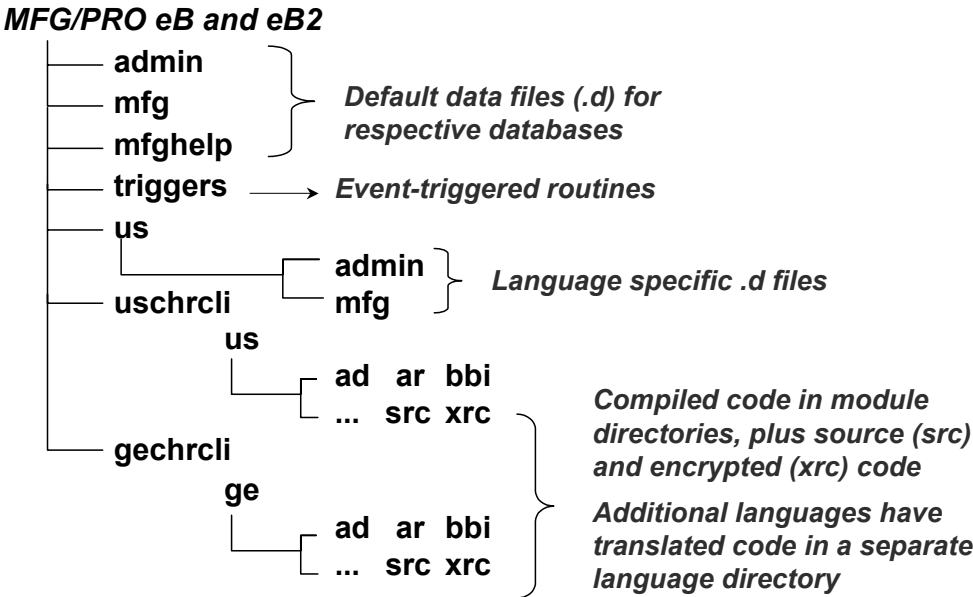
PR-PT1-120

Multi-Volume Components



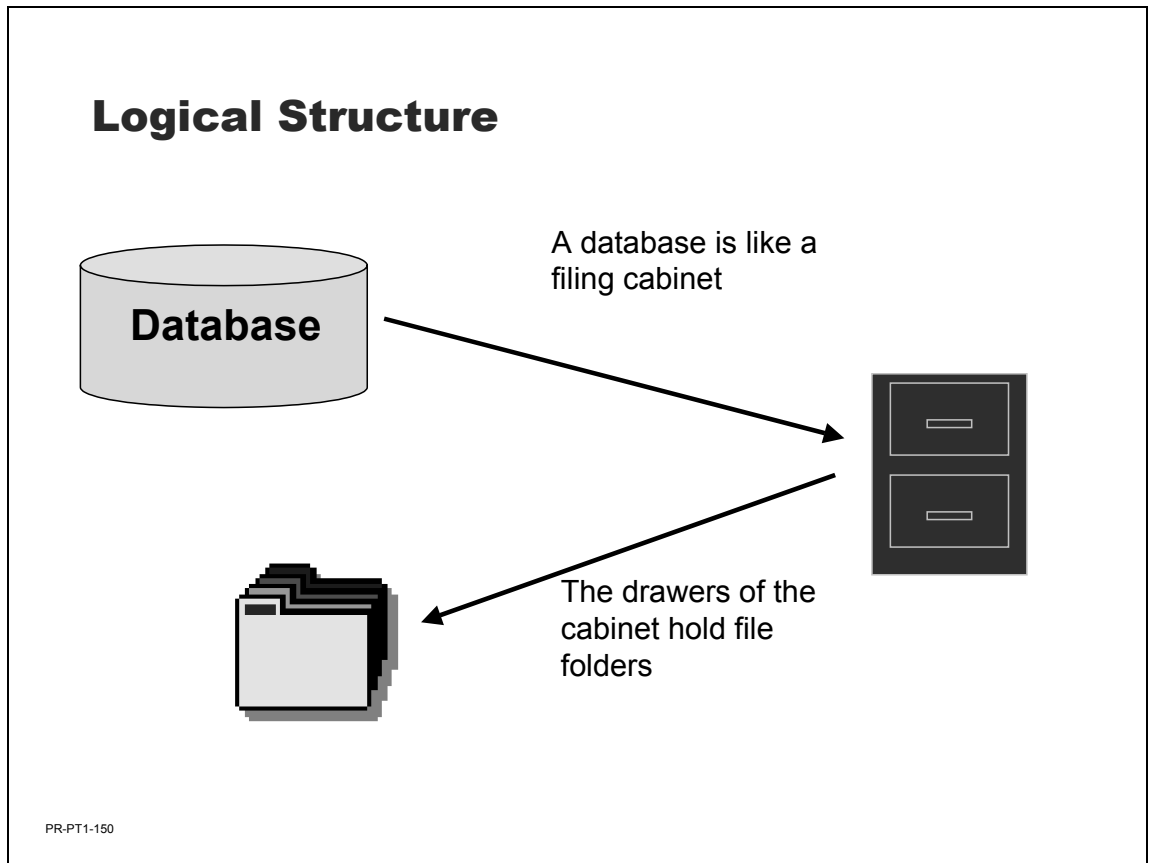
Disk Layout Example

MFG/PRO Directory Structure



PR-PT1-140

MFG/PRO Directory Structure

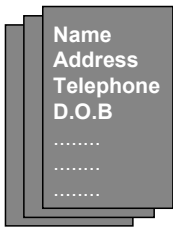


Logical Structure

Logical Structure



A file holds a group of related information such as Customer Names and Addresses. These files are called Tables.



A particular Table holds many records, each of which details a particular object such as Employee.

Record segments, like Name, Address and Date of Birth, are known as *FIELDS*.

PR-PT1-160

Logical Structure, cont.

Logical Structure

Database



Each file in the cabinet represents a Table

Tables



Record

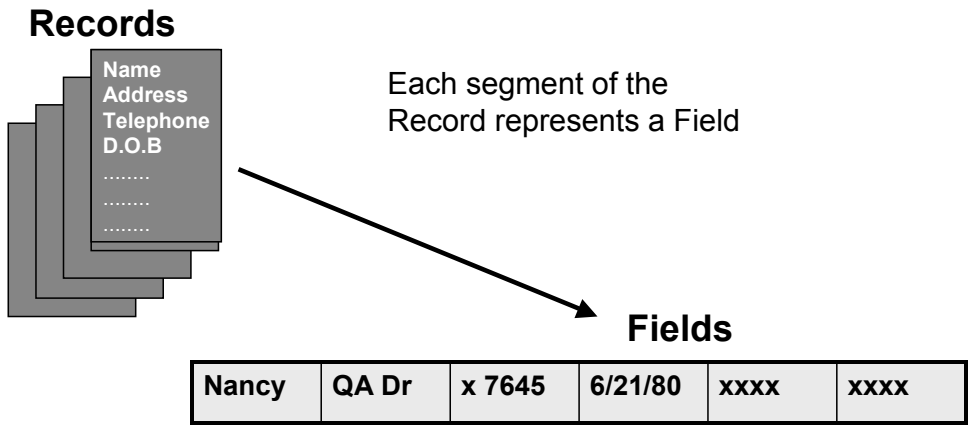


Each entry in the Table represents a Record

PR-PT1-170

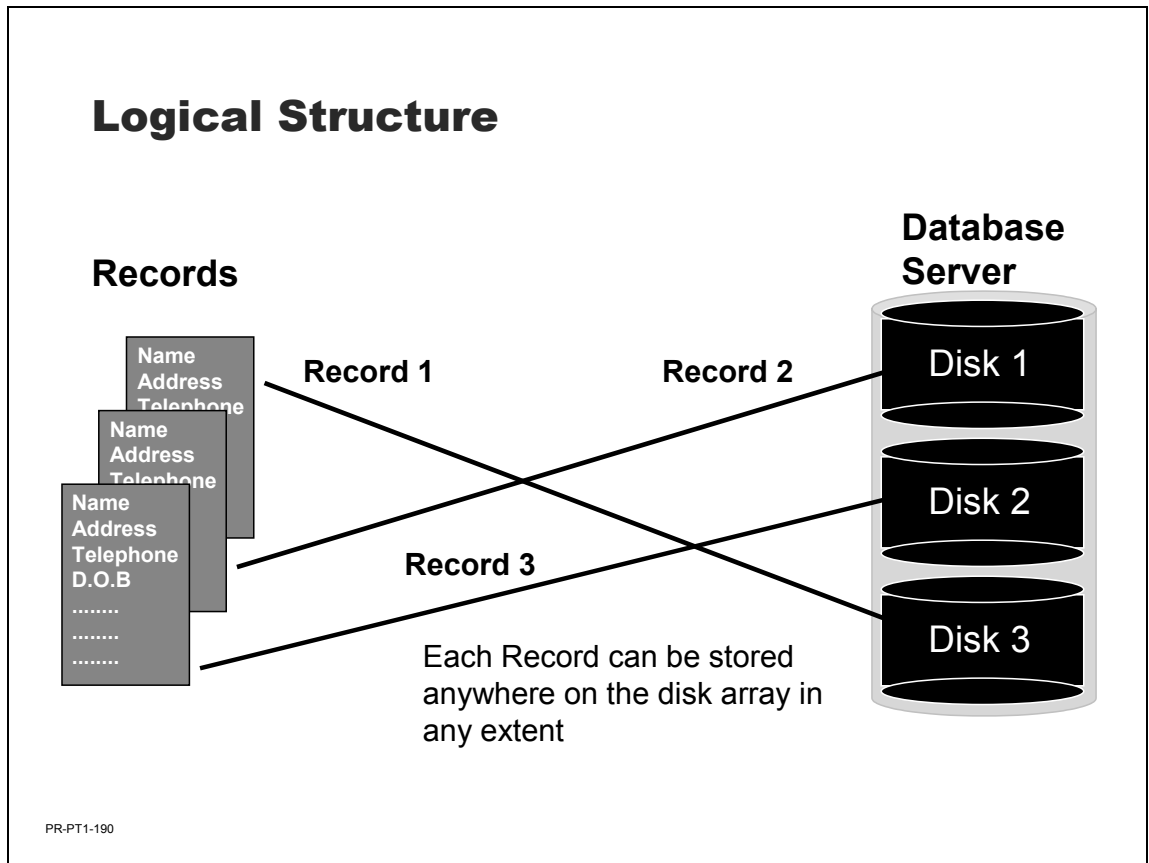
Logical Structure, cont.

Logical Structure



PR-PT1-180

Logical Structure, cont.



Logical Structure, cont.

Progress Schema

- **Table**
- **Field**
- **Indexes**
- **Trigger**
- **Sequence**
- **Validation**

PR-PT1-200

Progress Schema

Progress Schema

- **Table** – Collection of similar data
- **Field** – One item of information in a Table
- **Indexes** – Structure to locate a specific record

PR-PT1-210

Progress Schema, cont.

Progress Schema

- **Trigger** – Event-driven procedure
- **Sequence** – Pointer to next occurrence
- **Validation** – Security and data integrity test

PR-PT1-220

Progress Schema, cont.

Tables

- Collection of similar data
- Contains records or rows of data
- Contains fields or columns of data
- Indexed for performance

PR-PT1-230

Tables

Tables

	<u>Field 1</u>	<u>Field 2</u>	<u>Field 3</u>	<u>Field 4</u>	<u>Field 5</u>	<u>Field 6</u>
	Name	Addr	Phone	DOB		
Rec 1	Bill	Main St	x 2887	5/11/67	xxxx	xxxx
Rec 2	Bob	1 st Ave	x 9845	8/19/60	xxxx	xxxx
Rec 3	Nancy	QA Dr	x 7645	6/21/80	xxxx	xxxx
Rec 4	Sam	10 th St	x 8347	9/18/45	xxxx	xxxx
Rec 5	Mary	Box 32	x 8934	2/12/77	xxxx	xxxx

Each Row is a Record in the Table

Each column is a Field of the Table

PR-PT1-240

Tables, cont.

Records

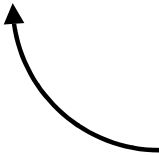
- Data in one row of the table
- Contain data in multiple fields
- Identified by unique recid in Database

PR-PT1-250

Records

Record

	<u>Field 1</u>	<u>Field 2</u>	<u>Field 3</u>	<u>Field 4</u>	<u>Field 5</u>	<u>Field 6</u>
	Name	Addr	Phone	DOB		
Rec 3	Nancy	QA Dr	x 7645	6/21/80	xxxx	xxxx



Each Row is a Record in the Table

PR-PT1-260

Records, cont.

Fields

- One segment of data in a record
- Stores data of particular type
- Multiple fields within a record
- One column in a Table

PR-PT1-270

Fields

Fields

	<u>Field 1</u>	<u>Field 2</u>	<u>Field 3</u>	<u>Field 4</u>	<u>Field 5</u>	<u>Field 6</u>
	Name	Addr	Phone	DOB		
Rec 1		Main St				
Rec 2		1 st Ave				
Rec 3		QA Dr				
Rec 4		10 th St				
Rec 5		Box 32				

Each column is a Field in the Table
This column is the Addr field

PR-PT1-280

Fields, cont.

Data Types: Default Display Formats

- Character "x(8)"
- Date "99/99/99"
- Decimal "->>, >>9.99"
- Integer "->, >>>, >>9"
- Logical Yes/No

PR-PT1-290

Data Types: Default Display Formats

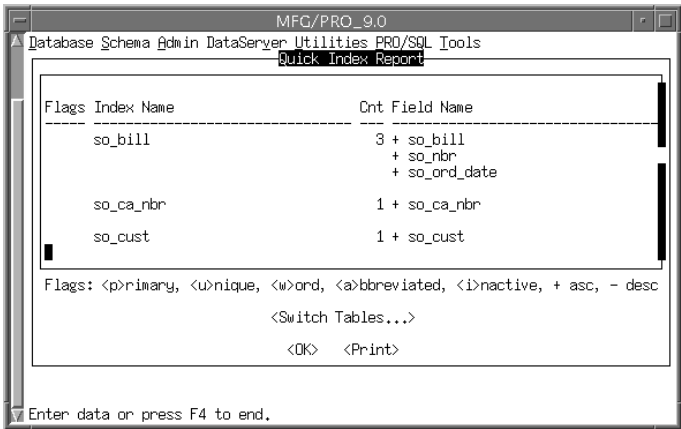
Indexes

- An index sorts the records of a Table
- Index is based on one or more fields in the Table
- Value in the field decides place in the index
- Unique index – only one record in the Table with a specific combination of values

PR-PT1-300

Indexes

Indexes



PR-PT1-310

Indexes, cont.

Triggers

- Event-driven procedure
- Set up in the schema or in the program code
- Program code takes precedence
- .t programs

PR-PT1-320

Triggers

Sequences

- Auto-generated incremental integer values
- Stores the next occurrence
- Separate from the database Tables
- Definable increment values

PR-PT1-330

Sequences

Validation

- Field-level test function
- Data integrity
- Security
- .v files

PR-PT1-340

Validation

Progress Components

- Procedure Editor
- The Data Dictionary
- Database Manager

PR-PT1-350

Progress Components

Database Manager

Enterprise RDBMS
(Relational Database Management System)

- Access Control
- Performance utilities
- Creation, Backup, Recovery utilities
- Compiling
- Procedure Editor
- Data Dictionary

PR-PT1-360

Database Manager

Procedure Editor

- Database Access
- Text Editor Window
- Menus
- Data Dictionary

PR-PT1-370

Procedure Editor

MFG/PRO Databases

- Databases
- Naming Conventions
- Tables
- Relationships

PR-PT1-380

MFG/PRO Databases

MFG/PRO Databases

8.x/9.0	eB
<ul style="list-style-type: none"> • Production • Help • GUI • CFG 	<ul style="list-style-type: none"> • Production • Help • Admin

PR-PT1-390

MFG/PRO Databases, cont.

MFG/PRO Naming Conventions

- Underscore in field and file names
- Master files (Headers and Footers) are named xxxx_mstr and detail records are named xxxd_det (note the d!)
- Field names are prefixed by the filename prefix; that is, so_mstr has a field called so_nbr
- Control files (holding default system values) are named xxxc_ctrl

PR-PT1-400

MFG/PRO Naming Conventions

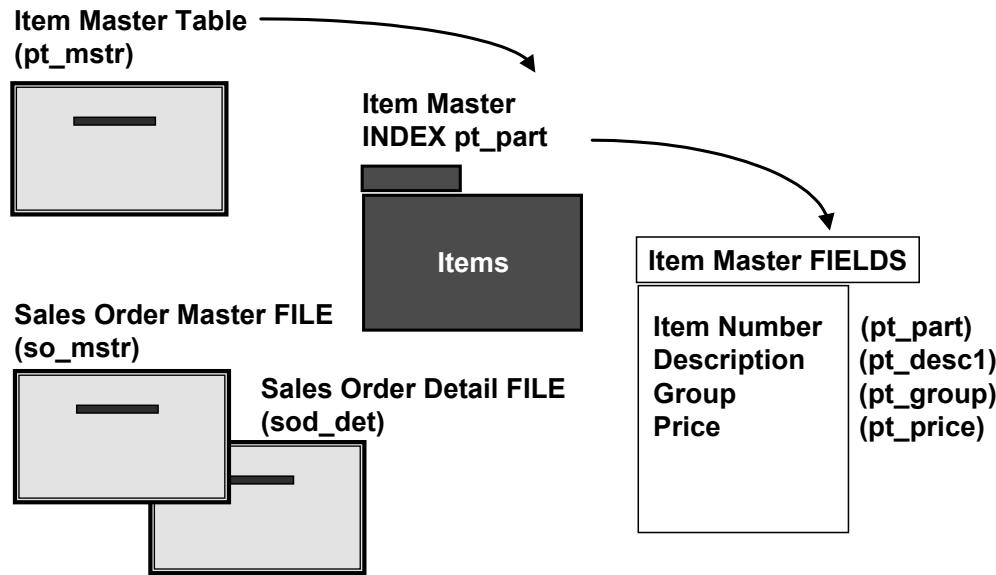
Naming Conventions

- History Files (used to record data after certain events, such as invoice posting) are named xxxx_hist
- Internal system workfiles are named xxxx_wkfl. The qad_wkfl is used by QAD development staff *ONLY!*
- User fields can either be *Double or Single Underscore Userx*; that is, xx__ or xx_user1 etc.
- Index names usually mimic field names

PR-PT1-410

Naming Conventions, cont.

Naming Conventions



PR-PT1-420

Naming Conventions, cont.

Tables

so_mstr – Sales Order Master

cm_mstr – Customer Master

ca_mstr – Call Master

tr_hist – Transaction History

mnd_det – Menu Detail

arc_ctrl – Accounts Receivable Control File

qad_wkfl – QAD Workfile

PR-PT1-430

Tables

Indexes

so_mstr has 10 indexes

- so_bill
 - so_bill
 - so_nbr
 - so_ord_date
- so_nbr (primary, unique)
 - so_nbr
- so_to_inv(unique)
 - so_to_inv
 - so_nbr

PR-PT1-440

Indexes

Indexes

ca_mstr has 11 indexes

- ca_area
 - ca_category
 - ca_area
 - ca_eu_nbr
- ca_assign
 - ca_assign
 - ca_category
- ca_nbr (primary, unique)
 - ca_category
 - ca_nbr

PR-PT1-450

Indexes, cont.

Database Relationships

Sales Order File (so_mstr)

<i>so_nbr</i>	<i>so_cust</i>
SO00011	A111
SO00022	B111
SO00033	B111

Customer File (cm_mstr)

<i>cm_addr</i>	<i>cm_name</i>
A111	Smith Ltd
B111	Jones Ltd
C003	Thompson Plc

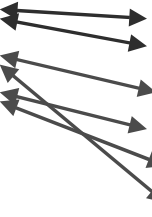


Part File (pt_mstr)

<i>pt_desc1</i>	<i>pt_part</i>
Front Seat	001
Door	002
Wheel	003

Sales Order Detail File (sod_det)

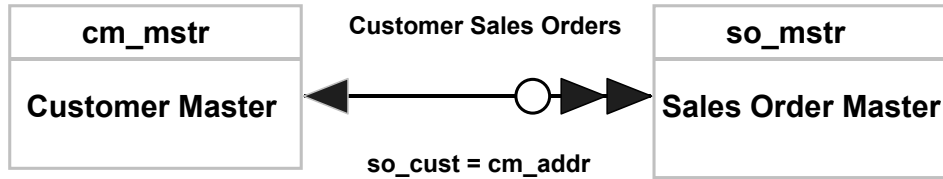
<i>sod_part</i>	<i>sod_nbr</i>	<i>sod_qty_item</i>
001	SO00011	3
001	SO00022	6
002	SO00022	10
003	SO00022	20
003	SO00033	4
002	SO00033	2



PR-PT1-460

Database Relationships

File Relationships Manual

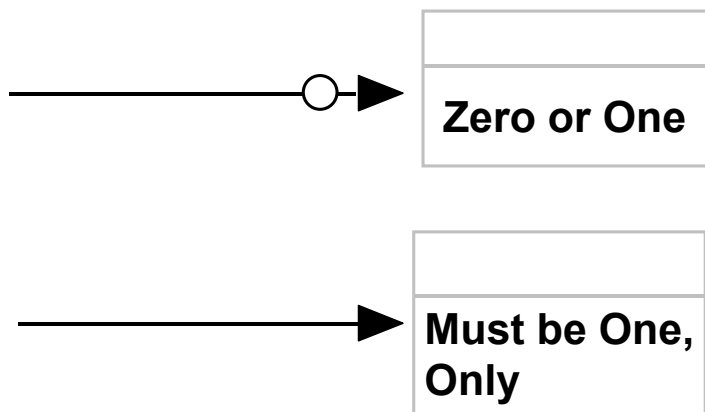


```
FOR EACH cm_mstr :  
  FOR EACH so_mstr WHERE so_cust = cm_addr:  
    DISPLAY cm_addr so_nbr.  
  END.  
END.
```

PR-PT1-470

File Relationships Manual

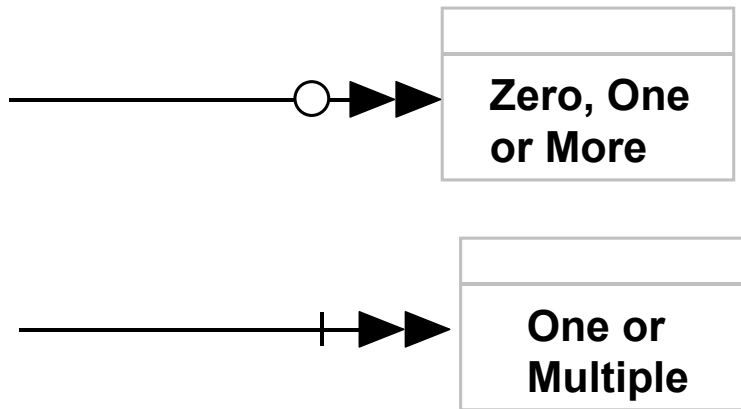
File Relationship Symbols



PR-PT1-480

File Relationship Symbols

File Relationship Symbols



PR-PT1-490

File Relationship Symbols, cont.

