TPF Database Facility

General Information

Release 1

GH31-0177-02

TPF Database Facility

General Information

Release 1

GH31-0177-02

Note!
Before using this information and the product it supports, be sure to read the general information under “Notices”.

Third Edition (May 1999)

This is a major revision of, and obsoletes, GH31-0177-01.

This edition applies to Version 1 Release 1 Modification Level 3 of IBM Transaction Processing Facility Database Facility, program number 5706-196, and to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of the product.

IBM welcomes your comments. Address your comments to:

IBM Corporation
TPF Systems Information Development
Mail Station P923
2455 South Road
Poughkeepsie, NY 12601-5400
USA

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

About This Book

This book introduces the TPF Database Facility (TPFDF) product, an IBM licensed program. The TPFDF product is a database manager for application programs that run in a Transaction Processing Facility (TPF) or Airline Control System (ALCS) operating environment.

In this book, abbreviations are often used instead of spelled-out terms. Every term is spelled out at first mention followed by the all-caps abbreviation enclosed in parentheses; for example, structured programming macro (SPM). Abbreviations are defined again at various intervals throughout the book. In addition, the majority of abbreviations and their definitions are listed in the master glossary in *TPFDF Glossary*.

Who Should Read This Book

This book is intended for all data processing professionals who work with TPF or ALCS systems and do not have a detailed knowledge of the TPFDF product. Anyone who is new to the TPFDF product should read this book before reading any of the other books in the TPFDF product library.

After reading this book, you should be able to:

- Understand the benefits obtained by using the TPFDF product
- Understand the components and features of the TPFDF product
- Evaluate the TPFDF product to determine if it is suitable for your installation
- Prepare for the installation of the TPFDF product.

Conventions Used in the TPFDF Library

The TPFDF library uses the following conventions:

Typography	Examples of Usage
italic	Used for important words and phrases. For example: A *database* is a collection of data. Used to represent variable information. For example: Enter **ZUDFC DISPLAY ID-***fileid*, where *fileid* is the file identifier (ID) of the file for which you want statistics.
bold	Used to represent keywords. For example: Enter **ZUDFC HELP** to obtain help information for the ZUDFC command.

`monospaced`	Used for messages and information that displays on a screen. For example:
	`PROCESSING COMPLETED`
	Used for C language functions. For example:
	`dfcls`
	Used for examples. For example:
	`ZUDFC DISPLAY ID-J5`

bold italic	Used for emphasis. For example: You ***must*** type this command exactly as shown.

Typography	Examples of Usage
CAPital LETters	Used to indicate valid abbreviations for keywords. For example: KEYWord=*option*

Related Information

A list of related information follows. For information on how to order or access any of this information, call your IBM representative.

IBM TPF Database Facility (TPFDF) Books

- *TPFDF Database Administration*, SH31-0175
- *TPFDF Installation and Customization*, GH31-0178
- *TPFDF Programming Concepts and Reference*, SH31-0179
- *TPFDF and TPF Structured Programming Macros*, SH31-0183
- *TPFDF Program Directory*
- *Memo to Current Licensees of IBM TPF Database Facility, TPF 4.1 and ALCS*.

Online Information

- *TPFDF Commands*
- *TPFDF Glossary*
- *TPFDF Messages (System Error, Online, Offline)*
- *TPFDF Utilities*.

How to Send Your Comments

Your feedback is important in helping to provide the most accurate and highest quality information. If you have any comments about this book or any other TPF information, use one of the methods that follow. Make sure you include the title and number of the book, the version of your product and, if applicable, the specific location of the text you are commenting on (for example, a page number or table number).

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

- If you prefer to send your comments electronically, do either of the following:
  - Go to http://www.ibm.com/tpf/pubs/tpfpubs.htm.

    There you will find a link to a feedback page where you can enter and submit comments.
  - Send your comments by e-mail to tpfqa@us.ibm.com
- If you prefer to send your comments by mail, address your comments to:

  IBM Corporation
  TPF Systems Information Development
  Mail Station P923
  2455 South Road
  Poughkeepsie, NY 12601-5400
  USA
- If you prefer to send your comments by FAX, use this number:
  - United States and Canada: 1 + 845 + 432 + 9788

– Other countries: (international code) + 845 + 432 +9788

TPFDF Product Introduction

The TPF Database Facility (TPFDF) licensed program is a database manager for application programs that run in a Transaction Processing Facility (TPF) or Airline Control System (ALCS) operating environment.

Note: ALCS is also referred to as TPF/MVS.

Traditional database management functions for these systems were the responsibility of application programmers. Throughout this publication, *traditional* means TPF or ALCS databases that are non-TPFDF. To increase the productivity of application programmers, the TPFDF product provides:

- A logical method of database organization
- A set of standardized assembler macros or C functions that form the application program interface (API)
- Central routines for database access and manipulation
- Utilities for database maintenance and testing.