Procedure Editor Access: MFG/PRO Character Client

- From the MFG/PRO main menu, menu option line, press F4 as you would to exit
- Type “P” at the ‘Please confirm exit’ prompt (typing “Y” quits MFG/PRO, “N” returns you to the menu option line)
- Press the Enter or Return key
- Access to PROGRESS can be restricted through Password Maintenance

PR-PT1-500

Procedure Editor Access: MFG/PRO Character Client

Procedure Editor Access: MFG/PRO Character Window



PR-PT1-510

Procedure Editor Access: MFG/PRO Character Window

Menus

File

New	ESC-N
Open...	F5
Close	F8
Save	F6
Save As...	ESC-A
Print	
Exit	

Edit

Cut	F10
Copy	F11
Paste	F12
Insert Files. . .	
Insert Fields. . .	

PR-PT1-520

Menus

Menus

Search

Find. . .	CTRL-F
Find Next	ESC-F
Find Previous	ESC-I
<u>Replace. . .</u>	ESC-R
Goto Line. . .	ESC-G

Buffer

List. . .	ESC-B
Next Buffer	ESC-TAB
Previous Buffer	ESC-CTRL-U
Information. . .	

PR-PT1-530

Menus, cont.

Menus

Compile

<u>R</u> un	F1
<u>C</u> heck Syntax	ESC-P
<u>D</u> ebug	
<hr/>	
Compiler <u>M</u> essages...	

Tools

<u>D</u> ata Dictionary
<u>Q</u> S Shell
Application <u>C</u> ompiler

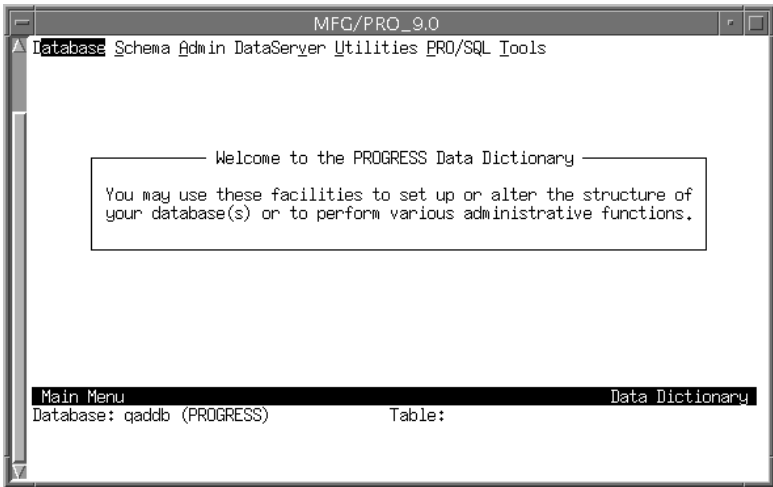
Help

<u>M</u> essages...	
<u>R</u> ecent Messages...	
<u>K</u> eyboard...	
<hr/>	
<u>A</u> bout Procedure Editor	

PR-PT1-540

Menus, cont.

Data Dictionary: Character Window



PR-PT1-550

Data Dictionary: Character Window

Data Dictionary Menus

Database

Select Working Database...

Create...

Connect...

Disconnect...

Reports ->

Exit

PR-PT1-560

Data Dictionary Menus

Data Dictionary Menus

Schema

Modify Table...

Add New Table...

Delete Table(s)...

Field Editor...

Reorder Fields...

Global Field Name Change...

Index Editor...

Sequence Editor...

PR-PT1-570

Data Dictionary Menus, cont.

Data Dictionary Menus

Admin

Dump Data and Definitions ->

Load Data and Definitions ->

Security ->

Export Data ->

Import Data ->

Create Bulk Loader Description File...

PR-PT1-580

Data Dictionary Menus, cont.

Data Dictionary Menus

Dump Data and Definitions ->

Data Definitions (.df file) . . .
Table Contents (.d file) . . .
SQL Views . . .
User Table Contents . . .
Auto-Connect Records only . . .
Collation Tables . . .
Sequences Definitions . . .
Sequences Current Values . . .

Create Incremental .df File . . .

PR-PT1-590

Data Dictionary Menus, cont.

Data Dictionary Menus

Load Data and Definitions ->

- Data Definitions (.df file) . . .
- Table Contents (.d file) . . .
- SQL Views . . .
- User Table Contents . . .
- Sequences Current Values . . .
- Reconstruct Bad Load Records . . .

PR-PT1-600

Data Dictionary Menus, cont.

Data Dictionary Menus

Utilities

Editor for Parameter Files...

Quoter Functions ->

Generate Include Files ->

Edit PROGRESS Auto-Connect List...

Freeze/Unfreeze...

Index Deactivation...

Information...

PR-PT1-610

Data Dictionary Menus, cont.

Data Dictionary Menus

PRO/SQL

PRO/SQL View Report

Dump as CREATE VIEW Statement...

Dump as CREATE TABLE Statement...

PR-PT1-620

Data Dictionary Menus, cont.

Data Dictionary Menus

Tools

Procedure Editor

QS Shell

Application Compiler

PR-PT1-630

Data Dictionary Menus, cont.

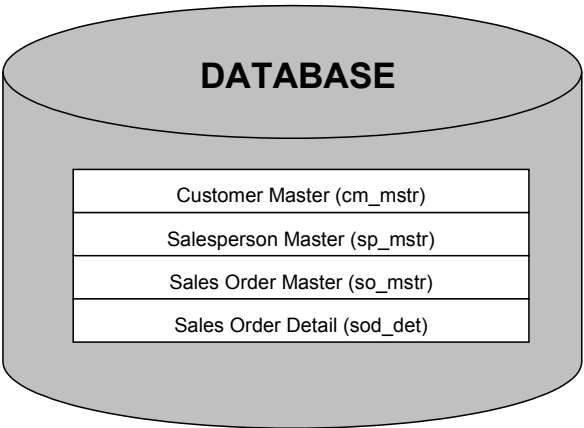
Recent Progress Versions

- 8.2: first 32-bit version of Progress (ADM 1)
- 8.3: Marketing's version of 8.2D (ADM 1)
- 9.0: first implementation of ADM 2
 - Separated the UI from the business logic
 - Many new features added
 - See www.v9stuff.com.
- 9.1B: significant WebClient is added
- 9.1D: current version

PR-PT1-640

Recent Progress Versions

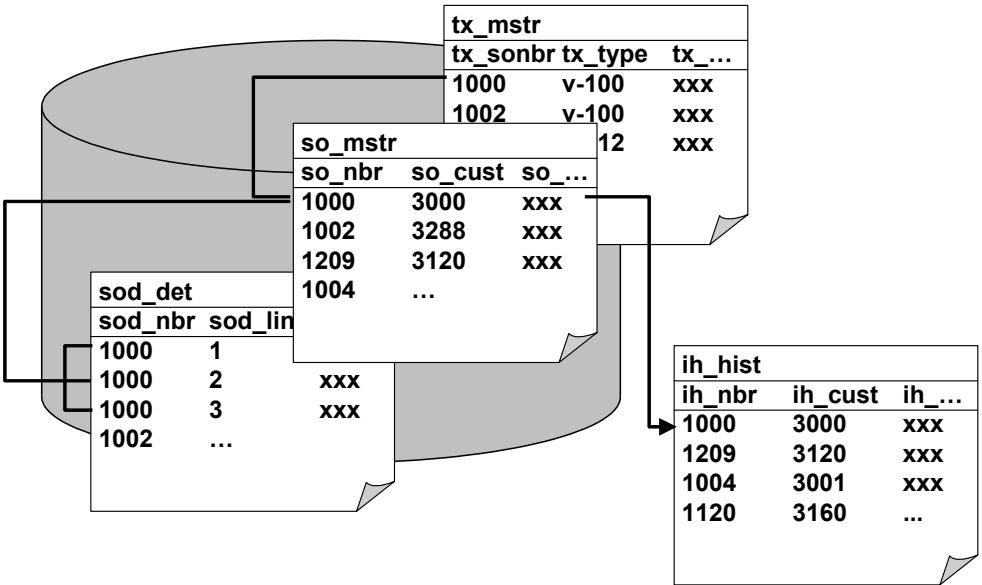
What Is a Relational Database?



PR-PT1-650

What Is a Relational Database?

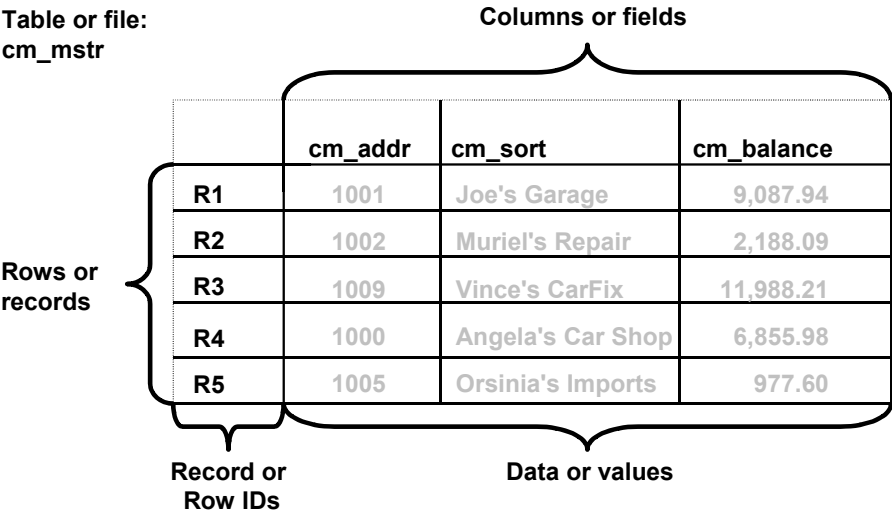
What Is a Relational Database?



PR-PT1-660

What Is a Relationship Database, cont.

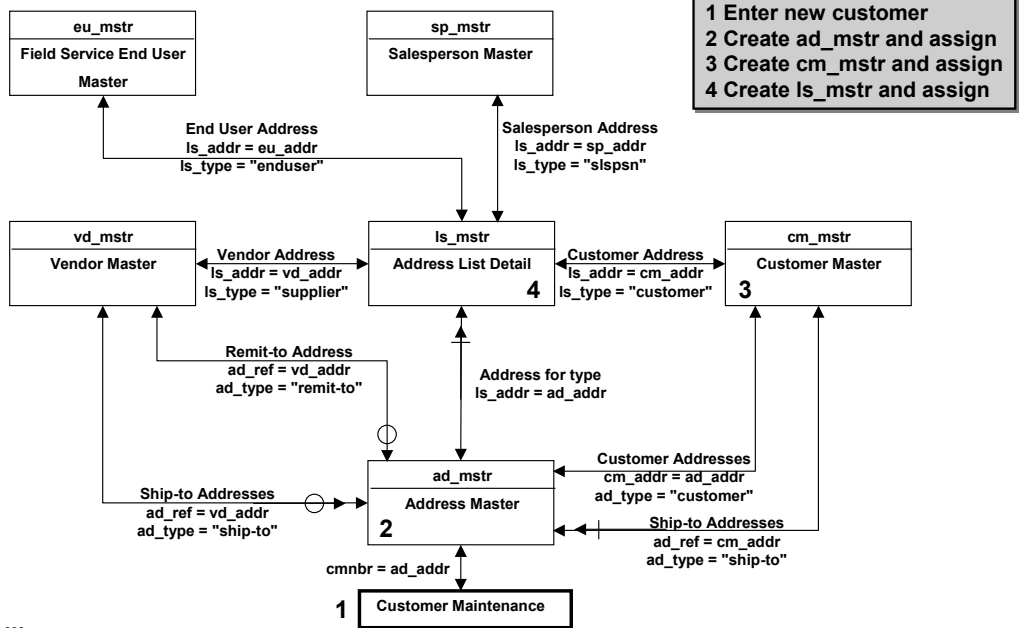
What Is a Database File?



PR-PT1-680

What Is a Database File?

File Relationships Example



PR-PT1-690

File Relationships Example

Indexing

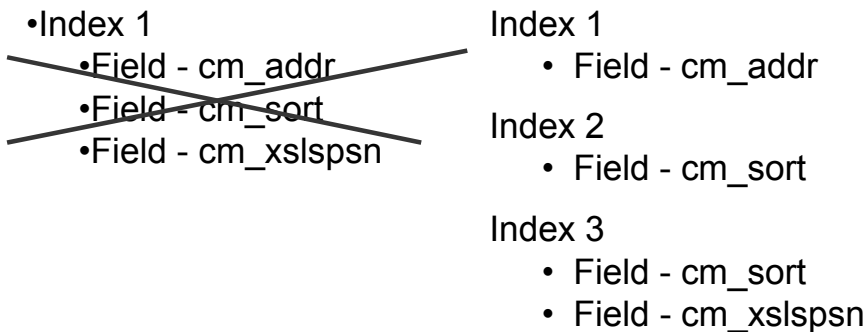
- Fast retrieval
 - satisfy search criteria without opening record
 - automatic record ordering
 - may contain multiple fields (compound index)
 - MUST BE ACCESSED IN FIELD ORDER
- Overhead
 - when a record is created, an index entry is created
 - when a record is deleted, an index entry is deleted
 - when an indexed field is updated, the index is updated
 - DO NOT CREATE INDEXES NEEDLESSLY

PR-PT1-700

Indexing

Compound Indexes

- Get the best “bracket” you can
 - See PROGRESS Database Design Guide, section 4



PR-PT1-710

Compound Indexes

Qualifying Field and Table Names

- When to use more than just the field name ?
 - Filename.Fieldname
 - Used when the field name is shared among other files, buffers, workfiles, work-tables, or temp-tables
 - Database.Filename.Fieldname
 - Used when the file name is shared among other connected databases

PR-PT1-720

Qualifying Field and Table Names

Data Types

Data Types	Description
Character	contains any type of data (letters, numbers, and special characters)
Integer	holds whole numbers, positive or negative, ranging from -2,147,483,648 to 2,147,483,647
Decimal	contains decimal numbers up to 50 digits in length, including up to ten places to the right of the decimal
Date	holds dates ranging from 1/1/32768 B.C. to 12/31/32767 A.D. (MFG/PRO's hi_date = 12/31/3999)
Logical	values that evaluate to a Yes/No or True/False

PR-PT1-730

Data Types

Default Initial Values

Data Types	Initial Value
Character	null string
Integer	0
Decimal	0
Date	? (unknown value)
Logical	no/false
RECID	? (unknown value)

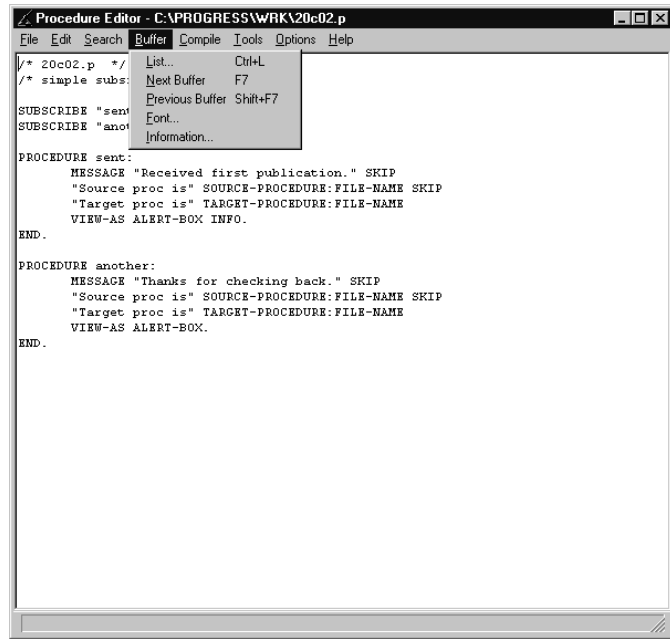
Null strings and unknown values display as blanks

PR-PT1-740

Default Initial Values

Progress Editor

- File management
- Cut & paste
- Searching
- Multiple buffers
- Compiles & debugging
- Data dictionary
- 4GL Help & messages help



PR-PT1-750

Progress Editor

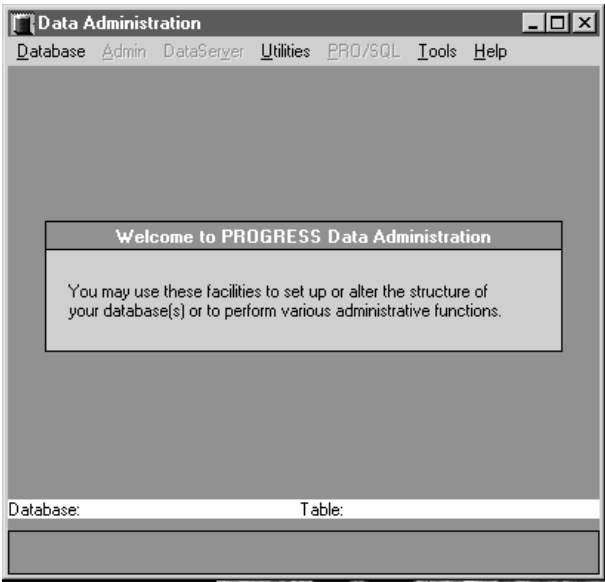
Progress Key Functions

Key	Function	Key	Function
F1	HELP	F8	CLOSE
F2	GO	CTRL+F8	FIND
SH+F2	Check Syntax	F9	FIND NEXT
F3	File Open	SH+F9	FIND PREV.
SH+F3	New Proc. File	CTRL+R	REPLACE
CTRL+F3	New Buffer	CTRL+Z	UNDO
SH+F4	Debug	CTRL+X	CUT
F6	SAVE	CTRL+C	COPY
SH+F6	SAVE AS	CTRL+V	PASTE
F7	Next Buffer	CTRL+G	Go To Line
SH+F7	Prev. Buffer	CTRL+E	Compiler Messages

PR-PT1-760

Progress Key Functions

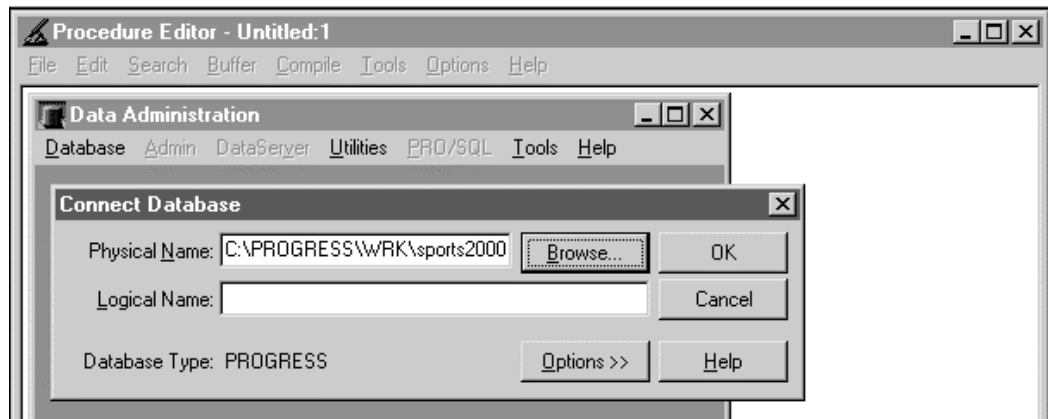
The Data Administration Tool



PR-PT1-770

The Data Administration Tool

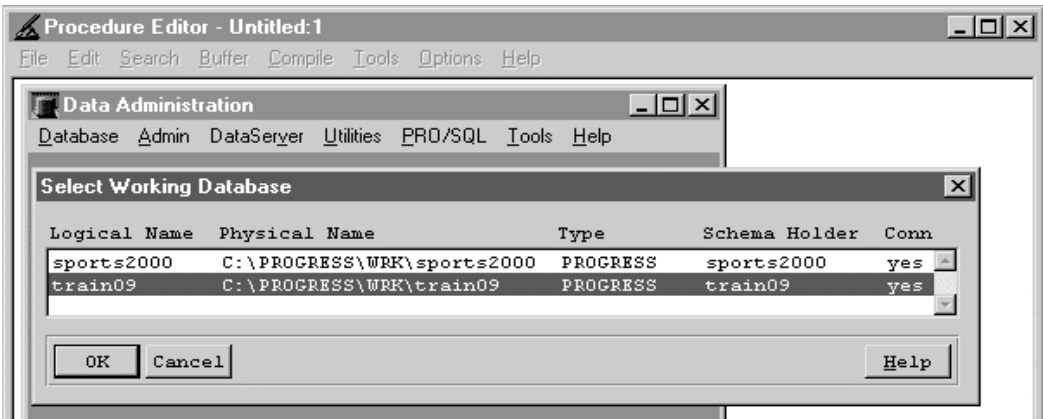
Connecting to a Database



PR-PT1-780

Connecting to a Database

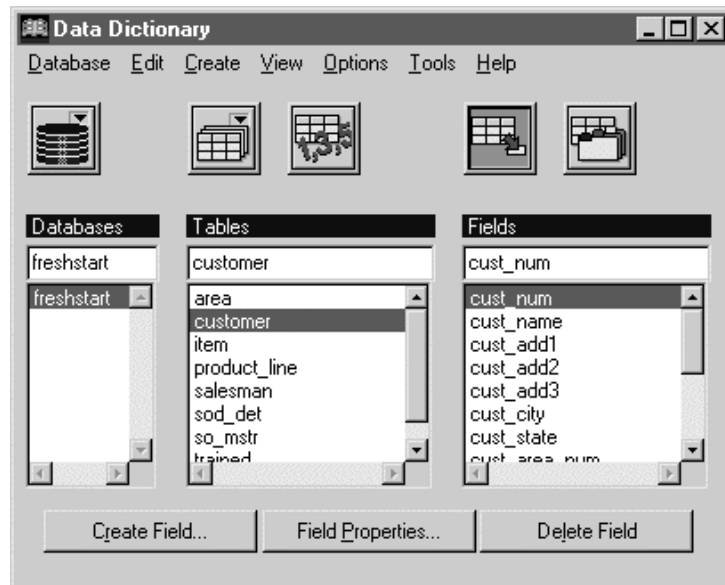
Selecting a Working Database



PR-PT1-790

Selecting a Working Database

Data Dictionary



PR-PT1-800

Data Dictionary

Create a Table

Create Table [X]

Table Name:

Optional

Dump File: ☐ Hidden ☐ Frozen

Table Type: PROGRESS

Label:

Description:

Replication:

DB Link: n/a Record Size:

DataServer Name: n/a

Owner: n/a

PR-PT1-810

Create a Table

Create a Field

Create Field for Table area

Field Name: Copy Field...

Data Type: CHARACTER

Optional

Format: X(8)

Label:

Column Label:

Initial Value:

Order #: 30 Decimals: ?

Description:

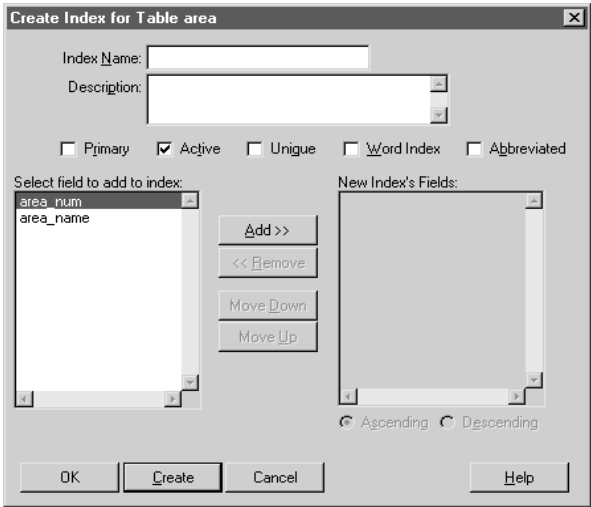
Help Text:

☐ Mandatory ☐ Case-Sensitive ☐ Extent 0

PR-PT1-820

Create a Field

Create an Index



PR-PT1-830

Create an Index

Sequences

- Database objects that provide incremental values to an application
- Can generate sequential values within any range of a PROGRESS integer with your choice of increment (positive or negative)
 - nonterminating - provides continual unique values
 - terminating - terminates after it reaches a specified limit
 - cycling - restarts after it reaches a specified limit
- Limit of 250 to 2000 per database*
- Not continuous, allows for gaps
- Fast access, no lock waits

* depends on db block size (2K to 8K)

PR-PT1-840

Sequences

Dictionary Reports

- Detailed Table
- Quick Table
- Quick Field
- Quick Index
- PRO/SQL View
- Sequence
- Trigger
- User
- Table Relations
- Storage Areas

Detailed Table Report

===== Table: ad_mstr =====

Flags: <c>ase sensitive, <i>ndex component, <m>andatory, <v>iew componen

Order	Field Name	Data Type	Flags	Format	Initia
10	ad_addr	char	i	X(8)	
20	ad_city	char		x(20)	
30	ad_country	char		x(28)	
40	ad_date	date		99/99/99	?
50	ad_fax	char		x(16)	
60	ad_line1	char		x(28)	
70	ad_line2	char		x(28)	
80	ad_line3	char		x(28)	
90	ad_mod_date	date		99/99/99	?
100	ad_name	char		x(28)	
110	ad_phone	char		x(16)	

Change Field Order

OK

Print

Help

PR-PT1-850

Dictionary Reports

Dumping/Loading Data and Defs

```
ADD SEQUENCE "cust_num_seq"
  INITIAL 3003
  INCREMENT 1
  CYCLE-ON-LIMIT no
  MIN-VAL 3003
```

```
ADD TABLE "cm_mstr"
  AREA "Schema Area"
  DESCRIPTION "Customer Master File"
  DUMP-NAME "cm_mstr"
  TABLE-TRIGGER "CREATE" NO-OVERRIDE PROCEDURE "crecust" CRC "29598"
  TABLE-TRIGGER "DELETE" NO-OVERRIDE PROCEDURE "delcust" CRC "19521"
  TABLE-TRIGGER "WRITE" NO-OVERRIDE PROCEDURE "custwrite" CRC "40584"
```

```
ADD FIELD "cm_addr" OF "cm_mstr" AS character
  FORMAT "X(8)"
  INITIAL ""
  LABEL "Customer"
  POSITION 2
  COLUMN-LABEL "Cust"
  ORDER 10
```

- Dump/Load:
 - Data Definitions
 - Table Contents
 - SQL Views
 - User Table Contents
 - Current Sequence Values
 - etc.

Partial Data Definitions (.df) Dump

PR-PT1-860

Dumping/Loading Data and Defs

Lab 1

1. Launch the Data Dictionary
2. Connect to the reachv9 database
3. Add the cm_mstr table, fields and indexes as shown on the next page

All names are important for future dependencies

4. In the Data Administration tool, use the Admin menu to load the data for the table (cm_mstr.d)

PR-PT1-870

Lab 1

```

Table: customer
Order Field Name      Data Type  Flags Format      Initial
-----
10 cm_addr            char      i      X(8)
20 cm_balance         deci-2    ->>, >>>, >>>, >>9 0
30 cm_class           char      X(8)
40 cm_cr_hold         logi      yes/no      no
50 cm_cr_limit        inte      >>>, >>>, >>>, >>9 0
60 cm_cr_rating       char      X(8)
70 cm_cr_terms        char      X(8)
80 cm_sort            char      i      x(28)
90 cm_xslspn         char[2]   X(8)

Field Name            Label      Column Label
-----
cm_addr              Customer   Cust
cm_balance           Balance   Bal
cm_class             Class     Class
cm_cr_hold           Credit Hold Hold
cm_cr_limit          Credit Limit Cr Limit
cm_cr_rating          Credit Rating Cr Rate
cm_cr_terms           Terms     Terms
cm_sort              Sort Name ?
cm_xslspn            Salesperson ?

===== INDEX SUMMARY =====
===== Table: cm_mstr =====
Flags: <p>primary, <u>nique, <w>ord, <a>breviated, <i>nactive, + asc, - desc

Flags Index Name      Cnt Field Name
-----
pu    i_cm_addr        1 + cm_addr
      i_cm_sort        1 + cm_sort

** Index Name: i_cm_addr
Storage Area: N/A
** Index Name: i_cm_sort
Storage Area: N/A

```

PR-PT1-880

SECTION 2

Procedural Coding

CHAPTER 2

Working With Records

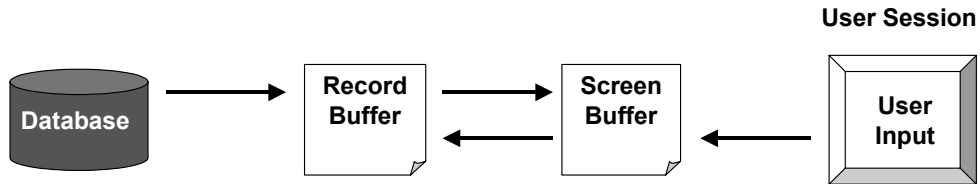
Syntax Elements

- **statement** one complete instruction to Progress
- **phrase** unit within a statement that alters or refines processing of the statement
- **variable** a variable holds data in memory
- **operator** language element used to manipulate, compare or test values in an expression
- **expression** constant, field or variable name, function, or combination of these and an operator
- **procedure** source file containing Progress 4GL code
- **block** sequence of statements treated as a unit
- **function** performs a discreet task within expressions or statements, usually returns a value

PR-PT2-090

Syntax Elements

Data Buffers



The only connection between users is the database

The data is protected through the 4GL:

- All data modifications are first moved to the record buffer

- The screen buffer allows users to see it

- User input allows user modifications

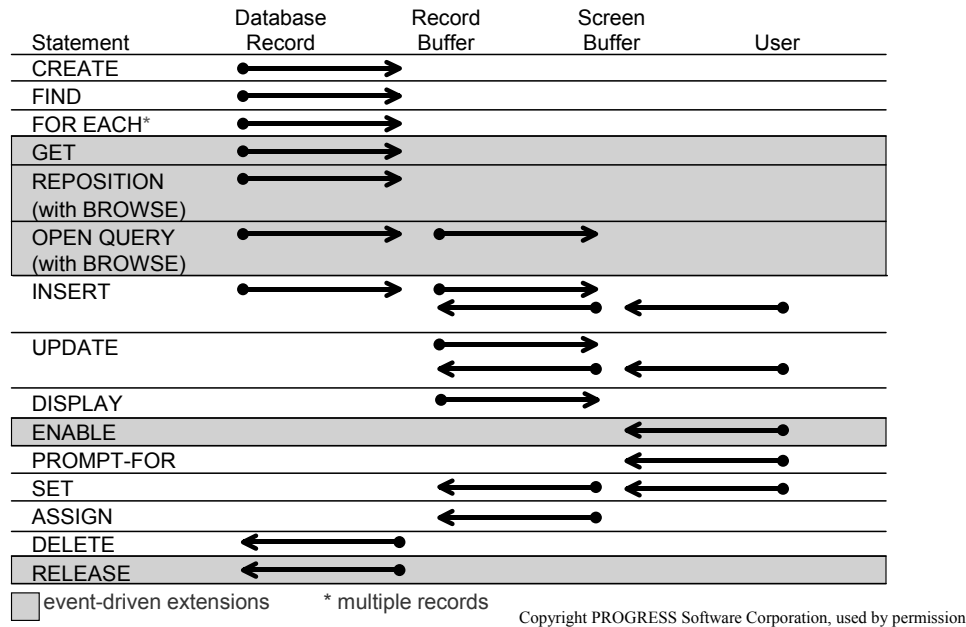
- All changes are written to the record buffer

Progress manages database writes

PR-PT2-100

Data Buffers

Data Movement



PR-PT2-110

Copyright PROGRESS Software Corporation, used by permission

Data Movement

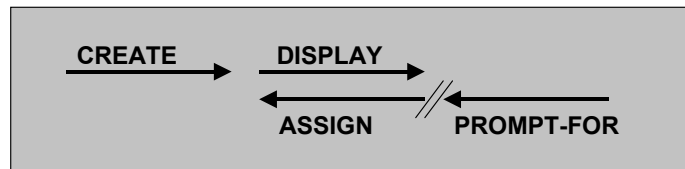
02-001

INSERT Statement

- Statement that performs entire record creation and update task
 - top labels
 - all fields are displayed in Data Dictionary 'order'
 - if no field label is defined, field name is used
 - PROGRESS subscripts arrays

```
INSERT pt_mstr.
```

INSERT =



PR-PT2-120

INSERT Statement

02-002

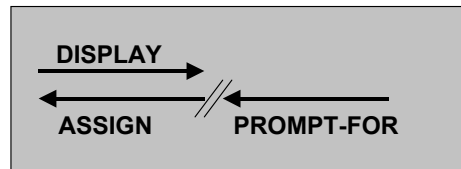
CREATE & UPDATE

- CREATE creates a record in a file, sets fields to default initial values, and moves a copy of the record to the record buffer
- UPDATE displays fields or variables, requests input, and then puts the input data in both the screen buffer and in the specified fields or variables
 - UPDATE controls the TAB sequence among fields

```
CREATE pt_mstr.
```

```
UPDATE pt_mstr.
```

UPDATE =



PR-PT2-130

CREATE and UPDATE

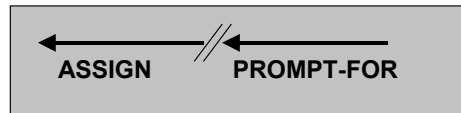
02-003

DISPLAY & SET

- **DISPLAY** moves data to the screen buffer and displays it on the screen or other output destination
- **SET** requests input, and then puts the data in the screen buffer frame and in the specified fields or variables

```
CREATE pt_mstr.  
DISPLAY pt_mstr.  
SET pt_mstr.
```

SET =



PR-PT2-140

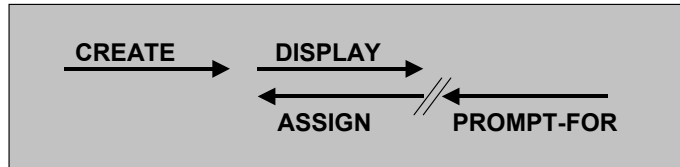
DISPLAY and SET

02-004

PROMPT-FOR & ASSIGN

- PROMPT-FOR makes the fields or the record available for input in the screen buffer
- ASSIGN moves data from the screen buffer to the record buffer

```
CREATE pt_mstr.  
DISPLAY pt_mstr.  
PROMPT-FOR pt_mstr.  
ASSIGN pt_mstr.
```



PR-PT2-150

PROMPT-FOR and ASSIGN

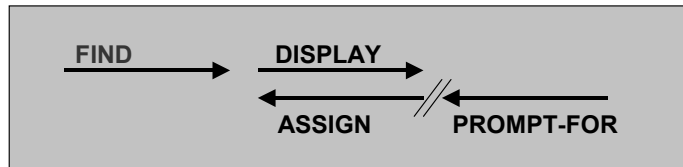
02-005

FIND Statement

- Locates a single record in a table and moves that record into a record buffer
- Raises an error if statement fails
- Optional: FIRST/LAST/NEXT/PREV

```
FIND FIRST pt_mstr.
```

```
UPDATE pt_mstr.
```



PR-PT2-160

FIND Statement

FIND must locate one and only one unique record. If the FIND statement finds two or more records that match the selection criteria, then the FIND actually fails.

02-006

FIND Statement

- USING reads field value from screen buffer
- EXCEPT keyword excludes fields from a statement

```
PROMPT-FOR pt_part.  
FIND FIRST pt_mstr USING pt_part.  
DISPLAY pt_mstr.  
UPDATE pt_mstr EXCEPT pt_part.
```

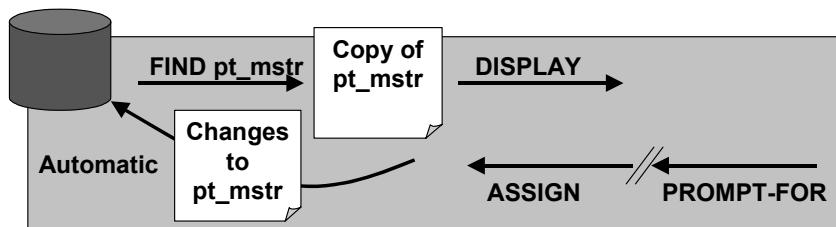
PR-PT2-170

FIND Statement, cont.

- USING command can only be performed with screen input value
- The EXCEPT statement is used to update or display all fields EXCEPT the one(s) listed

Record Buffers

- Every time Progress opens or creates a record, it creates a buffer in memory with the same name as the record created or opened
- Control the size of the buffer area with the startup parameter -Mr n where n is the size in bytes (default is 1012, max is 3200)
- Buffers can be named, copied, compared



PR-PT2-180

Record Buffers

If naming a buffer, it must be explicitly called out in procedure.

02-007

DELETE Statement

- Removes a record from a record buffer and from the database

```
FIND LAST pt_mstr.  
DISPLAY pt_mstr.  
DELETE pt_mstr.
```



PR-PT2-190

DELETE Statement

02-008

PAUSE Statement

- Suspends processing indefinitely, or for specified number of seconds, or until user presses any key
- Method to debug, or separate statements in processing

```
FIND LAST pt_mstr.  
DISPLAY pt_mstr.  
PAUSE.  
DELETE pt_mstr.
```



PR-PT2-200

PAUSE Statement

USERID Function

- PROGRESS functions return a value
- USERID returns current user's ID

```
ASSIGN pt_userID = USERID.
```

PR-PT2-210

USERID Function

The USERID value is the user ID that was logged in to Progress.

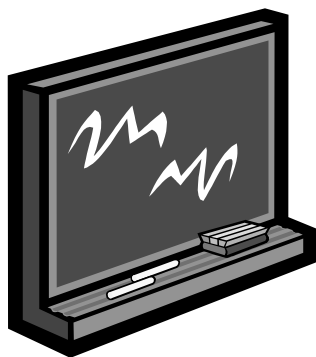
LAB 2

1. Add a part master record. Show the user any initial values. Display the item number first. (L0201.p)
2. Prompt the user for a part number, then find the correct record with the part number. Capture the user's ID in the record and update everything but the part number and user ID. (L0202.p)
3. Allow the user to enter a part number, then find the correct record with the part number and delete the record. (L0203.p)

PR-PT2-220

Lab 2

Lab 2 Review



PR-PT2-230

Lab 2 Review

CHAPTER 3

Blocks

Blocks

- Series of statements Progress treats as a single unit
- Each begins with a header statement and ends with an END statement
 - Except for the default block: procedure block
- Blocks in Progress include:
 - DO
 - REPEAT
 - FOR
 - Trigger
 - Procedure
 - Internal Procedure
 - User-Defined Function

PR-PT2-250

Blocks

Programming Impact

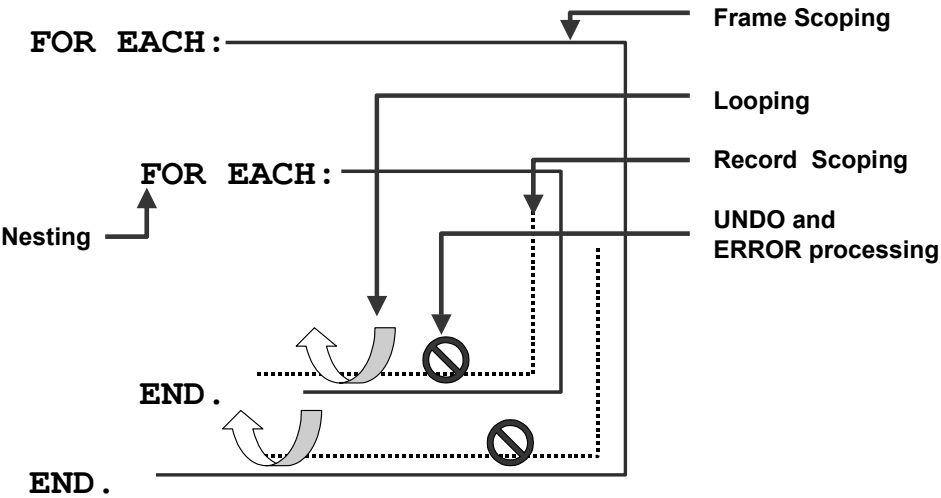
- Blocks are the method used to group activities and assign those grouped activities specific attributes
- Attributes include:
 - looping
 - record reading
 - error processing
 - record scope
 - frame scope
 - nesting and joins
- Impacts compile and runtime conditions (errors)
- Impacts memory usage and performance

PR-PT2-260

Programming Impact

- Record Scope: The time that the record is active in the record buffer
- Frame Scope: The outer most block in which a frame is referenced
- Transaction Scope: The unit of work that is done and undone as a whole

How Blocks Work



PR-PT2-270

How Blocks Work

Block Properties

Property	FOR		REPEAT		DO		Procedure or Trigger	
	Implicit	Explicit	Implicit	Explicit	Implicit	Explicit	Implicit	Explicit
Looping	YES	WHILE TO/BY	YES	WHILE TO/BY	NO	WHILE TO/BY	NO	NO
Record reading	YES	Record Phrase	NO	NO	NO	NO	NO	NO
Frame scoping	YES	WITH FRAME	YES	WITH FRAME	NO	WITH FRAME	YES	NO
Record scoping	YES	NO	YES	FOR	NO	FOR	YES	NO
UNDO	YES	TRANSACTION ON ERROR	YES	TRANSACTION ON ERROR	NO	TRANSACTION ON ERROR	YES	NO
ERROR processing	YES	ON ERROR	YES	ON ERROR	NO	ON ERROR	YES	NO

PR-PT2-280

Copyright PROGRESS Software Corporation, used by permission

Block Properties

- SCOPE is one of the properties of all blocks. It is the duration that a resource is available to an application. Blocks determine the scope of that resource availability.
- FOR – do for all
- REPEAT – do until I tell you not to
- DO – only what I say and how often I say
- Trigger or Procedure – Just do it! When called.

DO Blocks

- Groups statements into a single block, optionally specifying processing services or block properties
- Block executes once rather than iteratively (unless you explicitly tell it to loop)
- Use a DO statement rather than a REPEAT statement when you loop through each element of an array. This way PROGRESS does not create separate subtransactions within a transaction.

PR-PT2-290

DO Blocks

Block executes once and creates a transaction. If use WHILE (DO WHILE), then the block will loop as long as the condition is true.

DO statements for arrayed fields will not cause subtransactions. If a REPEAT loop is used, then each of the arrayed fields will initiate a subtransaction within the main transaction.

02-009 - 02-010

DO Blocks

- DO:
 - Includes the DO WHILE and DO i = 1 to 100 blocks.
- DO FOR <buffername>:
 - scopes a record
- DO TRANSACTION:
 - scopes a transaction
- DO ON ERROR <action>:
 - scopes a transaction
- DO ON ENDKEY <action>:
 - scopes a transaction
- DO WITH <frame name>:
 - scopes a frame

PR-PT2-300

DO Blocks, cont.

- DO, DO WHILE and DO i = : Do while a condition is true, start a transaction
- DO FOR <buffername>: Iterates through the buffer until buffer end. Scopes a record (that is, the function performed to that one record is a single transaction).
- DO TRANSACTION: Scopes the entire block to a transaction. All included functions are scoped to this transaction.
- DO ON ERROR: When an error is encountered, perform an action.
- DO ON ENDKEY: If exit key read (F4 or ESC), then perform an action
- DO WITH FRAME: A frame must be explicitly scoped to a DO transaction type

REPEAT Blocks

- Begins a block of statements that are processed repeatedly until the block ends
- Simple REPEATs are terminated with END-KEY
 - F4 in character / ESC in GUI
 - UNDO, LEAVE at first UPDATE in block
 - UNDO, RETRY after first UPDATE in block
- Repeat blocks scope frames and records
- Default error and undo processing

PR-PT2-310

REPEAT Blocks

If ERROR or END are not specified, default exit behavior is employed.