After the TPFDF product is installed, application programs are no longer sensitive to the physical implementation of the database.

TPFDF Product Highlights

- **The TPFDF product is a database manager that is designed to provide:**
  - Significant increases in the productivity of an application programmer
  - Improved application program performance
  - Improved database integrity.
- **The TPFDF product is easy to install and:**
  - Allows gradual migration from a traditional database
  - Provides user exits for easy installation and migration.
- **The TPFDF product is easy to use and includes:**
  - High-level assembler application macros
  - C language functions
  - Comprehensive hardcopy and softcopy documentation
  - Education (as a separately priced option).
- **The TPFDF product is easy to maintain and provides:**
  - Centralized database handling routines
  - Utilities to maintain the database
  - Performance measurement tools.

TPFDF Features

The TPFDF Distributed Data Access (TPFDF/DDA) optional feature lets you propagate data from a TPFDF database to a DATABASE 2 database using structured query language (SQL) statements. You can collect the SQL data from one or more TPFDF files and from one or more fields in each file.

The database administrator must define (in the DBDEF macro) a file to use the DDA feature. The data propagation is transparent to the TPFDF application program.

See “The TPFDF/Distributed Data Access (DDA) Feature” on page 19 for more information about the the TPFDF/DDA feature.

TPFDF User-Specific Processing

The TPFDF product provides the following user exits that allow the described functions:

DFGDS To share TPFDF data with an MVS system.

DFTDC To implement the dialogue control facility.

You can write the code to provide these functions or obtain the necessary code from a third party.

Overview of TPFDF Benefits

TPFDF provides benefits to application programmer productivity and system management.

Application Programmer Productivity Benefits

Typically, in traditional database handling, there is no standard database organization. Therefore, there are no common routines for data retrieval, searches, sorts, or updates. Application programmers need to be aware of the size and location of data.

The TPFDF product enforces a standard for database organization. For example, it provides common routines to:

- Find and modify data
- Search, sort, and merge data
- Display data
- Read data from tape.

The TPFDF product provides high-level macros that act as an interface to these common routines. Application programmers only need to know the logical relationships of data, not the physical characteristics, to code these macros.

The TPFDF product also provides functions to allow C language programs to access TPFDF file structures.

System Management Benefits

The TPFDF product enforces a centralized database structure that provides many system management benefits. Because the definition of the database is centrally maintained, the database administrator can modify database characteristics without affecting application programs. This central maintenance, together with the utilities that the TPFDF product provides, means that the database administrator can:

- Check the integrity of the database
- Optimize application program performance
- Minimize the application program migration effort
- Minimize the application program enhancement effort
- Ease the migration process from a traditional database to a TPFDF database
- Easily define different file characteristics for different subsystems when using the multiple database function (MDBF) of the TPF High Performance Option (HPO) feature

- Easily install the TPFDF Distributed Data Access (TPFDF/DDA) feature.

TPFDF installations need a database administrator to install and manage the physical database. The database administrator communicates with both application and system programmers.

TPFDF Performance

The TPFDF product makes it possible to optimize application program performance. For example, it allows physical data to be organized to reduce direct access storage device (DASD) I/Os by:

- Selecting optimum physical record sizes
- Allocating spare space in physical records for adding data.

The TPFDF product provides data collection tools for monitoring application program performance. These tools can highlight database designs that can cause performance problems (for example, excessive DASD I/Os). The database administrator can modify the central database definitions to optimize performance. One central change can improve the performance of many application programs without modifying them.

Application Program Portability

The TPFDF product makes it easier to integrate new and existing application programs. Typically, when adding an application program, there is some data that both the existing and new application program needs to access.

Often in a traditional database, the way the data is stored is not compatible between the two application programs (for example, the data is held in different block sizes). This requires the modification of each new program that accesses the data.

With the TPFDF product, programs do not need to be modified because of the way data is stored. Block size changes and any other physical database changes are transparent to the application.

Because TPFDF application programs are independent of the physical database, they are easier to enhance.

TPFDF Migration

The TPFDF product allows gradual migration from a traditional database:

- Traditional files and TPFDF files can coexist in the same database.
- An application program can work with both traditional and TPFDF files.
- TPFDF macros and functions can retrieve and write traditional files.

This allows application programs that use the TPFDF product to coexist with application programs that use traditional database handling methods.

TPFDF Product Overview

The TPFDF product is an interface between application programs that use TPFDF macros and functions to request information from a database and the system software that physically accesses the data. It provides a uniform method for designing data layouts and writing application programs to access and read information maintained in the TPF or ALCS database.

Figure 1 shows the relative position of the TPFDF product in a TPF or ALCS operating environment.

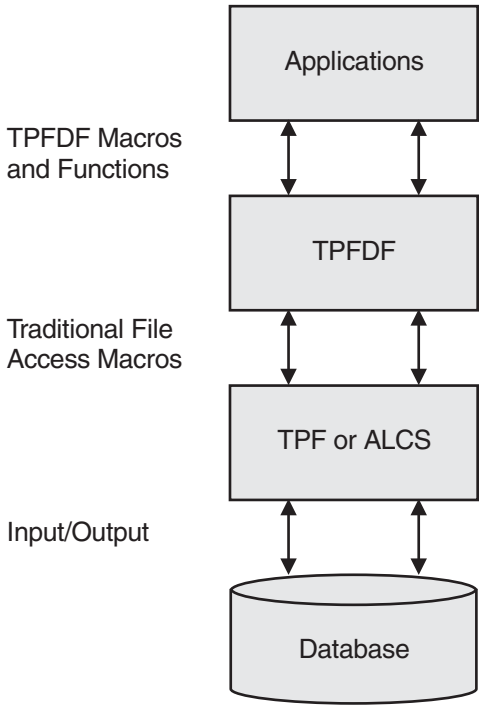

Figure 1. TPFDF Application Programs in TPF or ALCS Operating Environments

TPFDF Components

The TPFDF product provides the following software functions:

- Macros and functions for processing data on the database
- Utilities for testing and maintenance (ZUDFM commands)
- A data collection utility (ZUDFC commands)
- A database validation, capture and restore utility (ZFCRU commands)
- A recoup utility for maintaining pool space and validating references (ZRECP commands).

Database definition (DBDEF) tables contain detailed information about each file in the database. Application programs directly or indirectly use DBDEF tables.

When opening a file, the TPFDF product allocates a SW00SR slot in a work area called the database interface block (DBIFB). The next TPFDF macro or function in the application program copies relevant information from the DBDEF table into the SW00SR slot. This SW00SR slot, therefore, contains a working copy of the file

definitions while a file remains open. The SW00SR slot is closed when the file is closed. Figure 2 shows how the DBDEF table relates to the other components of the TPFDF product.