02-011

REPEAT Blocks

REPEAT:

REPEAT FOR <buffername>:

scopes a record

REPEAT TRANSACTION:

scopes a transaction

REPEAT ON ERROR <action>:

scopes a transaction

REPEAT ON ENDKEY <action>:

scopes a transaction

REPEAT WITH <framename>:

scopes a frame

PR-PT2-320

REPEAT Blocks, cont.

02-012

FOR Blocks

- Iterating block
- Reads zero or more records from one or more tables at the start of each block iteration
- Can access records using multiple indexes
- FOR EACH
- TRANSACTION, ERROR, ENDKEY, WITH <frame>

PR-PT2-330

FOR Blocks

Nested Blocks vs. Joins

- Methods for accessing records from multiple tables in a single statement or block
- Tables are accessed in a parent-child manner
- Nested blocks (outer joins)
 - Displays every parent record and each child record that relates to that parent
- Inner joins
 - Displays only parents that have related child records

PR-PT2-340

Nested Blocks vs. Joins

- Outer Join: Show all data, with relationship as available
- Inner Join: Only show data where relationship is available

02-013

Nested Blocks

- Outer join in a 4GL procedure
- Nested FOR EACH statements
 - the outer FOR EACH statement executes once, with one iteration per matching record
 - the inner FOR EACH statement executes n times
 - where n is the number of records returned by the outer FOR EACH

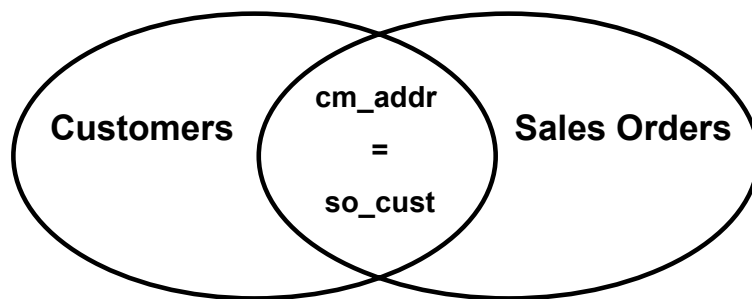
PR-PT2-350

Nested Blocks

02-014

Inner Joins

- A *join* is a binary operation that selects and combines the records from multiple tables so that each result contains a single record from each table
- An *inner join* returns the records selected for the table on the left side combined with the related records selected from the table on the right



PR-PT2-360

Inner Joins

UNDO Processing

- Used in ON ERROR, ON ENDKEY phrases to:
 - RETRY (retries current or named enclosing block)
 - LEAVE (leaves current or named enclosing block)
 - NEXT (iterates current or named block)
 - RETURN (to calling proc from trigger or run proc)
- Default behavior is to restore variable values to value at start of transaction
- NO-UNDO option on DEFINE statements prevents backing out of variable data in memory
 - Use for incrementing values (for example, tracking number of records created)

PR-PT2-380

UNDO Processing

The value in the variable is not **UNDONE** if the transaction is backed out. Variables defined as **NO-UNDO** do not write to the .lbi (Local Before Image) file, therefore they require fewer resources.

Use **NO-UNDO** when the value of a variable is reset after undo of a transaction, or if the significant value is not required later in the process, or if the transaction is re-processed (that is, **RETRY**).

NO-ERROR Option

- Tells PROGRESS not to display error messages for any errors it might encounter
- Possible errors are:
 - not finding a record that satisfies the record-phrase
 - finding more than one record for a unique find
 - finding a record that is locked with the NO-WAIT option on the FIND
- Use the AVAILABLE function to test if FIND found a record.

```
PROMPT-FOR pt_part.  
FIND pt_mstr USING pt_part NO-ERROR.
```

PR-PT2-390

NO-ERROR Option

IF AVAILABLE then do something. IF NOT AVAILABLE then do something.

AVAILABLE Function

- Returns TRUE if the named record buffer contains a record, FALSE if the record buffer is empty
- FIND or FOR statements read database records into a record buffer with the same name as the file used by the FIND or FOR
- Manage availability with FIND or FOR to avoid error condition

```
IF NOT AVAILABLE cm_mstr THEN UNDO, RETRY.
```

PR-PT2-400

AVAILABLE Function

02-015

IF...THEN...ELSE Statements

- Makes the execution of a statement or a block of statements conditional
 - IF *expression* THEN {*statement*}.
 - [ELSE {*statement*}].
- Can nest statements together

```
PROMPT-FOR pt_part.
FIND pt_mstr USING pt_part NO-ERROR.
IF NOT AVAILABLE pt_mstr THEN UNDO, RETRY.
ELSE DO:
    ASSIGN pt_userID = USERID.
    UPDATE
        pt_critical pt_desc1 pt_desc2
        pt_group pt_price pt_status.
    DISPLAY pt_mstr.
END.
```

PR-PT2-410

IF...THEN...ELSE Statements

If the DO: is not used, then only the next statement is executed on the condition. All other statements that follow are executed unconditionally.

NAMED BLOCKS

- Allows reference to a specific block other than default block
- Block label precedes block header

```
blk1:  
REPEAT:  
    PROMPT-FOR pt_part.  
    FIND pt_mstr USING pt_part NO-ERROR.  
    IF NOT AVAILABLE pt_mstr THEN UNDO blk1, RETRY blk1.  
    UPDATE pt_mstr EXCEPT pt_part.  
END.
```

PR-PT2-420

NAMED Blocks

The name of the block **blk1:** does not require an **END** statement to complete the block. The real block starts at the **REPEAT:** statement and ends at the **END**. The **blk1:** statement is just a name for the block.

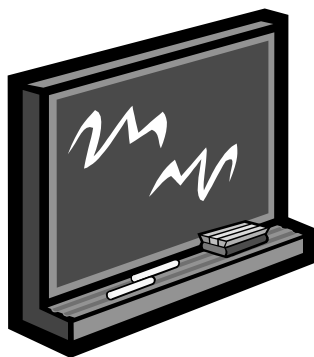
LAB 3

1. Display all item records showing only the part number, description, and price. (L0301.p)
2. Allow users to select an item record for updates. Allow updates to all fields except the userID and part number. Make sure you capture the current user's ID. Handle error and availability conditions. (L0302.p)
3. Add the ability to create a record if it does not exist. (L0303.p)

PR-PT2-430

Lab 3

Lab 3 Review



PR-PT2-440

Lab 3 Review

Transactions and Record Scoping

Scoping

- Persistence of a transaction, record, or frame for purposes of access and memory usage
- Impacts:
 - Transactions
 - controls when changes are committed to the database
 - Records
 - controls buffer creation/close, helps control locks
 - Frames
 - controls activation, attributes, field-level display

PR-PT2-460

Scoping

Programming Impact

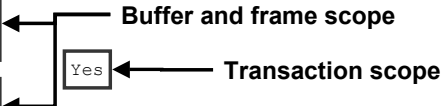
- Scope controls writing of data to the database, record availability, error processing, and user interface behavior
- Closely related to blocks
- Scope can be raised to encompassing blocks or minimized as necessary
- Improper scoping underlies numerous programming errors

PR-PT2-470

Programming Impact

Sample Listing

```
...code\02-015list.p                                03/24/02 15:30:52  PROGRESS(R) Page 1
{} Line Blk
-- -----
1      /* 02-015list.p */
2
3      FIND FIRST sp_mstr NO-LOCK.
4      DISPLAY sp_addr sp_sort.
5
6      1 FOR EACH cm_mstr:
7      1      UPDATE cm_sort cm_balance.
8      END.
9
10     DISPLAY "goodbye" WITH FRAME a.
□      File Name      Line Blk. Type Tran      Blk. Label
-----
...code\02-015list.p      0 Procedure No
  Buffers: reachv9.sp_mstr
  Frames:  a
      Unnamed
...code\02-015list.p      6 For
  Buffers: reachv9.cm_mstr
  Frames:  Unnamed
```

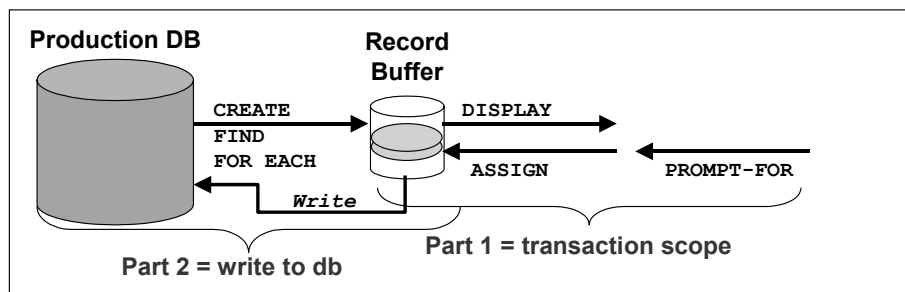


PR-PT2-480

Sample Listing

Data Integrity

- Data integrity means that PROGRESS stores completed data and does not store incomplete data in the database
- PROGRESS uses **transactions** to automatically handle this processing
- Transactions managed in two parts:



PR-PT2-490

Data Integrity

Transactions

- A transaction is a set of changes to memory variables or database records which the system either completes or discards (leaving no modification to the database) in full
- A transaction controls:
 - when records are written to the database
 - how long a record lock remains after the data is changed
 - how much data will be written or discarded on completion or abandonment of the transaction
 - whether variables changed during the transaction will be restored to their original values if a transaction is undone

PR-PT2-500

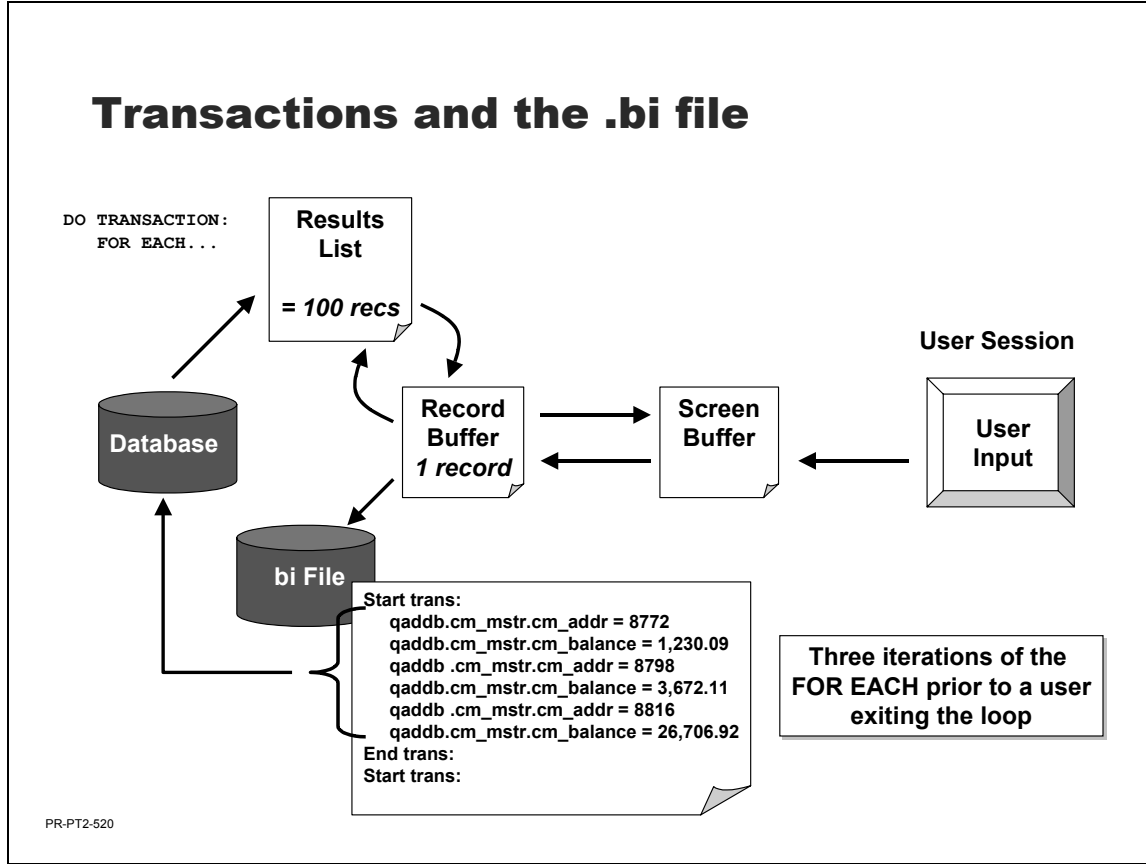
Transactions

Managing Transactions

- Each transaction gets a start note, and if successfully closed, an end note in the Before Image file
- Purpose is to protect the database from partial or corrupt data due to system failures
 - All .BI changes without end notes are backed out on system restart
- Also maintains data integrity, allowing users to undo changes prior to committing them to the database

PR-PT2-510

Managing Transactions



Transactions and the .bi File

Development Objectives

- Does the program behave identically (commit or rollback data) under all conditions?
 - Normal data processing
 - User exit (ENDKEY) from all entry points
 - Unintended exit, such as power outage or Progress error

PR-PT2-530

Development Objectives

02-017 - 02-020

Starting a Transaction

- Only one transaction per session at any time
- Any change to the database
 - Create, Delete, Update, Assign, Insert
- Use of the TRANSACTION keyword in a block header (DO, FOR EACH, or REPEAT)
- Invoking an EXCLUSIVE-LOCK on a FIND or GET
- Don't start transactions unnecessarily:
 - All changes, including variables, are written to disk once a transaction is started
 - All incomplete changes are backed out on start-up or intentional exit

PR-PT2-540

Starting a Transaction

Cannot do multiple simultaneous transactions for a given session.

Avoid EXCLUSIVE-LOCK on FIND statements if no update is taking place.

02-021

TRANSACTION Function

- Returns a logical value that indicates whether a transaction is currently active
- Example:

```
IF TRANSACTION  
  THEN MESSAGE = "TRANSACTION ACTIVE."
```
- Often usually used as a debugging method to establish where a transaction occurs in extended segments of code

PR-PT2-550

TRANSACTION Function

Used in addition to the LISTING function to determine transaction scope.

Transaction Scope

- Transactions are scoped to:
 - FOR EACH blocks that update the database
 - REPEAT or REPEAT FOR blocks that update the database
 - DO TRANSACTION or other blocks using the TRANSACTION keyword
 - DO ON ERROR or DO ON ENDKEY blocks
- If none of these blocks are present and a change to the database is encountered, scope is raised to the entire procedure

PR-PT2-560

Transaction Scope

Development Objectives II

- Does the program behave identically (commit or rollback data) under all conditions?
- Does the program scope the transaction as required by the business case?
 1. Commit or rollback parent record as a unit, and commit or rollback all child records as a separate unit?
 2. Commit or rollback parent and children as a single unit?

PR-PT2-570

Development Objectives II

02-022 - 02-026

Transaction Scope

- Generally, allow PROGRESS to start and stop transactions
- Raise the transaction scope to ensure that all related database updates are backed out on system failure
- Decrease the transaction scope to ensure that only the smallest possible record update is backed out on system failure

PR-PT2-580

Transaction Scope

Depends on application. MRP is very small in scope, as each planned order generated is one transaction. If MRP is terminated, the planned orders remain.

If an Invoice is Posting, and the invoice post is terminated, the entire post is undone, indicating very large transaction scope.

The type of application drives the behavior of the transaction scope.

Subtransactions

- A subtransaction is started when a transaction is already active and PROGRESS encounters a *subtransaction* block:
 - each iteration of a FOR EACH block nested within a transaction block
 - each iteration of a REPEAT block nested within a transaction block
 - each iteration of a DO ON ERROR, or DO ON ENDKEY block
 - a procedure block that is run from a transaction block in another procedure

PR-PT2-590

Subtransactions

02-027 - 02-028

Subtransactions

- Each subtransaction close gets an end note in the local before image file (LBI)
- Purpose is to allow for incremental control of modifications through undo and error processing and system failures
 - All .LBI changes without end notes are backed out on ENDKEY, ERROR, CTRL-BREAK (failures)
- System failures, and correctly structured transactions, will back out entire transaction and all complete and incomplete subtransactions

PR-PT2-600

Subtransactions, cont.

Allow a smaller scope of control within a larger transaction.

On local terminations (Ctrl-C), the subtransaction is undone. To ensure Data Integrity, the main transaction should then have error processing to continue.

Record Locking

- Record locks apply to individual records
- Three types of locks: NO-LOCK, SHARE-LOCK & EXCLUSIVE-LOCK
- Five lock states (view in PROMON):
 - Locked: record exclusively locked by local process
 - Limbo: record released, but transaction remains open. The record lock is not released until transaction end.
 - Purged: record released and lock freed
 - Queued: record lock requested for a record already held by another process
 - Upgrade: lock upgrade requested (from SHARE to E-LOCK) for a record held by another process

PR-PT2-610

Record Locking

Record Locking

- **NO-LOCK**
 - Bypasses the normal locking mechanism
 - allows record reading regardless of lock status
 - allows record reading without locking records
- **SHARE-LOCK**
 - Whenever PROGRESS reads a record (the default lock status), it puts a SHARE-LOCK on that record. Other users can read the record but cannot update it until the procedure releases the SHARE-LOCK.
- **EXCLUSIVE-LOCK**
 - Whenever PROGRESS updates a record, it puts an EXCLUSIVE-LOCK on that record. Other users cannot read or update that record until the procedure releases the EXCLUSIVE-LOCK (except NO-LOCK reads).

PR-PT2-620

Record Locking, cont.

Record Contention

- Occurs when more than one user attempts to update a record at the same time
- Transaction scope is the primary determinant of lock duration
- Reduce record contention by:
 - Run batch updates during off hours (for file-intense applications)
 - Training: don't leave transactions open
 - Coding: don't leave transactions open

PR-PT2-630

Record Contention

02-029 - 02-031

Releasing Locks

Lock Acquired ➡	During Trans	Outside Trans	Outside but held going into Trans
SHARE	Held until trans. end or record release, whichever is later.	Held until record release	Held until trans. end or record release, whichever is later.
EXCLUSIVE	Held until trans. end; converted to SHARE if record scope is larger than transaction.	N/A	N/A

PR-PT2-640

Releasing Locks

Record Scope

- A record is read from the database and stored in a record buffer
- Scope is the portion of code during which the record is active
- Record scope controls:
 - When to write a record to the database
 - When to release a record for other users
 - also depends on type of lock and transaction scope
 - When to validate a record
 - When to reinitialize the index cursor
 - initialized upon entry to procedure block where record is scoped

PR-PT2-650

Record Scope

RECORD SCOPE: The portion of time that a record is stored and active in the record buffer. Determines how long the record is in the buffer, when it is written to the database, and how long a record lock is in effect.

Programming Impact

- Important for its impact on transaction scope
- Also a source of programming compile and runtime errors
- Progress manages memory resources using record scope
- Communicates with the transaction process to upgrade record locks as necessary

PR-PT2-660

Programming Impact

Types of Record Scope

- Four types of record scope:
 1. Strong-scoped reference
 - Buffer is available only in the block where the record is scoped
 2. Medium-scoped reference
 - Buffer available for future finds, but empty, at end of scope
 3. Weak-scoped reference
 - Buffer available, and record in buffer available, at end of scope
 4. Free reference - all other buffer references (FIND, GET etc.)
 - Scopes buffer to the procedure block and raises the scope of weak and medium scoped records where they occur together

PR-PT2-670

Types of Record Scope

Scope Reference

Can you FIND a record after?
Can you DISPLAY a record after?
Which block is record scoped to?

strong	medium	weak
NO	YES	YES
NO	NO	YES
Strong	Medium*	Weak*

* Scope raised to enclosing block on free references

PR-PT2-680

Scope Reference

Block Scope Reference

STRONG:	DO FOR <buffer>: REPEAT FOR <buffer>:
MEDIUM:	REPEAT: REPEAT ON ERROR, ENDKEY: FOR EACH:
WEAK:	REPEAT TRANSACTION: FOR FIRST/LAST:
NONE (FREE):	DO: DO TRANSACTION: DO ON ERROR, ENDKEY:

PR-PT2-690

Block Scope Reference

02-032 - 02-033

Strong Scoped Records

- Strong-scoped reference to a buffer
 - buffer is always scoped to the strong-scoped block
 - two sequential strong-scoped blocks can reference the same buffer
 - cannot reference that buffer in a containing block
 - illegal references to fields return compile error:
**** Missing FOR, FIND or CREATE for a table with cm_addr in current block. (232) "*
 - illegal references to a table return compile error:
**** Reference to table cm_mstr conflicts with block statement reference. (244) "*

PR-PT2-700

Strong Scoped Records

02-034 - 02-036

Medium Scoped Records

- Medium-scoped reference to a buffer
 - buffer is scoped to the block, unless raised by a free reference to the enclosing block
 - cannot redisplay buffer outside the medium-scope block
 - cannot nest two medium-scope blocks for the same buffer
 - "*** Illegal nested block statement reference to table cm_mstr. (243)"
 - illegal DISPLAY of record outside the block returns the runtime error:
 - "***No <tablename> record is available (91)."
 - can perform a find on the buffer table

PR-PT2-710

Medium Scoped Records

02-037 - 02-039

Weak Scoped Records

- Weak-scoped reference to a buffer
 - buffer is scoped to the block, unless raised by a free reference to the enclosing block
 - can display or find buffer outside the weak-scope block
 - cannot nest two weak-scope blocks for the same buffer
 - weak-scope block cannot contain any free references to the same buffer (that is, no FINDs allowed in a FOR EACH)

PR-PT2-720

Weak Scoped Records

02-040

Free References

- PROGRESS tries to scope buffer to the nearest enclosing block with record scoping properties
- Free references are:
 - FIND
 - GET
 - CREATE
 - INSERT

PR-PT2-730

Free References

Predicates, Indexes, and Field Lists

Predicates, Indexes & Field Lists

	F1	F2	F3	F4	F5
R1				x	
R2		x			
R3		x		x	
R4		x		x	
R5		x			

Predicates:
"WHERE F2 AND F4..."
returns ENTIRE R3 & R4.
Value-dependent,
selects index used.

	F1	F2	F3	F4	F5
R1		1		R	
R2		2			
R3		22		S	
R4		3		Z	
R5		31			

Indexes:
"USE-INDEX F2-F4..."
return all records
sorted by F2 and F4.

	F1	F2	F3	F4	F5
R1				x	
R2		x			
R3		x		x	
R4		x		x	
R5		x			

Field Lists:
"F2, F4."
returns two columns
for all records.

Combine to select records by value, sort records, and select relevant columns.

PR-PT2-750

Predicates, Indexes, and Field Lists

- Predicate: Where clause. Returns entire record selected.
- Index: When specified using the USE-INDEX keyword, indicates sort, not just efficient lookup
- Field Lists: Only returns the list of fields specified to the record buffer

WHERE Expression

- Qualifies the records you want to access
- The expression is a constant, field name, variable name, or expression whose value you want to use to select records
- Most efficient when the expression uses an index
 - equality matches
 - single range matches
 - the data path (including the index) is written into the r-code at compile time

```
FIND FIRST pt_mstr WHERE pt_part > "100" NO-ERROR.
```

PR-PT2-760

WHERE Expression

The expression (where clause) should bracket an index whenever possible. It should use indexed fields for efficient search.

Equality, Range, and Sort Matches

- Equality match

WHERE field = *expression*

- Range match

WHERE field < | <= | > | >= *expression*

WHERE field BEGINS *expression*

- Sort match

WHERE fieldX = expression BY *fieldY*

PR-PT2-770

Equality, Range, and Sort Matches

- Equality: Where field = value or expression
- Range: Where field is greater than, less than, greater than or equal to, less than or equal to a value
- Begins: The value is present at the beginning of the string
- Sort Match: Select based on X criteria, but sort using field Y via the BY word

Operators—Logical

Operator	Description
NOT	returns a TRUE value if an expression is false, and FALSE if an expression is true
AND	returns a TRUE value if all expressions are true, otherwise FALSE
OR	returns a TRUE value if at least one value is true, otherwise FALSE

PR-PT2-780

Operators—Logical

02-041

Operators—Comparison

Name	Operator
less than	< or LT
less than or equal to	<= or LE
greater than	> or GT
greater than or equal to	>= or GE
equal to	= or EQ
not equal to	<> or NE

PR-PT2-790

Operators—Comparison

BEGINS Operator

- Tests a character expression to see if that expression begins with a second character expression
- May search on an index if the field is the leading component of an index.

- Example:

```
FOR EACH cm_mstr WHERE cm_sort BEGINS "jo":  
    DISPLAY cm_sort.  
END.
```

- NOTE: The query will find "Johnson" and "John Smith." The query will not find "J Johnson & CO."

PR-PT2-800

BEGINS Operator

MATCHES Operator

- Compares a character expression to a pattern and evaluates to TRUE if the expression satisfies the pattern criteria
- Performs a sequential search, will not bracket from an index
- Example:

```
FOR EACH cm_mstr WHERE cm_sort MATCHES "jo*":  
    DISPLAY cm_sort.  
END.
```
- Use wildcard characters for pattern matching
 - asterisk (*) = any one or more characters
 - period (.) = one character only

PR-PT2-810

MATCHES Operator

02-042

CONTAINS Operator

- Checks whether the supplied string is contained anywhere in the field reference when the field is defined as a word index
- Requires a word index

```
FOR EACH cm_mstr WHERE cm_sort CONTAINS "John Doe"  
    DISPLAY cm_sort.  
END.
```

- The space is treated as a logical AND. The search will find "John Doe and Company," "Doe & John," and "John Franklin Doe." The search will not find "Johnson and Doe Inc.," "Doe & Company," or "John Acker CEO."

PR-PT2-820

CONTAINS Operators

Sorting with **BY** Option

- Syntax:

```
WHERE searchExp BY field1 [DESCENDING]
```

- Sorts the selected records using *field1*
- Without the BY option, records retrieved in the order of the index used to satisfy the *searchExp* criteria
- If the BY field is not indexed, a temporary sort table is generated after selection to sort and then retrieve the records; could be a performance hit

- Example:

```
FOR EACH cm_mstr BY cm_cr_limit BY cm_sort:
```

PR-PT2-830

Sorting with **BY** Option

02-043

USE-INDEX Option

- Identifies the index to use while selecting records
- Overrides specified in the WHERE, USING, OF, or constant options.

```
FIND FIRST pt_mstr USE-INDEX i_pt_desc.  
REPEAT WHILE AVAILABLE pt_mstr:  
    DISPLAY pt_part pt_desc1.  
    FIND NEXT pt_mstr.  
END.
```

PR-PT2-840

USE-INDEX Option

Using the index of I_pt_desc, find the first pt_mstr.

This indicates that it might not be the first part, but the first description alphabetically. Then, since the FIND NEXT does not specify any index, the primary index is invoked for the remainder of the procedure. So the first pt_mstr using the description will be found. Where ever that part number is within the pt_mstr, the remainder of the pt_mstr will then be displayed.

Indexing

- Indexing greatly impacts performance
- Top reasons of poor performance
 - incorrect index design
 - index design okay but correct index not used by code with use-index
 - the BY option does not match the index sort

PR-PT2-850

Indexing

Simple WHERE Clause

```
WHERE searchExp [BY field1]
```

- If there is an index on the searchExp field, or if the field is first in a compound index, uses that index
- Else: Primary index
- If searchExp references a word index, uses that
- BY clause uses an index for *field1* if available
- Else: temporary sort table built

PR-PT2-860

Simple WHERE Clause

Compound WHERE Clause 1

WHERE searchExp AND searchExp [BY field1]

- Progress evaluates both sides of compound clauses
- On FOR EACH, if both sides are indexed, or are leading components of compound indexes, uses both
- Else: whichever index qualifies and primary index
- FINDs use only first qualifying index, else primary
- Progress does not guarantee sort order for multiple index retrievals
 - Use the BY option, or the USE-INDEX option to guarantee sort order

PR-PT2-870

Compound WHERE Clause 1

Compound WHERE Clause 2

WHERE searchExp OR searchExp [BY field1]

- Progress evaluates both sides of compound clauses
- If both sides are indexed, or are leading components of compound indexes, uses both
- Else: whichever index qualifies and primary index

PR-PT2-880

Compound WHERE Clause 2

Choosing a Single Index

- Use the word index if CONTAINS phrase appears
- Unique index with all of its components used in active equality matches
- Index with the most active* equality matches
- Index with the most active range matches
- Index with the most sort matches
- Index that comes first alphabetically
- Primary index

* **active matches are stand-alone or leading index components, and if joined, are joined by AND**

PR-PT2-890

Choosing a Single Index

This is the order in which Progress will select an index for any lookup if the index is not mandated via the USE-INDEX option.

XREF Output

```
1 COMPILE d:\v7\v7ctrain\code\x.p
1 CPINTERNAL iso8859-1
1 CPSTREAM ibm850
3 STRING "pt_mstr" 8 NONE UNTRANSLATABLE
3 SEARCH train1.pt_mstr i_pt_part WHOLE-INDEX
4 ACCESS train1. pt_mstr pt_part
4 STRING ">>9" 3 NONE TRANSLATABLE  FORMAT
5 STRING "Part Number" 15 LEFT TRANSLATABLE
5 STRING "pt_part" 8 LEFT TRANSLATABLE
5 STRING "-----" 17 NONE UNTRANSLATABLE
5 STRING "i_pt_part" 10 NONE UNTRANSLATABLE
```

PR-PT2-900

XREF Output

02-044

Word Index

```
FOR EACH so_mstr WHERE so_rmks CONTAINS "rush"  
    AND so_nbr = "3000":  
    DISPLAY so_nbr so_due_date so_rmks.  
END.
```

- Uses index so_rmks

PR-PT2-910

Word Index

02-045

Unique Index - Equality Matches

```
FOR EACH pt_mstr WHERE pt_critical = NO  
    AND pt_group = "G1"  
    AND pt_status = "":  
    DISPLAY pt_group pt_status pt_price.  
END.
```

Uses index i_pt_critical

- Equality matches on all components
- Used on *type*, *status*, *group* fields

PR-PT2-920

Unique Index - Equality Matches

02-046

Most Active Equality Matches

```
FOR EACH pt_mstr WHERE pt_group = "G1" AND  
    pt_part >= "100" AND pt_price > 1.00:  
    DISPLAY pt_group pt_status pt_price.  
END.
```

Uses index i_pt_group

- pt_group has an active equality match, pt_part has an active range match

PR-PT2-930

Most Active Equality Matches

02-047

Most Active Range Matches

```
FOR EACH pt_mstr WHERE pt_desc1 >= "a"  
    AND pt_group MATCHES "G*":  
    DISPLAY pt_part pt_group pt_price.  
END.
```

- Uses index i_pt_desc
 - the MATCHES operator never brackets an index
 - the field pt_desc1 is the only active range match

PR-PT2-940

Most Active Range Matches

02-048

Most Active Sort Matches

```
FOR EACH pt_mstr WHERE pt_qoh > 100 BY pt_part:  
  DISPLAY pt_desc1 pt_part pt_group pt_qoh.  
END.
```

- Uses index i_pt_part
 - the field pt_part is the sort index
 - pt_qoh is not indexed

Note: all sort matches are active

PR-PT2-950

Most Active Sort Matches

02-049

Alphabetic Index Selection

```
FOR EACH pt_mstr WHERE pt_part > "10"  
  AND pt_desc1 BEGINS "a":  
  DISPLAY pt_part pt_desc1.  
END.
```

- Uses index i_pt_desc
 - a tie: the two indexes have the same number of active range matches
 - use the alphabet to pick the index. i_pt_desc comes before i_pt_part alphabetically

PR-PT2-960

Alphabetic Index Selection

02-050 - 02-051

Primary Index

```
FOR EACH pt_mstr WHERE pt_price > 100.00
    AND pt_qoh > 100:
    DISPLAY pt_part pt_price pt_qoh.
END.
```

- Uses primary index, i_pt_part
 - neither search field indexed
 - selects using primary index
 - may use WHOLE-INDEX

PR-PT2-970

Primary Index

02-052

PRESELECT Phrase

- Used instead of FOR, PRESELECT builds a complete results list for all requested records
 - Use this to identify immediately records to be returned, or
 - To immediately lock all records returned
- Initial performance cost, rather than ongoing

PR-PT2-980

PRESELECT Phase

FIND is then used on the records that are part of the PRESELECT

Bracketing

- Once one or more indexes are selected, Progress attempts to return the fewest records that satisfy
- Rules for bracketing – examine each index component:
 - Bracket on active equality matches
 - Bracket an active range match, but no more after that

Unbracketed search:

```
SEARCH reachv9.pt_mstr i_pt_part WHOLE-INDEX
```



PR-PT2-990

Bracketing

Some Indexing Guidelines

- As number of records that satisfy a query to total records increases, desirability of index access decreases
- Implement fewer indexes on tables with heavy update activity and limited access, and vice versa
- Most index processing occurs on server side, returning the minimum required records to the client
- To avoid:
 - Joining range matches with AND
 - Non-indexed ORs
 - Multiple wild card strings in word indexes
 - Word indexes OR non-indexed criterion

PR-PT2-1000

Indexing Guidelines

Field Lists

- A *field list* is a subset of the fields that define a record and includes those fields that the client actually requires from the database server
 - optimizes preselected/presorted fetches from the database server
 - may be used with many types of record fetches
 - FOR Statements
 - queries
 - PRESELECT Statements
 - SQL SELECT Statements
- Syntax

```
record-bufname [FIELDS[([field ...])] | EXCEPT[([field ...])]]
```

PR-PT2-1010

Field Lists

Field Lists

- File: Customer
 - Fields: cm_addr cm_balance cm_class cm_cr_hold
cm_cr__limit cm_cr_rating cm_cr_terms cm_sort
cm_xslspn

```
FOR EACH cm_mstr  
NO-LOCK:
```

=

```
FOR EACH cm_mstr  
  FIELDS(cm_addr cm_balance  
         cm_class  
         cm_cr_hold  
         cm_cr_limit  
         cm_cr_rating  
         cm_cr_terms  
         cm_sort  
         cm_xslspn)  
NO-LOCK:
```

PR-PT2-1020

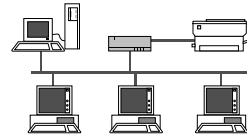
Field Lists, cont.

Can be deactivated using the `-fldisable` parameter. Progress will then retrieve the whole record, overriding the field list in the procedure.

02-053 - 02-056

Field List Benefits

- Fetching over a network
 - reduction in network traffic
- Fetching from local DataServers
 - benefits from reading only a portion of the record
- Fetching from remote DataServers that send multiple rows at a time
 - DataServers that package multiple rows per network message benefit from field list fetches
- Compiling using DataServers for foreign databases
 - fetch commands store foreign database commands as ASCII text in the r-code. Without field lists, the list of field information can be quite large, thus increasing the size of the r-code



PR-PT2-1030

Field List Benefits

MUST call out every field that is required for update or display.

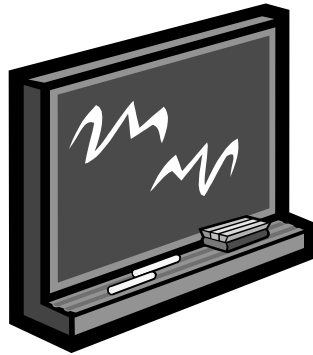
LAB 4

1. Allow users to update customer records. Make sure appropriate updates occur to the associated ad_mstr records, using the Class File Relationships slide. (L0401.p)
2. Allow users to update or create new customer records. Make the appropriate assignments to ad_mstr. (L0402.p)
3. Display the sales order detail by customer and order. Display the following fields: sod_line, sod_qty_item, sod_price, and sod_price multiplied by sod_qty_item. (L0403.p)

PR-PT2-1040

Lab 4

Lab 4 Review



PR-PT2-1050

Lab 4 Review

Variables, Messages, and Conversions

DEFINE VARIABLE Statement

- Temporary work space to compute and store data
- Contains user data, copies of db fields, results of expressions

```
DEFINE VARIABLE variable {AS datatype}|{LIKE field}  
{[NO-UNDO] [FORMAT string] [LABEL string]}. 
```

- Can be shared globally or hierarchically within a call stack

PR-PT3-020

DEFINE VARIABLE Statement

SHARED or GLOBAL

If a variable is defined LIKE a database field, then it takes on the same attributes as database field.

If format is not specified, the default format for the field type is the defined format.

NO-UNDO

- Retains a variable value beyond transaction scope by writing the original value to the BI file
- Intentional data back-out retains variable value
- NO-UNDO variables are more efficient; use this option whenever possible
- If you are doing extensive calculations with variables, and you do not need to take advantage of undo processing for those variables, use the NO-UNDO option when defining the variables

PR-PT3-030

NO-UNDO

Used when value does not need to be undone if a particular block is terminated.

02-057 - 02-058

Screen Input: INPUT

- INPUT Option
 - references the value of a field or variable in the screen buffer
 - PROMPT-FOR stores user entries in window buffer; INPUT function can look-up, display or assign the values there
- Like USING option, but not limited to database fields

```
DEFINE VARIABLE x LIKE pt_part LABEL "Part Number".  
PROMPT-FOR x.  
FIND pt_mstr WHERE pt_part = INPUT x NO-ERROR.
```

PR-PT3-040

Screen Input: INPUT

Uses the value of the input buffer for the defined variable.

STATUS Statement

- Specifies the text that appears in the status line of a window
- Syntax

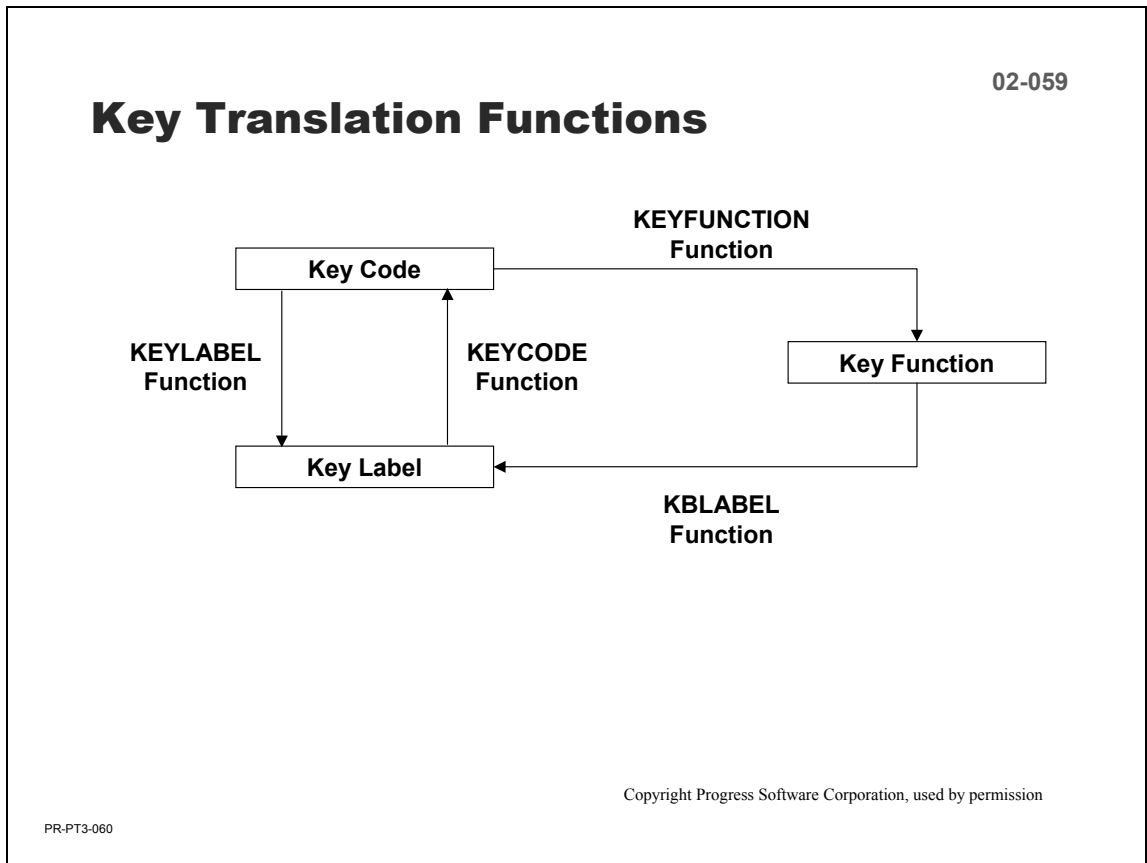
```
STATUS{{DEFAULT[expression]}|{INPUT[OFF|expression]}}
```

 - Limit of 63 characters
- Reset STATUS at the end of the code requiring the change

PR-PT3-050

STATUS Statement

STATUS appears frequently in MFG/PRO code. It is used to display the function key options at the bottom of the screen. This does not control how these keys operate, but lists what keys are active for what purpose.



Key Translation Functions

PROGRESS can port across platforms because of the ability to translate key definitions. For example, GO is F1 in a character environment, CTRL-X in some platforms and F2 in GUI/Windows. How these keys translate can be built into the code for easy error or exit processing. **READKEY:** Reads one keystroke at a time and sets the value of LASTKEY to the KeyCode of that keystroke.

LASTKEY: Returns the integer value (keycode) of the most recent user event (keystroke)

Message Statement

- Displays messages in the message area at the bottom of the window or in an alert box
- Can include simple SET or UPDATE
- Procedural checkpoint
- Basic syntax:

```
MESSAGE {expression|{SKIP [(n)]}}... [VIEW-AS  
  ALERT-BOX [alert-type] [BUTTONS button-set]  
  [TITLE title-string]] [{SET|UPDATE} field]
```

- Example:

```
DEFINE VARIABLE x AS LOGICAL.  
MESSAGE "Add Part?" UPDATE x.
```

PR-PT3-070

Message Statement

02-060 - 02-061

ALERT-BOX Phrase

- Graphical message box with buttons
- INFO, QUESTION, WARNING, ERROR types
- YES-NO, YES-NO-CANCEL, OK, OK-CANCEL, RETRY-CANCEL button sets
- Alert-box messages can only update logical variables

MESSAGE

"Update item?"

VIEW-AS ALERT-BOX QUESTION BUTTON YES-NO-CANCEL

UPDATE x AS LOGICAL.

PR-PT3-080

ALERT-BOX Phrase

STRING, SUBSTRING Functions

- STRING converts a value of any data type into a character value
- Syntax:
`STRING(source[,format])`
- SUBSTRING extracts a portion of a character string from a field or variable
- Syntax:
`SUBSTRING (source,position[,length[,type]])`

PR-PT3-090

STRING, SUBSTRING Functions

TRIM Function

- Removes leading and trailing white space, or other specified characters, from a character string.

- Syntax:

```
TRIM(string[,trim-chars])
```

PR-PT3-100

TRIM Function

02-065

CASE Statement

- Provides a multi-way decision based on the value of a single expression
- Put the most likely matches first
 - On the first branch where the *value* matches the *expression*, the associated *statement* is executed

```
CASE expression:  
    {WHEN value [OR WHEN value]... THEN {statement}}  
    [OTHERWISE {statement}]  
END [CASE].
```

- Added in Version 8

PR-PT3-120

CASE Statement

IF...THEN...ELSE Function

- Evaluates to one of two expressions, depending on the value of a third conditional expression

- Syntax:

```
IF conditional-expression  
  THEN expression1 ELSE expression2
```

- Example:

```
ASSIGN  
  msg = IF x > 5 THEN "High" ELSE "Low".
```

PR-PT3-130

IF...THEN...ELSE Function

Editing Phrase

- Allows the application to manage the input one key stroke at a time during a PROMPT-FOR, SET, or UPDATE statement
- This block executes once for each key stroke received during execution of the input statement
- Basic syntax

```
UPDATE field1 field2... EDITING:  
    READKEY.  
    APPLY LASTKEY.  
END.
```

PR-PT3-140

Editing Phrase

EDITING is used when there needs to be a protection against specific characters within an input field. For example, in User Maintenance, if a special character is restricted, there could be logic to prevent a special character by disallowing entry of that character key stroke.