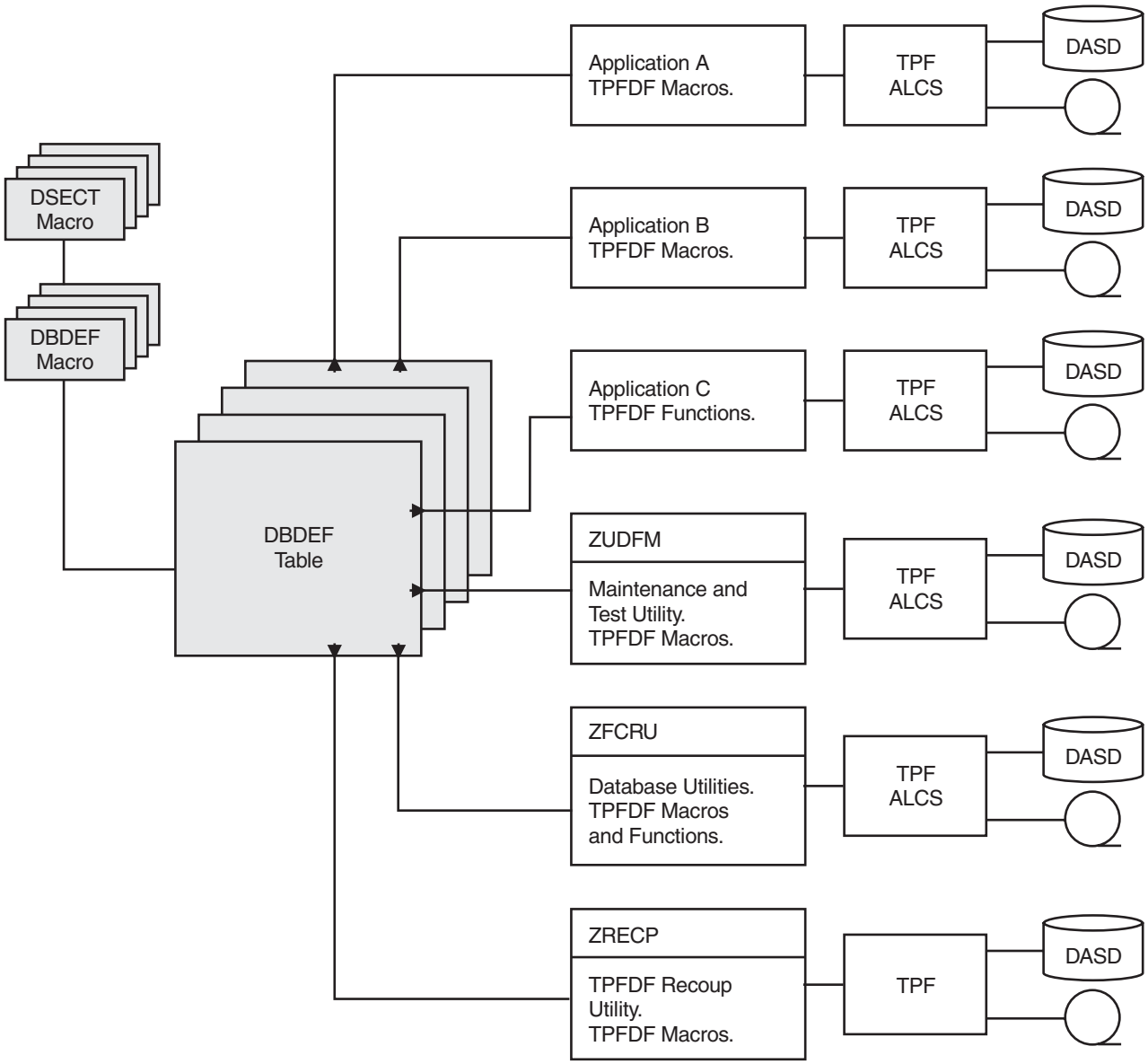

Figure 2. Relationship between the DBDEF Table and Other TPFDF Components

DBDEF Tables, DBDEF Macros, and DSECT Macros

Each DBDEF table is generated using:

- A DBDEF macro instruction with parameters that describe the file to the TPFDF product
- A DSECT macro definition that describes files and logical record layouts.

DBDEF tables provide central definitions for the database. The DBDEF tables hold information about the location, organization, and processing attributes of the database. Information about the characteristics of a file are also held in the DBDEF tables.

There is one DBDEF table for each file ID defined to the TPFDF product. For each file ID, there is also an assembler DSECT macro. This DSECT is designed by using samples provided with the TPFDF product.

The database administrator codes processing information about a file in a DBDEF macro. This information includes the file ID and the name of the DSECT. Application programmers use this name to reference the file.

A DBDEF table is the assembled output of the DBDEF macro. Load the DBDEF table to the online system. Once loaded, the table is available as read-only information to the TPFDF product. It is available both to the programs that interpret the TPFDF macros, functions, and utilities.

Database Interface Block (DBIFB)

When an application opens a file using the DBOPN macro or `dfopn` function, the TPFDF product creates a work area called the database interface block (DBIFB). The DBIFB contains several slots known as SW00SR slots, each of which contains information about an individual subfile. When a subfile or file is opened, the TPFDF product allocates a SW00SR slot in the DBIFB. The next macro or function moves the relevant information from the DBDEF table into the SW00SR slot. As processing continues, the TPFDF product returns information to the application program using various fields in the SW00SR slot. Each SW00SR slot contains the following fields that must not be changed:

Field	Description
SW00CCA	Core address of the current block.
SW00CFA	File address of the current block.
SW00FAD	File address of the prime block.
SW00ITM	Base address of the current LREC.
SW00PCA	Core address of the prime block.
SW00REC	This field also provides the base address of the current LREC.
SW00RTN	Return code from the last TPFDF macro call.
SW00RT1	Error information from the last TPFDF macro call.
SW00RT2	Additional error information from the last TPFDF macro call.
SW00SEQ	File update sequence code that can be moved into a work area for a later integrity check with DBRST macro or `dfrst` function.
SW00UKY	Last unique key supplied by the TPFDF product.
SW00WCC	Area used to hold a record code check (a value can be supplied with the DBADD or DBCRE macro).
SW00WKA	Start address of free space provided by the TPFDF product when the SPACE or SPACEB parameter is entered with the DBOPN or DBSPA macro or the `dfopn` or `dfspa` function.

There are also two fields that an application program can use. They are:

SW00USI	A 1-byte user indicator
SW00USA	A 4-byte user address field.

TPFDF Macros and Functions

Application programmers do not directly access the information in DBDEF tables. They can code high-level TPFDF assembler macros or C language functions in the application program to retrieve or manipulate data. These macro instructions call online TPFDF programs. The TPFDF programs use the information in the DBDEF tables to generate traditional TPF or ALCS file handling requests.

Table 1 summarizes the assembler macros and C language functions that are available with the TPFDF product. For more information about TPFDF macros and C language functions, see *TPFDF Programming Concepts and Reference*.