02-066 - 02-067

FRAME-FIELD, FRAME-VALUE

- FRAME-FIELD returns the name of the input field the cursor is in during data entry; otherwise returns the name of the input field the cursor was last in
- FRAME-VALUE returns the (character string) value of the input field that the cursor is in to the current input field; otherwise returns the (character string) value of the input field the cursor was last in
 - Can be assigned

PR-PT3-150

FRAME-FIELD, FRAME-VALUE

Used along with EDITING. Identifies what value is in the screen buffer where the cursor resides. For example, if you begin to type a part number and then press F2 help. The browse recognizes what was already input and begins the browse as of that substring.

CHAPTER 7

Validation and Help

Validation Overview

- Validation is method of ensuring data integrity
- Most effective on the schema level
- Improved schema validation via triggers unavailable in earlier Progress versions - covered later in course
- Field validation only in procedural model

PR-PT3-170

Validation Overview

Field-Level Validation

- Schema-defined
 - works in all programs that reference the field
- Session-defined
 - overrides the schema-defined validation
 - specified in the field's *format-phrase*
 - can also use to “turn off” schema defined validation
 - VALIDATE Statement and Method()
- Bound in the r-code at compile time
- For complex validations use schema or session triggers

PR-PT3-180

Field-Level Validation

SCHEMA-Defined: Actually resides within the data dictionary on the field. The .v program is bound into the r-code at compile time for any procedure that references that field.

Session-defined Validation

- Specifies a condition which must be true for user input to be accepted
- Specifies error message to be displayed if not
- Validation occurs when the field is modified
 - if the field is not modified, validation occurs on the GO event
- Example

```
UPDATE
```

```
  X
```

```
  Y VALIDATE (Y > 0, "Must be greater than zero")
```

```
  Z.
```

PR-PT3-190

Session-Defined Validation

VALIDATE statement forces the condition to be met. Additionally, it can 'grand-father' in old records that should not be subject to new validations. For example, if a new trigger forces new parts to have price greater than 0, it is possible with the validate statement to only validate for new items, letting old items to not be held to new rules.

02-068

VALIDATE Statement

- Verifies that a record complies with mandatory field and unique index definitions
- Because validation is done automatically, you rarely have to use the VALIDATE statement
- PROGRESS automatically validates a record when
 - a record in the record buffer is replaced by another
 - a record's scope iterates or ends
 - the innermost iterating subtransaction block that creates a record iterates, or
 - a transaction ends

PR-PT3-200

VALIDATE Statement

VALIDATE() Method

- Executes any validation tests established in a database or by VALIDATE option of the format phrase
- Executes validation tests for every supported field-level object in the frame or dialog
- Logical; returns TRUE if successful
- On failure, validation message displays and focus moves to first failed screen object

PR-PT3-210

VALIDATE() Method

CAN-FIND Function

- Returns TRUE if a record is found that meets the specified FIND criteria; otherwise returns FALSE
- Use to see if a record exists with less system overhead than a FIND – does not retrieve record

- Basic syntax

```
CAN-FIND([FIRST|LAST] record [WHERE expression]
[USE-INDEX index] [SHARE-LOCK|NO-LOCK] [NO-WAIT])
```

- Example

```
IF CAN-FIND(cm_mstr WHERE cm_addr = "1") THEN
  MESSAGE "Record exists."
```

PR-PT3-220

CAN-FIND Function

Does not actually pull record into the record buffer. Merely a way to test for existence of the record.

02-069 - 02-070

CAN-DO Function

- Returns TRUE if a character value exists in a comma-delimited list; otherwise returns FALSE
- Use the CAN-DO function instead of stringing many OR conditions together
- Example

```
IF CAN-DO ("A,B,C,D",x) THEN MESSAGE "Value is in set".
```

PR-PT3-230

CAN-DO Function

If the current value of 'x' is present in the list of arguments (A, B, C and D), then TRUE is returned.

Field-Level Help

- If you define field-level help in the schema, the help message appears in all programs that reference the field
- To override the schema field-level help, you may define field-level help in the *format-phrase* for the particular field
- PROGRESS binds the field-level help into the r-code at compile time

PR-PT3-240

Field-Level Help

Field-Level Help is hard-coded if not applied at the schema level. This is not the same as traditional FIELD HELP or PROCEDURE HELP. Field-Level Help displays at the bottom of a screen while the field the help applies to is in focus.

Use the HELP statement to hardcode.

Field-Level Help

- Limited to 63 characters
 - truncated at runtime
- If input device is not terminal, HELP strings ignored
- Can also specify NO-HELP to override Data Dictionary and FORMAT phrase help with no help.

PR-PT3-250

Field-Level Help, cont.

02-071 - 02-072

Field-Level Help

- Field-level help must be a constant
 - expressions provided as field-level help do not compile
- Example:

```
/* This program does not compile */  
FOR EACH cm_mstr:  
  UPDATE  
    cm_addr HELP "Enter Address Code for " +  
    cm_sort.  
END.
```



PR-PT3-260

Field-Level Help, cont.

APPLHELP.P

- A centralized means to implement *context-sensitive help*
- One per application
- PROGRESS uses the PROPATH to find it
- Does not allow parameters, but it has access to all the default PROGRESS parameters [FRAME-FIELD, FRAME-DB ...]

PR-PT3-270

APPLHELP.P

Along with gpaplhlp.p (in MFG/PRO), APPLHELP.P controls the use of help via the F2 or F6 (user menu) functions.

applhelp.p

Manipulating the **PROPATH**

- You can change your **PROPATH** on the fly
- PROGRESS searches the PROPATH from left to right to find a program to run
- Changing the PROPATH on the fly speeds up operating system searches
- PROPATH is a character string
- All string functions may be used to manipulate the PROPATH
- Example:

```
UPDATE x.  
IF x = "GL" THEN PROPATH = "/usr/gl".  
ELSE IF x = "AR" THEN PROPATH = "/usr/ar".
```

PR-PT3-280

Manipulating the **PROPATH**

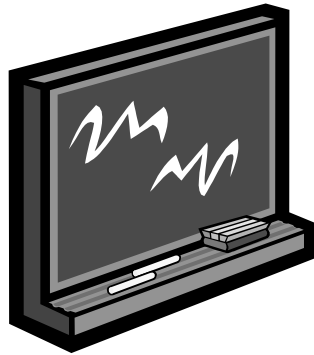
LAB 5

1. Update the part master using a variable to find records. Run the update from a user's response to a message statement. (L0501.p)
2. Add a message to allow user to create a new part record. (L0502.p)
3. Add help to pt_desc2 and validation for pt_group. (L0503.p) Valid values for pt_group are G1 and G2.

PR-PT3-290

Lab 5

Lab 5 Review



PR-PT3-300

Lab 5 Review

CHAPTER 8

Frames

Frames in PROGRESS

- A rectangular display area within a window that PROGRESS uses to display field-level widgets and other frames
- Allocated to hold field-level widgets referenced by screen display and input statements
- Allocation is determined by the compiler's top-to-bottom pass of a procedure
 - frames are allocated to blocks that display widgets
 - REPEAT blocks
 - FOR EACH blocks
 - DO WITH FRAME blocks
 - Procedure blocks

PR-PT3-320

Frames in PROGRESS

Frame Characteristics

- Allocation:
 - The display of field-level objects (widgets)
- Scope:
 - The display of multiple records or value sets
- Appearance:
 - Use of the frame-phrase to manage size, layout, color, and placement
- Processing:
 - Controlling "down" attributes, reusing frames, scrolling in frames

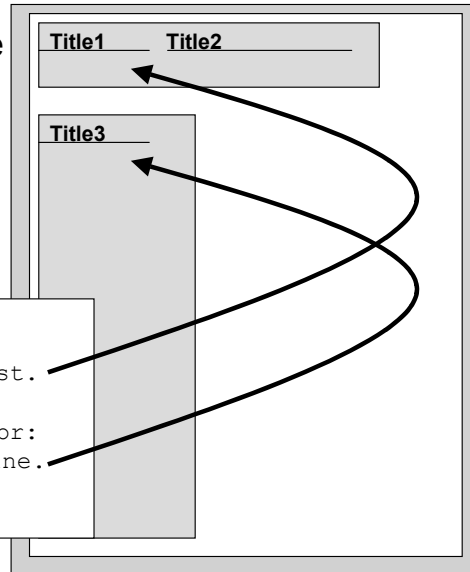
PR-PT3-330

Frame Characteristics

Allocation: Top-Down Compile

- One pass:
 - screen real estate
 - data processing
- Top-to-bottom
- Left-to-right

```
FOR EACH so_mstr:  
  DISPLAY so_nbr so_cust.  
  FOR EACH sod_det  
    WHERE sod_nbr = so_nbr:  
      DISPLAY so_line.  
  END.  
END.
```



PR-PT3-340

Allocation: Top-Down Compile

Default Frame Appearance

- Basic syntax

```
WITH [SIDE-LABELS] [n COLUMNS] [FRAME frame].
```

- Examples

```
INSERT pt_mstr.
```

```
INSERT pt_mstr WITH SIDE-LABELS.
```

```
INSERT pt_mstr WITH 2 COLUMNS.
```

PR-PT3-350

Default Frame Appearance

Appearance: Frame-Phrase

- `INSERT pt_mstr`
 - column labels
 - all fields are displayed
 - fields displayed by Data Dictionary “field order number”
 - if no field label is defined, field name is used
 - PROGRESS subscripts the arrays

PR-PT3-360

Appearance: Frame-Phrase

02-073

Appearance: Frame-Phrase

- `INSERT pt_mstr WITH SIDE-LABELS`
 - labels display on the left side of the field
 - all else is the same

PR-PT3-370

Appearance: Frame-Phrase, cont.

02-074

Appearance: Frame-Phrase

- `INSERT pt_mstr WITH 2 COLUMNS.`
 - formats data fields into a specific number of columns
 - implies SIDE-LABELS
 - labels are right-justified
 - labels are truncated
 - 1 COLUMNS, max label length is 16
 - 2 COLUMNS, max label length is 14
 - 3 COLUMNS, max label length is 12

PR-PT3-380

Appearance: Frame-Phrase, cont.

Frame Scope

- All REPEAT and FOR EACH blocks scope frames by default
- Innermost iterating block gets a "down frame"
- Frames are scoped to one block only
- Frame scope is not dynamic: memory is not freed when scope is complete

PR-PT3-400

Frame Scope

02-075

Frame Scope

```
DISPLAY "Hello".
```

```
FOR EACH cm_mstr:  
  DISPLAY cm_addr cm_sort.  
  LEAVE.  
END.
```

} Block 2

} Block 1

```
PAUSE.  
DISPLAY "I'm the PROCEDURE block 1".
```

- Block 1 = Procedure block
- Block 2 = FOR EACH block

PR-PT3-410

Frame Scope, cont.

02-076

Frame Scope



Frame 1 is the default frame scoped to the outer FOR EACH block.

If named, you can also access this frame from the inner FOR EACH block.

Frame 2 is the default frame scoped to the inner FOR EACH block.

You cannot access this frame from the outer FOR EACH block without raising its scope.

PR-PT3-420

Frame Scope, cont.

Determining Down Frames

Is the block an iterating block?	YES
Is the default frame scoped to the block?	YES
Is it the innermost block (no inside block?)	YES
<hr/>	
Then the frame type is:	DOWN
<hr/>	
All other default combinations:	ONE-DOWN

Copyright Progress Software Corporation, used by permission

PR-PT3-430

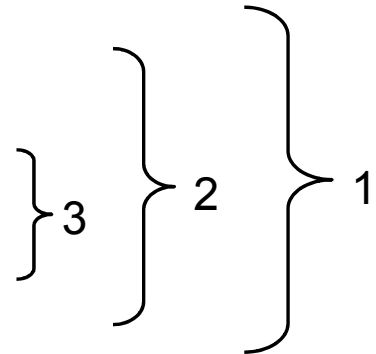
Determining Down Frames

02-077

Frame Scope

```
DISPLAY "Hello".
FOR EACH cm_mstr:
  DISPLAY cm_addr cm_sort.

  FOR EACH so_mstr
    WHERE so_cust = cm_addr:
      DISPLAY so_nbr.
  END.
END.
DISPLAY "I'm the PROCEDURE block 1".
```



- Block 1 = Procedure block
- Block 2 = FOR EACH cm_mstr block
- Block 3 = FOR EACH so_mstr block -- DOWN

PR-PT3-440

Frame Scope

You can only display in a frame where the frame is scoped.

Controlling Frames

- Using named frames
 - Managing allocation and scope
- Declaring DOWN frames
- Using FORM Statements or DEFINE FRAME
 - Names the frame and allows control over appearance and processing
- Adding processing capabilities to a frame
 - Sharing frames
 - Reusing a frame
 - Scrolling frames

PR-PT3-450

Controlling Frames

Sharing a frame is similar to sharing a variable. The frame definition is shared and accessed by multiple procedures, allowing data from multiple procedures to be displayed within the same frame.

Named Frames

- Use named frames to:
 - Reference frames in enclosed blocks (raise the scope of a frame)
 - Control frame appearance and processing
 - Override default frame characteristics

PR-PT3-460

Named Frames

Named Frames

- Four options:
 - Name the default frame (FOR EACH ... with FRAME a) to reuse the frame in an enclosed block
 - Takes on default characteristics
 - Name a new frame for use by the DISPLAY component (DISPLAY ... with FRAME a)
 - Loses default characteristics (DOWN)
 - Use the FORM statement
 - Scopes named frames to the procedure
 - Allows for extensive definition of appearance and processing
 - Use the DEFINE FRAME statement
 - Allows you to scope named frames as needed
 - Also allows extensive definition

PR-PT3-470

Named Frames, cont.

02-078 - 02-081

Named Frames

- The default frame is used unless you explicitly name a frame and use that frame
- If you name a frame it becomes the default frame and no unnamed frame is allocated to the block
- To raise the scope of a frame (include records from enclosing blocks), scope the frame to an enclosing block
- Scoping a frame to an encompassing block AFTER it has been scoped to an inner block returns a compile-time error:
 - “Invalid reference to a frame outside its scope. (231)”

PR-PT3-480

Named Frames, cont.

02-082 - 02-083

Creating Named Down Frames

- Down is allocation of lines AND data
- If a frame is scoped to the block but not the default frame of the block, make it a down frame
- If a frame is not scoped to the block, make it a down frame and use the DOWN WITH FRAME *<frame name>* statement

PR-PT3-490

Creating Named Down Frames

02-084 - 02-087

Frame Flashing

- Occurs when an iterating block displays to a frame that is scoped to an enclosing block
- Each iteration of the block overwrites the data from the previous iteration
- Solved with the DOWN WITH FRAME statement

PR-PT3-500

Frame Flashing

FORM Statement

- Defines the layout and certain processing attributes of a frame
- Scopes the frame to the current block
- Determines the display sequence
- An UPDATE statement using the FORM determines the Tab sequence (unless...)
- Basic syntax

FORM [*form-item...*] [*frame-phrase*]

PR-PT3-510

FORM Statement

DEFINE FRAME Statement

- Defines and creates a frame or dialog box that can be used within a procedure or shared among several procedures
- Does not scope the frame to a block until it is used
- Syntax
 - `DEFINE [[ NEW ] SHARED ] FRAME frame [ form-item ... ] [ DROP-TARGET ] [ { HEADER | BACKGROUND } head-item ... ] { [ frame-phrase ] }`
 - `DEFINE [ [ NEW ] SHARED ] FRAME frame record [ EXCEPT field ... ] [ DROP-TARGET ] { [ frame-phrase ] }`

PR-PT3-520

DEFINE FRAME Statement

FORMAT Phrase

- Specifies one or more attributes for a widget
- Apply to widgets in FORM or DEFINE FRAME
- Syntax

```
[at-phrase ][ AS datatype | LIKE field ][ ATTR-SPACE |
NO-ATTR-SPACE ][ AUTO-RETURN ][ BGCOLOR expression ][
BLANK ][ COLON n | TO n ][ COLUMN-LABEL label ][
DEBLANK ][ DCOLOR expression ][ DISABLE-AUTO-ZAP ][
FGCOLOR expression ][ FONT expression ][ FORMAT
expression ][ HELP string ][ LABEL label [ , label ]
... | NO-LABELS ][ NO-TAB-STOP ][ PFCOLOR expression
][ VALIDATE ( condition , msg-expression ) ][ view-as-
phrase]
```

- Example

```
DEFINE FRAME f1
var_1 AT 20 AS CHARACTER FORMAT "X(18)" HELP "Choose
carefully." LABEL "Pick One" VIEW-AS RADIO-SET RADIO-
BUTTONS "&Life", 1, "&Death", 2.
```

PR-PT3-530

FORMAT Phrase

02-088 - 02-090

Frame Phrase

- Specifies overall layout and processing properties of a frame for frame definition (DEFINE FRAME and FORM), block header (DO, FOR EACH, and REPEAT), and data handling (DISPLAY, SET, etc.) statements
- Syntax

```
WITH [ACCUM [max-length] [at-phrase] [ATTR-SPACE|NO-ATTR-
SPACE][CANCEL-BUTTON button-name] [CENTERED] [color-
specification][COLUMN expression] [n COLUMNS] [DEFAULT-BUTTON
button-name][[expression] DOWN] [EXPORT] [FONT expression]
[FRAME frame] [KEEP-TAB-ORDER] [NO-BOX] [NO-HIDE] [NO-LABELS]
[USE-DICT-EXPS] [NO-VALIDATE] [NO-HELP] [NO-UNDERLINE]
[OVERLAY][PAGE-BOTTOM|PAGE-TOP] [RETAIN n] [ROW expression]
[SCREEN-IO|STREAM-IO] [SCROLL n] [SCROLLABLE][SIDE-LABELS]
[size-phrase][STREAM stream] [THREE-D][title-phrase] [TOP-
ONLY][USE-TEXT] [V6FRAME [USE-REVVIDEO|USE-UNDERLINE]] [VIEW-
AS DIALOG-BOX] [WIDTH n] [IN WINDOW window]
```

PR-PT3-540

Frame Phrase

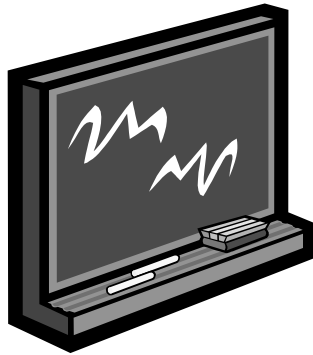
LAB 6

1. Update or create part records in a frame. Continue to use availability, validation, and help. (L0601.p)
2. Enable users to review sales orders for a part if they wish. Use a message box. (L0602.p)

PR-PT3-550

Lab 6

Lab 6 Review



PR-PT3-560

Lab 6 Review

Reporting and Input/Output

02-091 - 02-094

Aggregate Phrase

- Identifies one or more values to calculate based on a change in an expression or a break group
- Syntax



PR-PT3-580

Aggregate Phrase

Used in conjunction with looping through a FOR EACH.

BREAK Group

- Used when you want to conditionalize processing based on when the value of a certain field changes over a series of block iterations
- Use in conjunction with the ACCUMULATE statement and ACCUM function.
- Syntax

```
FOR EACH file WHERE expression BREAK BY field1:  
    IF FIRST-OF(field) THEN...  
    IF LAST-OF(field) THEN...  
END.
```

PR-PT3-590

BREAK Group

ACCUMULATE Statement

- Calculates one or more aggregate values of an expression during the iterations of a block
- Use the ACCUM function to access the result of this accumulation

- Syntax

`ACCUMULATE {expression(aggregate-phrase)}`

- Example

```
FOR EACH so_mstr, each sod_det where sod_nbr = so_nbr
BREAK BY so_nbr:
    ACCUMULATE sod_qty_item * sod_price(TOTAL BY so_nbr).
END.
```

PR-PT3-600

ACCUMULATE Statement

ACCUMULATE performs the function of accumulating (summing) a value.

ACCUM will access the value the ACCUMULATE statement arrived at.

02-095 - 02-096

ACCUM Function

- Returns the value of an aggregate expression that is calculated by an ACCUMULATE or aggregate phrase of a DISPLAY statement

- Syntax

ACCUM {aggregate-expression}

- Example

```
FOR EACH so_mstr, each sod_det where sod_nbr = so_nbr
BREAK BY so_cust:
  ACCUMULATE sod_qty_item * sod_price(TOTAL BY so_cust).
  IF LAST-OF(so_cust) then
    DISPLAY so_cust
    (ACCUM TOTAL BY so_cust sod_qty_item * sod_price).
END.
```

PR-PT3-610

ACCUM Function

Input/Output

- Input
 - the keyboard is the default input
 - you may redirect input from another device
 - FILE / OTHER
- Output
 - the screen is the default output
 - screen is not a paged device
 - you may redirect output to another device
 - PRINTER / FILE
 - GUI objects only display well on the screen, use STREAM-IO in your frame statement to format the frame for streaming to a text file or printer
- May have multiple INPUT & OUTPUT streams

PR-PT3-620

Input/Output

OUTPUT TO Statement

- Specifies a new output destination
- Use with DISPLAY, EXPORT, PUT statements
- Syntax:

```
OUTPUT [STREAM stream] TO {PRINTER [printer-name]
| opsys-file | opsys-device |   TERMINAL
| VALUE (expression)      |   "CLIPBOARD"   }

[ APPEND ] [ ECHO | NO-ECHO ] [ KEEP-MESSAGES ]
[ NO-MAP | MAP protermcap-entry ] [ PAGED ]
[ PAGE-SIZE { constant | VALUE ( expression ) } ]
[ UNBUFFERED ]
[ NO-CONVERT | { CONVERT [ TARGET target-codepage ] [
SOURCE source-codepage ] } ]
```

PR-PT3-630

OUTPUT TO Statement

Redirecting Output

- OUTPUT TO PRINTER
 - redirects the output to the default printer
- OUTPUT THROUGH `lp -d printer-name`
 - typical UNIX printer command
 - only available on shared memory systems
- OUTPUT TO *opsys-file*
 - outputs to a file
- OUTPUT TO TERMINAL PAGE-SIZE 20
 - good for quick testing
 - turns the screen into a paged device
- OUTPUT CLOSE
 - terminates output
 - the program end also terminates output

PR-PT3-640

Redirecting Output

DISPLAY Statement

- Uses default formats and displays it to the current output destination

- Example:

```
OUTPUT TO custfile.  
FOR EACH cm_mstr NO-LOCK:  
    DISPLAY cm_mstr.  
END.
```

PR-PT3-650

DISPLAY Statement

02-097

DISPLAY Statement

```
/* 02-097.p */  
  
OUTPUT TO TERMINAL.  
  
FOR EACH cm_mstr:  
  DISPLAY cm_mstr  
  EXCEPT cm_cr_hold  
            cm_cr_limit  
            cm_cr_rating  
            cm_cr_terms.  
  
END.
```

- Defaults to column labels
- Unquoted strings
- Untrimmed field values

Cust Sort Name	Bal Class	Salesperson[1] Salesperson[2]
3000	234,943.00	WSL
Mary Weathers	kip	jed
3001	887.00	EDU
Direct Update	gop	gar
30012	0.00	RTL
Forced Customer	kip	jed
3002	487.00	WSL
Green Stamps	kip	tor
3003	0.00	GVT
Fix this name	kip	jed
3004	0.00	EDU
WALLY'S OLD STUFF	gar	jed
3005	0.00	GVT
REVISED NAME	tor	kip
3007	0.00	
tery	464,564.00	WSL
	tor	jed

Press space bar to continue.