Table 1. Summary of TPFDF Macros and Functions

Macro	C Function	Description
DBADD	`dfadd`	Add a logical record to a file. DBADD and `dfadd` use designated keys to identify the location where a logical record will be inserted.
DBADR	`dfadr`	Designate a begin and end ordinal for sequential processing of the entire file.
DBCKP	`dfckp`	Write an open file in main storage to DASD.
DBCLR	`dfclr`	Allow an ECB to exit with open files.
DBCLS	`dfcls`	Complete processing and ensure that modified blocks are written to DASD.
DBCPY	`dfcpy`	Create a copy of an open file in pool files.
DBCRE	`dfcre`	Create an empty file.
DBIFB	`dfifb`	Locate the TPFDF control information for a file.
DBDIX	`dfdix`	Remove a logical record from a high-level index file for a detail file.
DBDEL	`dfdel`	Delete a logical record.
DBDSP	`dfdsp`	Produce an output message using FMSG or WTOPC.
DBFRL	`dffrl`	Verify that an entry control block (ECB) data level is not occupied by a TPFDF file.
DBIDX	`dfidx`	Create a high-level index reference for a detail file.
DBKEY	`dfkey`	Define keys.
DBMRG	`dfmrg`	Merge two input files into one output file.
DBMOD	`dfmod`	Indicate an LREC has been modified or modify LRECs that match previously established keys.
DBOPN	`dfopn`	Open a file. Initialize the file for application program use.
N/A	`dfopt`	Set options in an open subfile.
DBRED	`dfred`	Locate a logical record in a file.
DBREP	`dfrep`	Replace a logical record with a new logical record.
DBRET	`dfret`	Save a reference to the current logical record.
DBRST	`dfrst`	Restore a file from a copy to another file address.
DBSRT	`dfsrt`	Create an output file with sorted logical records.
DBSPA	`dfspa`	Allocate space for an opened file.
DBTLD	`dftld`	Write a file to a DASD location.
DBTLG	`dftlg`	Write a file to a real-time or general tape.
DBTRD	`dftrd`	Read a file from an input tape.
DBUKY	`dfuky`	Request a unique key.

TPFDF Maintenance and Test Utility (ZUDFM)

ZUDFM commands provide maintenance and test capabilities that include:

- Initializing files
- Displaying and modifying files.

See *TPFDF Commands* and *TPFDF Utilities* for more information about the ZUDFM utility.

TPFDF Data Collection Utility (ZUDFC)

ZUDFC commands allow the database administrator or system programmer who maintains the TPFDF product to gather and display statistics relating to system usage. See *TPFDF Commands* and *TPFDF Utilities* for more information about the ZUDFC utility.

TPFDF Capture/Restore Utility, Information and Statistics Environment (ZFCRU)

ZFCRU commands provide capture and restore capabilities that include:

- Capturing files
- Copying files
- Logging files
- Printing files
- Restoring files
- Validating files.

Note: The ZFCRU command is not supported in an ALCS environment. See *TPFDF Commands* and *TPFDF Utilities* for more information about the Capture/Restore Utility, Information and Statistics Environment (CRUISE).

TPFDF Recoup Utility (ZRECP)

The TPFDF product includes a recoup utility that does recoup functions for both TPFDF and traditional databases.

Note: TPFDF recoup runs as an extension to TPF recoup.

In the ALCS environment, you can chain chase TPFDF databases using the information contained in the DBDEF tables. To use the DBDEF table information during ALCS recoup, you must install the TPFDF sample user exit code. For more information about installing user exit code, see *TPFDF Installation and Customization*. See *TPFDF Commands* and *TPFDF Utilities* for more information about the ZRECP utility.

TPFDF Files and Subfiles

A TPFDF database consists of *files*. Each file contains one or more *subfiles*. The following describes the types of TPFDF files, the components of subfiles, and the methods of distributing records between subfiles.

Subfile Components

Each subfile contains a *prime block* and possibly one or more *overflow blocks*. The prime and overflow blocks contain *logical records* (LRECs). LRECs contain the actual data stored in the database.

Figure 3 shows a file that contains two subfiles. Each subfile shown in the figure has a prime block and three overflow blocks, but subfiles can have any number of overflow blocks, or none.