PR-PT3-660

DISPLAY Statement, cont.

02-098

DISPLAY Statement

```
/* 02-098.p */  
  
OUTPUT TO TERMINAL PAGED.  
  
FOR EACH cm_mstr:  
  DISPLAY cm_mstr  
  EXCEPT cm_cr_hold  
           cm_cr_limit  
           cm_cr_rating  
           cm_cr_terms  
  WITH NO-LABELS.  
END.
```

Procedure Editor - Run

3000	234,943.00	WSL
Mary Weathers	kip	jed
3001	887.00	EDU
Direct Update	gop	gar
30012	0.00	RTL
Forced Customer	kip	jed
3002	487.00	WSL
Green Stamps	kip	tor
3003	0.00	GVT
Fix this name	kip	jed
3004	0.00	EDU
WALLY'S OLD STUFF	gar	jed
3005	0.00	GVT
REVISED NAME	tor	kip
3007	0.00	
Mister Mystery		
3008	464,564.00	WSL
	tor	jed
3009	848,757.00	WSL
New Customer	tor	kip
3011	0.00	
bREAKER'S		
3012	1,000.00	
nEW nAME		
3013	19,993.00	
Sort name		
3015	398.00	

- PAGED removes default font information
- Labels removed by NO-LABELS
- Unquoted strings
- Untrimmed field values

PR-PT3-670

DISPLAY Statement, cont.

02-099

DISPLAY Statement

```
/* 02-099.p */  
  
OUTPUT TO TERMINAL PAGED.  
  
FOR EACH cm_mstr:  
  DISPLAY cm_mstr  
  EXCEPT cm_cr_hold  
            cm_cr_limit  
            cm_cr_rating  
            cm_cr_terms  
  WITH STREAM-IO.  
  
END.
```

Cust	Salesperson[1]	Salesperson[2]	Bal	Class	Sort Name
3000			234,943.00	WSL	Mary Weathers
kip	jed				
3001			887.00	EDU	Direct Update
gop	gar				
30012			0.00	RTL	Forced Customer
kip	jed				
3002			487.00	WSL	Green Stamps
kip	tor				
3003			0.00	GVT	Fix this name
kip	jed				
3004			0.00	EDU	WALLY'S OLD STUFF
gar	jed				
3005			0.00	GVT	REVISED NAME
tor	kip				
3007			0.00		Mister Mystery
3008			464,564.00	WSL	
tor	jed				
3009			848,757.00	WSL	New Customer
tor	kip				
3011			0.00		bREAKER'S

- STREAM-IO creates platform-independent output
- PAGED removes default font information
- Default column labels
- Unquoted strings
- Untrimmed field values -- exact field formats

PR-PT3-680

DISPLAY Statement, cont.

DISPLAY Statement

02-100

```
/* 02-100.p */
```

```
OUTPUT TO
```

```
  C:\_02-100.txt.
```

```
FOR EACH cm_mstr:
```

```
  DISPLAY cm_mstr
```

```
  EXCEPT cm_cr_hold
```

```
           cm_cr_limit
```

```
           cm_cr_rating
```

```
           cm_cr_terms.
```

```
END.
```

Procedure Editor - C:\Pv9\code\output01.txt

File Edit Search Buffer Compile Tools Options Help

Sort Name	Salesperson[1]Salesperson[2]	
3000	234,943.00	WSL
Mary Weathers	kip	jed
3001	887.00	EDU
Direct Update	gop	gar
30012	0.00	RTL
Forced Customer	kip	jed
3002	487.00	WSL
Green Stamps	kip	tor
3003	0.00	CVT
Fix this name	kip	jed
3004	0.00	EDU
WALLY'S OLD STUFF	gar	jed
3005	0.00	CVT
REVISED NAME	tor	kip
3007	0.00	
Hister Mystery		
3008	464,564.00	WSL
	tor	jed
3009	848,757.00	WSL
New Customer	tor	kip

- Output to file removes default font information
- Default column labels
- Unquoted strings
- Untrimmed field values
- Includes line breaks
- Note: single line of labels

PR-PT3-690

DISPLAY Statement, cont.

DISPLAY Statement

```
/* 02-101.p */  
  
OUTPUT TO  
  C:\_02-101.txt.  
  
FOR EACH cm_mstr:  
  DISPLAY cm_mstr  
  EXCEPT cm_cr_hold  
           cm_cr_limit  
           cm_cr_rating  
           cm_cr_terms  
  WITH STREAM-IO.  
END.
```

Procedure Editor - C:\Pv9\code\output02.txt

File Edit Search Buffer Compile Tools Options Help

Cust		Bal	Class	Sort	Name
Salesperson[1] Salesperson[2]					

3000		234,943.00	WSL		Mary Weathers
kip	jed				
3001		887.00	EDU		Direct Update
gop	gar				
30012		0.00	RTL		Forced Customer
kip	jed				
3002		487.00	WSL		Green Stamps
kip	tor				
3003		0.00	CVT		Fix this name
kip	jed				
3004		0.00	EDU		WALLY'S OLD STUFF
gar	jed				
3005		0.00	CVT		REVISED NAME
tor	kip				
3007		0.00			Mister Mystery
3008		464,564.00	WSL		
tor	jed				
3009		848,757.00	WSL		New Customer
tor	kip				

- Output to file removes default font information
- Default column labels
- Unquoted strings
- Includes line breaks
- Untrimmed field values -- exact field formats
- Note: both lines of labels

PR-PT3-700

DISPLAY Statement, cont.

EXPORT Statement

- Converts data to a standard character format and displays it to the current output destination

- Syntax

```
EXPORT [STREAM stream] [DELIMITER character]  
      {expression...|record [EXCEPT field...] }
```

- Example

```
OUTPUT TO custfile.  
FOR EACH cm_mstr NO-LOCK:  
    EXPORT cm_mstr.  
END.
```

PR-PT3-710

EXPORT Statement

02-103

EXPORT Statement

```

/* 02-103.p */

OUTPUT TO
  C:\_02-103.txt.

FOR EACH cm_mstr:
  EXPORT cm_mstr
  EXCEPT cm_cr_hold
           cm_cr_limit
           cm_cr_rating
           cm_cr_terms.
END.

```

```

Procedure Editor - C:\_Pv9\code\output02.txt
File Edit Search Buffer Compile Tools Options Help
"3000" 234943 "WSL" "Mary Weathers" "kip" "jed"
"3001" 887 "EDU" "Direct Update" "gop" "gar"
"30012" 0 "RTL" "Forced Customer" "kip" "jed"
"3002" 487 "WSL" "Green Stamps" "kip" "tor"
"3003" 0 "GVT" "Fix this name" "kip" "jed"
"3004" 0 "EDU" "WALLY'S OLD STUFF" "gar" "jed"
"3005" 0 "GVT" "REVISED NAME" "tor" "kip"
"3007" 0 "" "Mister Mystery" "" ""
"3008" 464564 "WSL" "" "tor" "jed"
"3009" 848757 "WSL" "New Customer" "tor" "kip"
"3011" 0 "" "BREAKER'S" "" ""
"3012" 1000 "" "NEW NAME" "" ""
"3013" 19993 "" "Sort name" "" ""
"3015" 398 "" "New Customer Again" "" ""
"3023" 0 "" "Bill's Flowers" "" ""
"333" 0 "RTL" "Changed name" "gar" "gop"
"3400" 0 "WSL" "Cust 3004" "gar" "kip"
"4031" 44099 "RTL" "Garrison Graphics" "gap" "kip"
"9000" 0 "RTL" "MARSHALL HOWDY GREEN" "tor" "gop"
"9001" 0 "RTL" "Harry's" "tor" "gop"
"9009" 0 "WSL" "BrevHaHa" "tor" "gop"
"9087" 0 "WSL" "Freeman's Foghorns" "gar" "kip"

```

- Output to file removes default font information
- No labels
- Includes line breaks
- Quoted strings
- Trimmed field values

PR-PT3-720

EXPORT Statement, cont.

02-104 - 02-105

EXPORT Statement

```
/* 02-104.p */
```

```
OUTPUT TO
```

```
  C:\_02-104.txt.
```

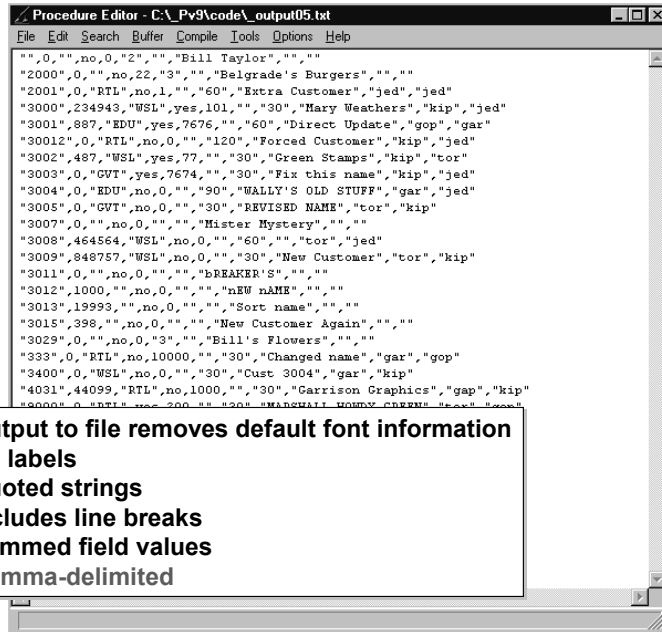
```
FOR EACH cm_mstr:
```

```
  EXPORT
```

```
    DELIMITER ","
```

```
    cm_mstr.
```

```
END.
```



```

"0","0","no","0","2","","Bill Taylor",""
"2000","0","no","22","3","","Belgrade's Burgers",""
"2001","0","RTL","no","1","","60","Extra Customer","jed","jed"
"3000","234943","WSL","yes","101","","30","Mary Weathers","kip","jed"
"3001","887","EDU","yes","7676","","60","Direct Update","gop","gar"
"30012","0","RTL","no","0","","120","Forced Customer","kip","jed"
"3002","487","WSL","yes","77","","30","Green Stamps","kip","tor"
"3003","0","CUT","yes","7674","","30","Fix this name","kip","jed"
"3004","0","EDU","no","0","","90","WALLY'S OLD STUFF","gar","jed"
"3005","0","CUT","no","0","","30","REVISED NAME","tor","kip"
"3007","0","no","0","","","Mister Mystery",""
"3008","464564","WSL","no","0","","60","tor","jed"
"3009","848757","WSL","no","0","","30","New Customer","tor","kip"
"3011","0","no","0","","","bBREAKER'S",""
"3012","1000","no","0","","","nEW NAME",""
"3013","19993","no","0","","","Sort name",""
"3015","398","no","0","","","New Customer Again",""
"3029","0","no","0","3","","Bill's Flowers",""
"333","0","RTL","no","10000","","30","Changed name","gar","gop"
"3400","0","WSL","no","0","","30","Cust 3004","gar","kip"
"4031","44099","RTL","no","1000","","30","Garrison Graphics","gap","kip"

```

- Output to file removes default font information
- No labels
- Quoted strings
- Includes line breaks
- Trimmed field values
- Comma-delimited

PR-PT3-730

EXPORT Statement, cont.

PUT Statement

- Sends the value of one or more expressions to an output destination other than the terminal

- Syntax:

```
PUT [STREAM stream] [UNFORMATTED]
[expression [FORMAT string] [{AT|TO} expression]]|
{SKIP[(expression)]}|{SPACE[(expression)]}...
```

- Or:

```
PUT [STREAM stream] CONTROL expression ...
```

PR-PT3-740

PUT Statement

PUT Statement

02-106

```
/* 02-106.p */  
  
OUTPUT TO  
  C:\_02-106.txt.  
  
FOR EACH cm_mstr:  
  PUT TODAY SPACE  
    cm_addr cm_sort  
    cm_xslspn[1]  
    cm_xslspn[2]  
    cm_balance  
    cm_class SKIP.  
END.
```

Procedure Editor - C:\Pv9\code\OUTPUT3.txt

08/07/00	3000	Mary Weathers	kip	jed	234,943.00WS:
08/07/00	3001	Direct Update	gop	gar	887.00ED:
08/07/00	30012	Forced Customer	kip	jed	0.00RT:
08/07/00	3002	Green Stamps	kip	tor	487.00WS:
08/07/00	3003	Fix this name	kip	jed	0.00CV:
08/07/00	3004	WALLY'S OLD STUFF	gar	jed	0.00ED:
08/07/00	3005	REVISED NAME	tor	kip	0.00CV:
08/07/00	3007	Mister Mystery			0.00
08/07/00	3008		tor	jed	464,564.00WS:
08/07/00	3009	New Customer	tor	kip	848,787.00WS:
08/07/00	3011	bBREAKER'S			0.00
08/07/00	3012	nEW nAME			1,000.00
08/07/00	3013	Sort name			19,993.00
08/07/00	3015	New Customer Again			398.00
08/07/00	3029	Bill's Flowers			0.00
08/07/00	333	Changed name	gar	gop	0.00RT:
08/07/00	3400	Cust 3004	gar	kip	0.00WS:
08/07/00	4031	Garrison Graphics	gap	kip	44,099.00RT:
08/07/00	9000	MARSHALL HOWDY GREEN	tor	gop	0.00RT:
08/07/00	9001	Harry's	tor	gop	0.00RT:
					0.00WS:
					0.00WS:

- Output to file removes default font information
- No labels
- Forced line breaks
- Unquoted strings
- Untrimmed field values
- Cannot PUT buffers; must use field names

PR-PT3-750

PUT Statement, cont.

02-107

PUT Statement

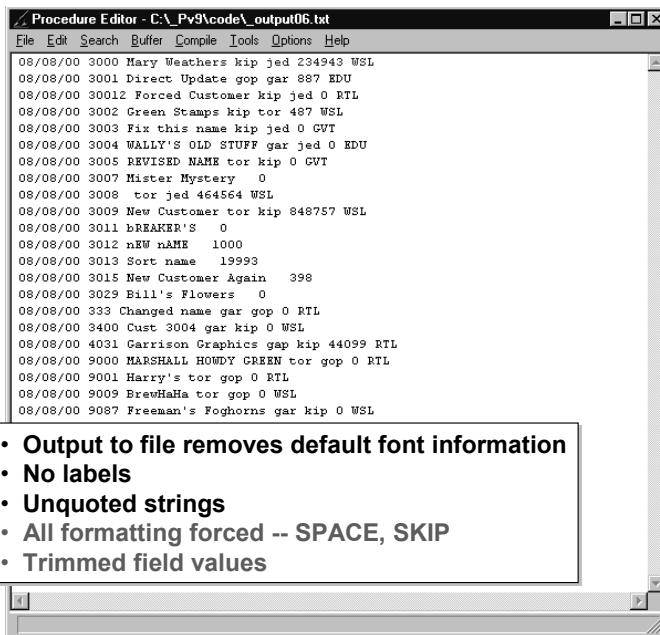
```

/* 02-107.p */

OUTPUT TO
  C:\_02-107.txt.

FOR EACH cm_mstr:
  PUT UNFORMATTED
    TODAY SPACE
    cm_addr SPACE
    cm_sort SPACE
    ...
    cm_class SKIP.
END.

```



PR-PT3-760

PUT Statement, cont.

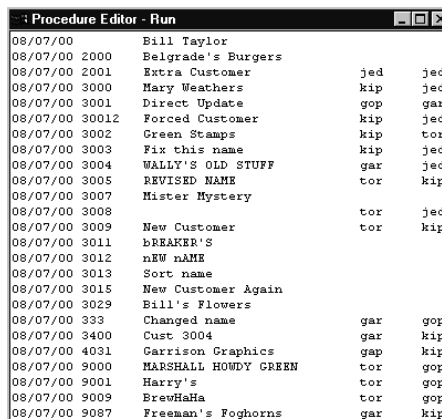
PUT Statement

02-108

```
/* 02-108.p */
```

```
OUTPUT TO TERMINAL PAGED.
```

```
FOR EACH cm_mstr:
  PUT TODAY SPACE
  cm_addr cm_sort
  cm_xslspn[1]
  cm_xslspn[2]
  cm_balance
  cm_class SKIP.
END.
```



```

08/07/00 Bill Taylor
08/07/00 2000 Belgrade's Burgers
08/07/00 2001 Extra Customer          jed   jed
08/07/00 3000 Mary Weathers           kip   jed
08/07/00 3001 Direct Update          gop   gar
08/07/00 30012 Forced Customer        kip   jed
08/07/00 3002 Green Stamps           kip   tor
08/07/00 3003 Fix this name          kip   jed
08/07/00 3004 WALLY'S OLD STUFF       gar   jed
08/07/00 3005 REVISED NAME          tor   kip
08/07/00 3007 Mister Mystery         tor   jed
08/07/00 3008                               tor   kip
08/07/00 3009 New Customer           tor   kip
08/07/00 3011 bBREAKER'S
08/07/00 3012 nEW NAME
08/07/00 3013 Sort name
08/07/00 3015 New Customer Again
08/07/00 3029 Bill's Flowers
08/07/00 333 Changed name            gar   gop
08/07/00 3400 Cust 3004              gar   kip
08/07/00 4031 Garrison Graphics      gar   kip
08/07/00 9000 MARSHALL HOWDY GREEN   tor   gop
08/07/00 9001 Harry's               tor   gop
08/07/00 9009 BrevHaHa              tor   gop
08/07/00 9087 Freeman's Foghorns     gar   kip
  
```

- System font
- No labels
- Unquoted strings
- Untrimmed field values
- Will not appear without PAGED keyword

PR-PT3-770

PUT Statement, cont.

02-109

PUT Statement

```
/* 02-109.p */
```

```
OUTPUT TO TERMINAL PAGED.
```

```
FOR EACH cm_mstr:
```

```
  PUT UNFORMATTED
```

```
    TODAY SPACE
```

```
    cm_addr cm_sort
```

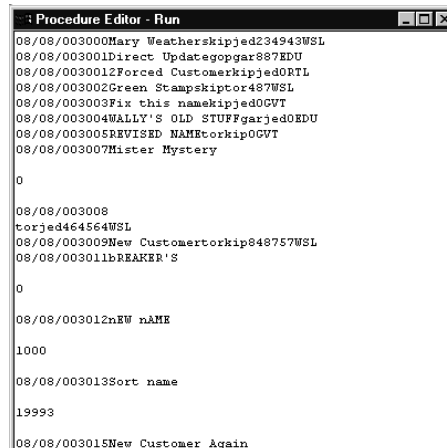
```
    cm_xslspn[1]
```

```
    cm_xslspn[2]
```

```
    cm_balance
```

```
    cm_class SKIP.
```

```
END.
```



```
08/08/003000Mary Weatherskipjed234943WSL
08/08/003001Direct Updatagopgar887EDU
08/08/0030012Forced Customerkipjed0RTL
08/08/003002Green Stampskiptor487WSL
08/08/003003Fix this namekipjed0GVT
08/08/003004WALLY'S OLD STUFFgarjed0EDU
08/08/003005REVISED NAMErtorkip0GVT
08/08/003007Mister Mystery

0
08/08/003008
torjed464564WSL
08/08/003009New Customertorkip848757WSL
08/08/003011bREAKER'S

0
08/08/003012nEW nAME
1000
08/08/003013Sort name
19993
08/08/003015New Customer Again
```

- System font
- No labels
- Unquoted strings
- Trimmed field values
- SPACE keyword ignored
- Will not appear without PAGED keyword

PR-PT3-780

PUT Statement, cont.

Headers and Footers

- Can be defined for any output device
- Steps:
 - Define the frame using the HEADER keyword
 - Use PAGE-TOP, PAGE-BOTTOM keywords
 - VIEW the frame

PR-PT3-790

Headers and Footers

RULES for Headers and Footers:

- Define a frame.
- Use HEADER keyword.
- Use PAGE-TOP keyword.
- VIEW FRAME statement.

02-110 - 02-111

Output to Multiple Streams

- Define each stream
- Attach each stream to an output device
- Use the streams when they're referenced:
 - `VIEW STREAM s1`
 - `DISPLAY STREAM s2`
 - `PUT STREAM s1`
 - `PAGE STREAM s1`
- For frames in streams, use STREAM-IO option on the Frame Phrase

PR-PT3-800

Output to Multiple Streams

INPUT Statement

- Specifies a new input source

- Basic Syntax:

```
INPUT [ STREAM stream ] FROM { opsys-file | opsys-device
| TERMINAL | VALUE ( expression ) | OS-DIR ( directory ) [
NO-ATTR-LIST ] }
```

- A single period is read as an END-ERROR unless period is in quotes (".") treated as ordinary character
- Tilde (~) (no space) indicates line continuation
- Hyphen indicates skipping a field in a subsequent INSERT, PROMPT-FOR, SET or UPDATE

PR-PT3-810

INPUT Statement

Defines a new input source other than the keyboard.