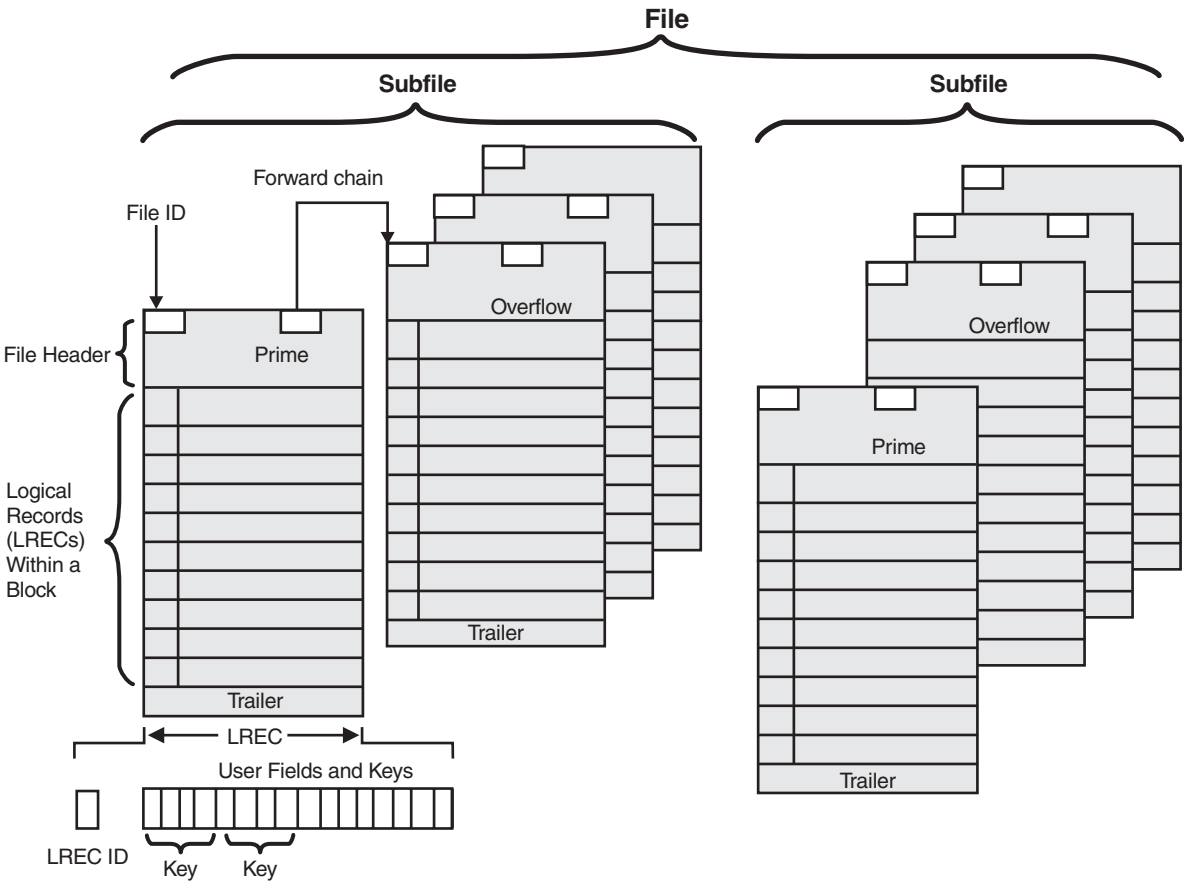

Figure 3. Logical Structure of a TPFDF File

Blocks

If the volume of data in a subfile is more than will fit into the prime block, TPFDF automatically allocates one or more *overflow blocks* to hold the extra data. It *chains* any overflow blocks to the prime block that requires them. See Figure 4 on page 12.