02-112 - 02-113

IMPORT Statement

- Reads a line from an input file that might have been created by EXPORT

- Syntax

```
IMPORT [STREAM stream] {[DELIMITER character]
  {field|^|record}}|{UNFORMATTED field}} [NO-
  ERROR]
```

- **STREAM *stream***
 - specifies the name of a stream. If you do not name a stream, PROGRESS uses the default unnamed stream
- **DELIMITER *character***
 - the character used as a delimiter between field values in the file. The character parameter must be a quoted single character. The default is a space character

PR-PT3-820

IMPORT Statement

Operating System Commands

- OPSYS Function
 - identifies the OS being used
- OS-APPEND Statement
 - file append command
- OS-COMMAND Statement
 - escapes to current OS and executes an OS command
- OS-COPY Statement
 - OS file copy command
- OS-CREATE-DIR Statement
 - creates new directory
- OS-DELETE Statement
 - file delete

PR-PT3-830

Operating System Commands

02-114

Operating System Commands

- OS-DRIVES Function
 - returns a list of available drives
- OS-ERROR Function
 - returns a PROGRESS error code that indicates whether an execution error occurred during the last OS command
- OS-GETENV Function
 - returns the value of the specified environment variable
- OS-RENAME Statement
 - file/directory rename

PR-PT3-840

Operating System Commands, cont.

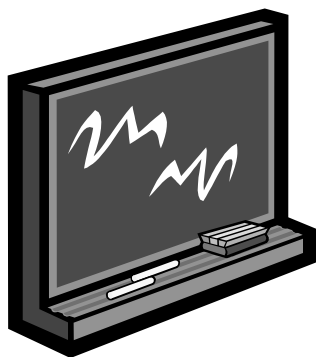
LAB 7

1. Create a sales report BY customer BY order. Send it to the terminal. (L0701.p)
2. Modify 1 to use defined frames. (L0702.p)
3. Modify 2 to use headers and footers (L0703.p)
4. BONUS: Modify 3 by adding a stream showing salesperson information per order to an operating system file. (L0704.p)

PR-PT3-850

Lab 7

Lab 7 Review



PR-PT3-860

Lab 7 Review

Code Structures

- Functions
- Include Files
- Internal & External Procedures

PR-PT3-870

Code Structures

4GL Functions

- Prepackaged solution, sometimes accepts runtime arguments, returning a value to be used in an expression
 - Arithmetic: RANDOM, EXP
 - Character: STRING, TRIM, LENGTH
 - Date: TODAY, MONTH
 - Validation: AVAILABLE, CAN-FIND, LAST-OF
 - State: CURRENT-CHANGED, PROPATH, CONNECTED
- Global
 - No need to declare the function

PR-PT3-880

4GL Functions

User-Defined Functions

- Let applications define a logical rule or transformation once, then apply the rule or transformation an unlimited number of times

- Syntax

```
FUNCTION function-name [RETURNS] data-type
  [(param[,param]...)] {FORWARD | [MAP [TO] actual-name] IN
  proc-handle}
```

- returns a single value
- useful for complex calculations
- no user interface statements
- local variables allowed

PR-PT3-890

User-Defined Functions

User-Defined Functions

- Avoid PROGRESS keywords
- Define the function, or forward declare it, before calling it
- Can be run externally
- Can be run persistently
- May be used in DISPLAY statements and BROWSE definitions

PR-PT3-900

User-Defined Functions, cont.

FORWARD declare allows the function to be defined (referenced) before it is called, but allows the actual structure of the function to be done later (forward) in the program for clarity.

04-001 - 04-007

User-Defined Functions

New PRIVATE attribute

- You cannot invoke PRIVATE functions from external procedures
- The INTERNAL-ENTRIES attribute on the procedure that defines the function does not provide the function name (unless the procedure that defines it is the current procedure file)
- The GET-SIGNATURE method on the procedure that defines it does not provide its signature (unless the procedure that defines it is the current procedure file)

v9

PR-PT3-910

User-Defined Functions, cont.

This new attribute allows you to initiate user-defined functions only for the initiating program.

04-008 - 04-010

Include Files

- Causes PROGRESS to retrieve the statements in a file and compile them as part of the main procedure
- Syntax

```
{include-file [argument | &arg-name="arg-value"]...}
```
- One method for reuse of code
- PROGRESS searches the PROPATH to find include files at compile time

PR-PT3-920

Include Files

PROGRESS Procedures

Internal and External Procedures

- Can contain all other types of blocks
- Can execute by name using the RUN statement
- Can accept run-time parameters for input or output
- Can define their own data and UI environment, with restrictions depending on the type of procedure
- Can share data and widgets defined in the context of another procedure, with restrictions depending on the type of procedure
- Can execute recursively

PR-PT3-930

Progress Procedures

- Procedure 1 is main block, calling procedure 2.
- Procedure 2 context is invisible to procedure 1.
- Procedure 2 can access shared objects or buffers from procedure 1 or parameters passed from 1.

RUN Statement

- Calls a procedure which can be local or remote, external or internal
- Basic syntax
 - `RUN [proc-name|VALUE(extern-expression)] .`
- PROGRESS tries to run an internal procedure first
 - if *proc-name* is defined as an internal procedure within the current procedure, PROGRESS first tries to execute the internal procedure
- If an internal procedure is not defined, PROGRESS tries to run an external procedure
 - searches the directories listed in the PROPATH
- V9, searches for super-procedures...

PR-PT3-940

RUN Statement

Internal Procedures

- Block of code defined within an external procedure
- Called with the RUN statement:
- Can be referenced by procedures other than the containing procedure
- Part of the context of the containing procedure
- Can use with parameters or a RETURN-VALUE

```
/* Proc1.p */  
...code  
RUN proc2.  
  
...  
  
PROCEDURE proc2:  
    DO WHILE TRUE:  
        WHAT I SAY.  
END.
```

PR-PT3-950

Internal Procedures

Scope versus context: context is the application environment in which the procedure executes. Scope is the duration a specific context is available.

Internal Procedure Limitations

- Cannot define any shared object definitions
- Cannot define streams
- Cannot define work tables
- Cannot define temp tables
- Cannot export locally defined frames or buffers

PR-PT3-960

Internal Procedure Limitations

04-011 - 04-016

RETURN Statement

- Leaves the procedure block and returns to the calling procedure, or to the Procedure Editor if there is no calling procedure
- Syntax
`RETURN [ERROR|NO-APPLY] [return-value]`
- Use the RETURN-VALUE function to check the value being returned

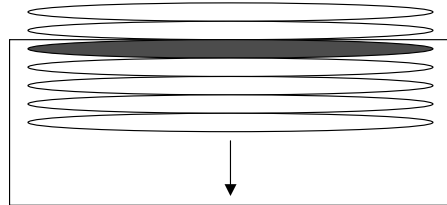
PR-PT3-970

RETURN Statement

External Procedures

- PROGRESS's fundamental procedure
- Created, stored, and compiled separately as an operating system file
- Can be called from any other procedure using the RUN Statement
- Can be run persistently (added to current and future procedure scope and context)

Call stack: If the colored procedure is run persistently, its functions, variables, internal procedures, etc. are available to programs added to the stack later in the same session. →



PR-PT3-980

External Procedures

Scope versus context: context is the application environment in which the procedure executes. Scope is the duration a specific context is available.

Passing Parameters

The diagram illustrates the flow of parameters between a `RUN` statement and a `PROCEDURE` definition. In the `RUN` statement, `INPUT-OUTPUT x` and `OUTPUT y` are enclosed in boxes. In the `PROCEDURE` definition, `newx.` and `newz.` are also enclosed in boxes. Arrows show the following connections: an arrow from the `newx.` box in the procedure to the `INPUT-OUTPUT x` box in the run statement; an arrow from the `newz.` box in the procedure to the `OUTPUT y` box in the run statement; and an arrow from the `newx.` box in the procedure to the `newz.` box in the procedure.

```
RUN nextproc (INPUT-OUTPUT x, OUTPUT y, z)
```

```
PROCEDURE nextproc:  
  DEFINE INPUT-OUTPUT PARAMETER newx.  
  DEFINE OUTPUT PARAMETER newy.  
  DEFINE INPUT PARAMETER newz.
```

*Output is requesting output from the called procedure.
Input is providing input to the called procedure.*

PR-PT3-990

Passing Parameters

04-017 - 04-019

DEFINE PARAMETER Statement

- Defines a run-time parameter in a sub-procedure or Windows dynamic link library (DLL) routine
- Basic syntax

```
DEFINE {INPUT | OUTPUT | INPUT-OUTPUT} PARAMETER
      parameter {AS datatype}
```
- Parameter is a variable in a called procedure
 - accepts an input value from calling procedure, outputs a value to a calling procedure, or both
- Must specify in the RUN statement in the same order they are defined with the DEFINE statements
 - Datatypes must agree
 - Cannot pass an array as a parameter

PR-PT3-1000

DEFINE PARAMETER Statement

04-020 - 04-023

DEFINE VARIABLE Statement

- Shared variables
 - global variables are global to your session only
 - use NEW and GLOBAL in the defining programs
 - PROGRESS searches the call stack looking for the most recent DEFINE NEW SHARED VARIABLE statement that created that shared variable
 - data types and array extents must match

- Syntax

```
DEFINE [[ NEW [GLOBAL]] SHARED] VARIABLE variable { AS datatype |
LIKE field } [NO-UNDO] [BGCOLOR expression]
[COLUMN-LABEL label] [DCOLOR expression] [DECIMALS n] [DROP-
TARGET] [EXTENT n] [FONT expression] [FGCOLOR expression]
[FORMAT string] [INITIAL { constant|{ [ constant [,constant]
...]]}] [LABEL string[,string] ...] [MOUSE-POINTER expression]
[[NOT]CASE-SENSITIVE] [PFCOLOR expression]{[view-as-phrase]}
{[trigger-phrase]}
```

PR-PT3-1010

DEFINE VARIABLE Statement

Persistent Procedures

- Creates context upon execution and retains context until the end of the Progress session
- PERSISTENT attribute on RUN statement
- Basic method for encapsulation in Progress

PR-PT3-1020

Persistent Procedures

MFG/PRO Code Details

- Maintenance Program
- Inquiry Program
- Report Program

PR-PT3-1030

MFG/PRO Code Details

Maintenance Program Template

```

/* samplemt.p - Sample Maintenance program */
/*V8:ConvertMode= Maintenance */
/*REVISION: x.x LAST MODIFIED MM/DD/YY BY: XXX *ECOX**/
{mfdtitle.I "g"}

Mainloop: REPEAT:
    PROMPT-FOR fields...    WITH FRAME EDITING:
        {mfnp.I} /*NEXT/PREV logic */
    END.

    /* ADD/MODIFY/DELETE SECTION */

END. /* MAINLOOP */
STATUS INPUT.

```

PR-PT3-1040

Maintenance Program Template

Maintenance Program Template

```

• /*ADD/MODIFY/DELETE */
FIND table WHERE key1 = ... NO-LOCK no-ERROR.
If NOT AVAILABLE table1 THEN DO:
    {mfmsg.I 1 1} /* ADDING NEW RECORD */
    Create New Record
END.
Status = stline[2].
STATUS INPUT ststatus.
DISPLAY fields.
Setloop: ON ENDKEY UNDO blockx, RETRY blockx:
    SET fields .... GO-ON (F5 CTRL-D).
    IF LASTKEY = KEYCODE(F5) OR .... /*DELETE */
        del_yn = YES.
        {mfmsg01.I 11 1 del_yn}
        if del_yn = YES THEN DO:
            DELETE table1.
        END.
        VALIDATIONS ...
    END. /* SETLOOP */

```

PR-PT3-1050

Maintenance Program Template, cont.

adcsmt.p b+ 2.1.1 Customer Maintenance 02/27/03

Customer Address

Customer: 001
 Name: PEI Computers Ltd.
 Address: 37 Atlantic Road
 Address: 1234567890123456789012345678
 Address:
 City: bedford State: ma Post: Format: 0
 Country: United States of America usa County:
 Attention: [2]:
 Telephone: 123-456-7890 Ext: [2]: Ext:
 Fax/Telex: 123-456-7891 [2]: Added: 04/28/95

Customer Data

Sort Name: PEI Computers Type:
 Salespsn1: Multiple: No Region:
 Ship Via: ups cur: usd
 AR Acct: 1200 0200 Dual Pricing Cur:
 Resale: Site: 10000
 Remarks: Lang: US

F1=Go 2=Hlp 3=Ins 4=End 6=Mnu 7=Rcl 8=Clr 9=Prev 10=Next 11=Buf
 F1=Go 2=Hlp 3=Ins 4=End 6=Mnu 7=Rcl 8=Clr 9=Prev 10=Next 11=Buf

PR-PT3-1060

```
• /* adcsmt.p */  
  
run p-upd-frameb(input-output reccm,  
                 input-output recad output l_exit).
```

PR-PT3-1070

```

•
  /* adcsmt.p */
do transaction: /* Oracle lock problem */
  loopa:
    do with frame a on endkey undo, leave loopa:

•
  display cm_addr
    ad_name ad_line1 ad_line2 ad_line
    ad_city ad_state ad_zip ad_format ad_country
    ad_ctry when ({txnew.i} or ec_ok) ad_date
    ad_attn ad_phone ad_ext ad_attn2 ad_phone2 ad_ext2 ad_fax
    ad_fax2 ad_county.
  set1:
    do on error undo, retry:
      set ad_name ad_line1 ad_line2 ad_line3
        ad_city ad_state ad_zip ad_format
        ad_country when ((not {txnew.i}) and (not ec_ok))
        ad_ctry when ({txnew.i} or ec_ok) ad_county
        ad_attn ad_phone ad_ext ad_fax
        ad_attn2 ad_phone2 ad_ext2 ad_fax2 ad_date
      go-on (F5 CTRL-D).

```

PR-PT3-1080

```

· /* adcsmt.p */
  set
    cm_sort cm_slspn[1] mult_slsp
    cm_shipvia
    cm_ar_acct
    cm_ar_sub
    cm_ar_cc
    cm_resale cm_rmks cm_type cm_region cm_curr
    cm_scurr when (et_print_dc and
      (new_cmmstr or is-union or is-member or new_curr <> old_curr))
    cm_site
    cm_lang

  with frame b.
setb2:
  do on error undo, retry:
    set
      cm_taxable when (not {txnew.i})
      cm_taxc when (not {txnew.i})
      cm_pr_list2
      cm_pr_list
      cm_fix_pr
      cm_class
      cm_partial
/*N20Y*/ cm_qadl01
  with frame b2.

```

PR-PT3-1090

adcsmt.p b+

2.1.1 Customer Maintenance

02/27/03

Customer: 001

Name: PEI Com

Address: 37 Atla

Address: 1234567

Address:

City: bedford

Country: United

Attention:

Telephone: 123-456

Fax/Telex: 123-456

Customer Address

Address Tax Data

Taxable: Yes

Tax Zone: SA

Tax Class: n

Tax Usage:

Tax In: No

Tax ID - Federal: dd

VAT Registration:

Tax ID - Misc 1:

Tax ID - Misc 2:

Tax ID - Misc 3:

In City: No

Format: 0

Ext:

Added: 04/28/95

Taxable: Ye

Price Table:

Discount Tbl:

Fixed Price: No

Class:

Invoice by Authorization: No

F1=Go 2=Help 3=Ins 4=End 6=Menu 7=Rcl 8=Clr 11=Paste

PR-PT3-1100

```
• /* adcsmt.p */
setb2:
    do on error undo, retry:
        set
            cm_taxable when (not {txnew.i})
            cm_taxc      when (not {txnew.i})
            cm_pr_list2
            cm_pr_list
            cm_fix_pr
            cm_class
            cm_partial
/*N20Y*/    cm_qadl01
            with frame b2.
```

PR-PT3-1110

Inquiry Program Template

```

/* samplemt.p - Sample Inquiry program */
/*V8:ConvertMode */
/*REVISION: x.x LAST MODIFIED MM/DD/YY BY: XXX *ECOX**/
{mfdtitle.I "g"}

REPEAT:
    [get selection criteria] with frame a.
    {mfselfprt.I "terminal 80}
    FOR EACH [selection criterial]
        DISPLAY ...
        {mfrpchk.I}
    END.
    {mfreset.I}
    {mfmsg.I 8 1}
END.

```

PR-PT3-1120

Inquiry Program Template

socriq.p b+7.1.14 Sales Order Credit Inquiry02/28/03

Order	Bill To	Ord Date	Due Date	St	Output
_____	_____	__/__/__	__/__/__	__	

PR-PT3-1130

```

·  /* socriq.p */
   form
       nbr
       cust
       ad_name no-label format "x(20)"
       ord_date
       due_date
       act_stat
       with frame a no-underline attr-space width 80.

       /* SET EXTERNAL LABELS */
       setFrameLabels(frame a:handle).

/*L024* find first gl_ctrl no-lock. *L024*/
repeat:
    if ord_date = low_date then ord_date = ?.
    if due_date = low_date then due_date = ?.
    update nbr cust ord_date due_date act_stat
    with frame a editing:
    {mfselfprt.i "terminal" 80}
    if nbr <> "" then
        for each so_mstr no-lock where (so_nbr = nbr)
        with frame c down width 80:
            /* SET EXTERNAL LABELS */
            setFrameLabels(frame c:handle).
            {mfrpchk.i}

```

PR-PT3-1140

```
·   else if cust <> "" then
      for each so_mstr no-lock where (so_bill = cust)    and
        (so_ord_date = ord_date or ord_date = low_date) and
        (so_due_date = due_date or due_date = low_date) and
        (so_stat = act_stat or act_stat = "") with frame d down:

      ...
      display so_bill so_nbr so_conf_date so_due_date so_ship so_cr_terms
              open_amt so_stat with frame e width 80 down.
```

PR-PT3-1150

Report Program Template

```

• /* samplmt.p - Sample Report program */
/*V8:ConvertMode FULLGUIReport*/
/*REVISION: x.x LAST MODIFIED MM/DD/YY BY: XXX *ECOX**/
{mfdtitle.I "g"}
DEFINE parameter/variable/form definition
REPEAT:
  IF Selectionx = hi_datatype_value THEN selectionx = default_value.
  UPDATE
    {mfquoter.I selectionx}
    {mfhead.I}
    FOR EACH [selection criterial]
      DISPLAY ...
      {mfrpchk.I}
    END.
    {mfreset.I}
  END.

```

PR-PT3-1160

Report Program Template

socrrp.p b+ 7.1.15 Sales Order Credit Report 02/28/03

Sales Or'r:	_____	To:	_____
Bill To:	_____	To:	_____
Order Date:	__/__/__	To:	__/__/__
Due Date:	__/__/__	To:	__/__/__
Action Status:	__	To:	__
Include Unconfirmed: <u>No</u>		Output: Batch ID:	

F1=Go 2=Help 3=Ins 4=End 6=Menu 7=Rcl 8=Clr 11=Paste

PR-PT3-1170

```

Form
    nbr          colon 20
    nbr1         label {t001.i} colon 54 skip
    cust         colon 20
    cust1        label {t001.i} colon 54 skip
    ord_date     colon 20
    ord_date1    label {t001.i} colon 54 skip
    due_date     colon 20
    due_date1    label {t001.i} colon 54 skip
    act_stat     colon 20
    act_stat1    label {t001.i} colon 54 skip(1)
    include_unconf colon 25
    with frame a side-labels attr-space width 80.
repeat:
    if nbr1 = hi_char then nbr1 = "".
    if cust1 = hi_char then cust1 = "".
    if ord_date = low_date then ord_date = ?.
    if ord_date1 = hi_date or ord_date1 = low_date then ord_date1 = ?.
    if due_date = low_date then due_date = ?.
    if due_date1 = hi_date or due_date1 = low_date then due_date1 = ?.
    if act_stat1 = hi_char then act_stat1 = "".
view frame a.
    update nbr
    nbr1
    cust
    cust1 ...
    with frame a side-labels attr-space width 80.
    bcdparm = "".
    {mfquoter.i nbr      }
    {mfquoter.i nbr1    } ...

```

PR-PT3-1180

```

.   if nbr1 = "" then nbr1 = hi_char.
   if cust1 = "" then cust1 = hi_char.
   if ord_date = ? then ord_date = low_date.
   if ord_date1 = ? then ord_date1 = hi_date.
   if due_date = ? then due_date = low_date.
   if due_date1 = ? then due_date1 = hi_date.
   if act_stat1 = "" then act_stat1 = hi_char.
/* Select printer */
   {mfsselbpr.i "printer" 132}
   {mfthead.i}
   last_so_bill = "".
   for each so_mstr use-index so_bill no-lock where
       so_bill   >= cust   and so_bill   <= cust1 and
       so_nbr    >= nbr    and so_nbr    <= nbr1 and
       so_ord_date >= ord_date and so_ord_date <= ord_date1
/*NOTF*   with frame b width 132 */
/*NOTF*/   with frame b width 132 down
       break by so_bill by so_ord_date:
       accumulate open_amt (total by so_bill).
       if last_so_bill <> so_bill then do with frame c:
           /* SET EXTERNAL LABELS */
           setFrameLabels(frame c:handle).
           Display
               cm_addr
               cust_name
               cm_high_date
               cm_cr_hold ...
           with frame c width 132.

```

PR-PT3-1190

Your Feedback Is Important to Us

Please take a few minutes to complete and return this feedback sheet. It will help us to improve the quality of our training materials. For your convenience, you can also download this form from our web site. After you have completed the form, return it to us by fax: (856) 840-2612, or send your comments to us by email.

Web site: <http://www.qad.com/services/learn>

Email: learningservices@qad.com

Training Guide: Progress Programming for MFG/PRO: Introduction

Item number: 70-2865A

How would you rate the quality of this training guide?

Organization of information:	☐ Excellent	☐ Good	☐ Fair
Ease of use:	☐ Excellent	☐ Good	☐ Fair
Content:	☐ Excellent	☐ Good	☐ Fair
Overall effectiveness:	☐ Excellent	☐ Good	☐ Fair

What suggestions, corrections, and additions to the training materials do you have?
(For specific changes, please refer to the page number and paragraph so that we can identify it.)

To order training materials from QAD, visit our web site and download the Training Materials Order Form. Send the completed order form to:

Fax: (856) 840-2612, or

email: COPS@qad.com