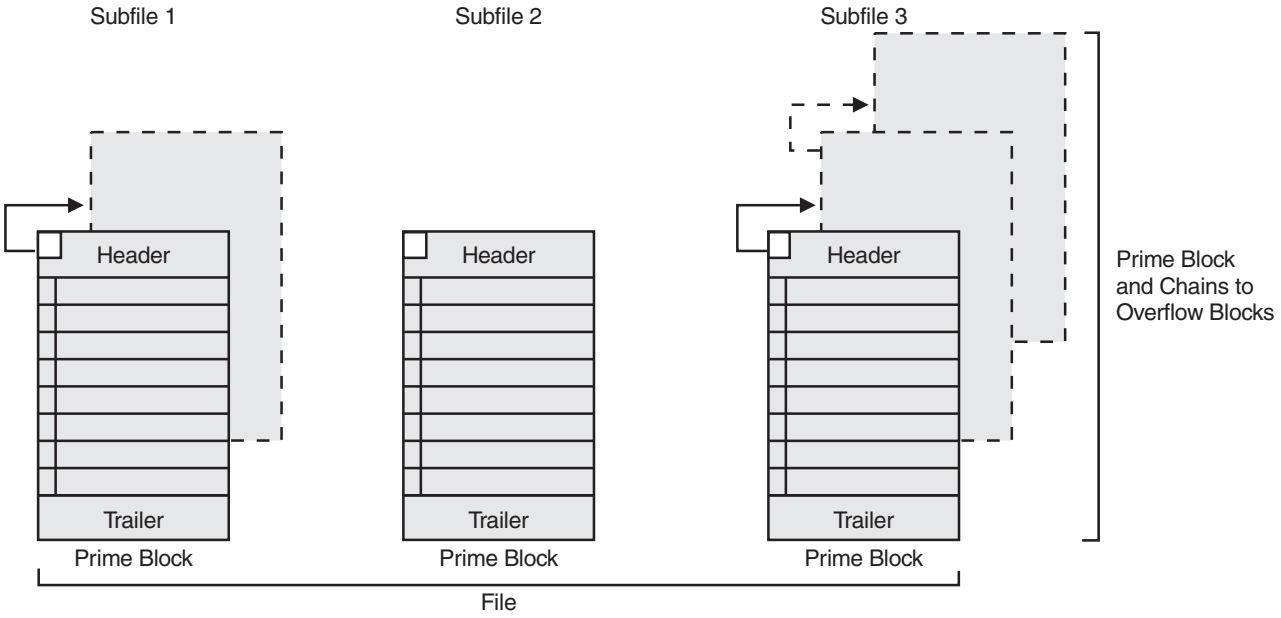

Figure 4. Prime Blocks and Overflow Blocks

All physical blocks in a file, whether they are prime or overflow blocks, have the same *file ID*. This is a 2-byte identifier that is held in the block header. (It is known in TPF systems and ALCS environments as the *record ID*.)

Block Sizes

TPFDF supports three block sizes when running with TPF. These block sizes are:

L1 381 bytes

L2 1055 bytes

L4 4095 bytes.

When running with ALCS, TPFDF supports eight block sizes, L1 to L8. ALCS block sizes are defined during ALCS installation. The L1, L2, and L4 block sizes are usually defined to be the same as the L1, L2, and L4 TPF block sizes.

The database administrator defines the block sizes for any TPFDF file. Prime and overflow blocks can be different sizes.

Block Headers and Trailers

Each block contains a *header* and an optional *trailer*. The block header consists of the following:

- A standard 16-byte TPF or ALCS portion
- A TPFDF extension to the header (10 bytes)
- An optional additional extension (of any length) that the application program can use for any purpose.

The optional block trailer is 36 bytes long and contains control information such as the last command issued and the date and time the block was last updated.

The trailer can provide useful information for debugging programs. When using trailers, at least 62 bytes of each block are reserved for TPFDF use. The trailer can also include an optional extension (of any length) that the application program can use for any purpose.

Each overflow block has the same amount of space reserved for the header and optional trailer as the prime block. (The header and trailer information can be different in prime and overflow blocks.)

Ordinal Numbers

The number of prime blocks in a fixed file is also called the number of *ordinals* in the file. The file has absolute start and end ordinal numbers that are used by TPFDF macros.

LRECs

The internal format of individual TPFDF files does not correspond to the internal formats of records in a traditional TPF or ALCS database. In every TPFDF file, the data is organized into logical groups called logical records (LRECs). Each logical record has mandatory items that allow TPFDF to recognize and work with it. These items include a logical record identifier (LREC ID) and, for variable-length and extended LRECs, the size of the LREC.

An LREC is the smallest unit of data that an application program can read, add, or delete. LRECs have LREC IDs that are used as *primary keys* when reading LRECs from, or adding LRECs to, a subfile.

LRECs cannot span blocks. Their size is limited to the physical block size, minus the length of the following components (the block header, optional block header extension, optional trailer, and optional trailer extension).

Figure 5 shows how LRECs are held in blocks.

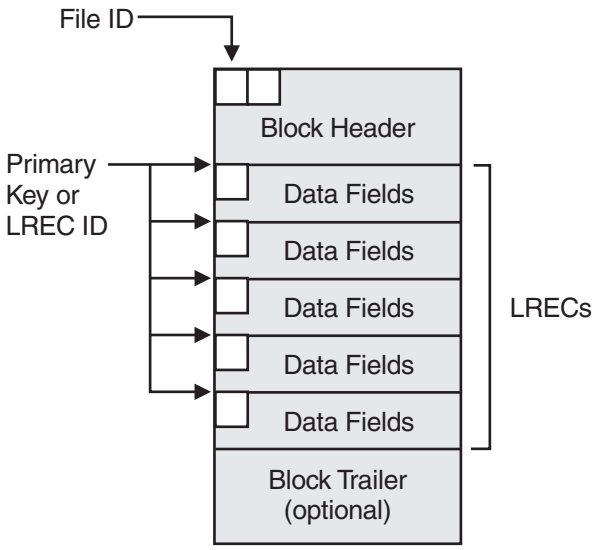

Figure 5. LRECs in a Block

LREC Types

LRECs can be:

- Fixed-length
- Variable-length
- Extended.

LREC Fields

A *field* is a subdivision of an LREC that contains an item of data. Fields hold one of the following:

- LREC ID
- User data; for example, a customer name, an address, or a balance amount
- Control information; for example, the length of the LREC (if it is variable-length or extended).

In variable-length (and extended) LRECs, user fields can vary in length.

Distributing LRECs between Subfiles

TPFDF lets you read LRECs from, and add LRECs to, a subfile in any file in the database, without having to worry about the physical structure of the file. You do need to know, however, how the file is split into subfiles, and what type of index support (if any) is being used with the file.

In a file, LRECs are distributed and accessed as follows:

- Algorithms
- Basic index support
- Block index support
- B^+Tree index support.

Algorithms

If a file contains more than one subfile, TPFDF must be able to calculate into which subfile any particular LREC belongs. The database administrator supplies this information as an *algorithm*. TPFDF uses this algorithm and the *algorithm argument*, passed as a parameter with a TPFDF macro or function, to determine where an LREC belongs.

For example, use an algorithm to divide a small customer file into 26 subfiles where each subfile contains the LRECs for a group of customers who all share the same initial surname letter: A, B, C, ..., Z.

The database administrator gives TPFDF this information by specifying a particular algorithm in the DSECT macro for the file. In the example, the required algorithm is #TPFDB01.

Provide an *algorithm argument* as a macro or function parameter to add LRECs to the file. In the example, the argument is the first letter of the customer name. TPFDF uses the algorithm argument, together with the algorithm specified for the file (#TPFDB01), to calculate the appropriate subfile for the LREC that contains details about this customer.

There are many different algorithms from which to choose. They are explained in *TPFDF Database Administration*.

Basic Index Support

Algorithms provide one method of distributing LRECs between subfiles; basic index support provides another. With the simplest type of basic index support, TPFDF uses an LREC in one file, the *index file*, to point to (reference) a subfile in another file, called the *detail file*.

Figure 6 shows a basic index support structure for an application program that processes customer data. For example, the application program is processing information for a customer named JONES. TPFDF searches the high-level index to find the reference to the detail file for JONES.

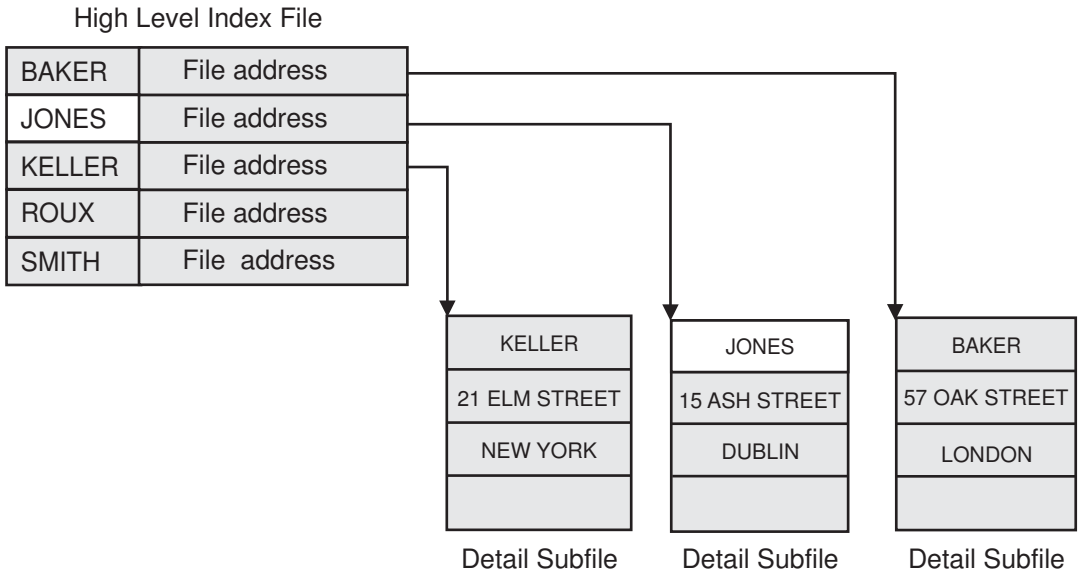

Figure 6. TPFDF Basic Index Support

Basic index support is transparent to the application program. The application program does not depend on the storage structure or the access path to the data. The actual steps involved in retrieving or building an indexed file are done automatically by TPFDF.

Indexed file structures are an effective way to store large, variable quantities of data. TPFDF supports:

- Multiple high-level index files (for example, a high-level index file points to another high-level index file that points to a detail file)
- Multiple high-level index files pointing to the same detail file
- A single high-level index file pointing to multiple detail files.

Block Index Support

Block index support optimizes retrieval of data in a TPFDF file that has many chains. Figure 7 on page 16 shows a block index structure for an application program that processes customer data. For example, the application program is processing information for a customer named FOX.

Without the block index facility, TPFDF searches the prime block and five chained blocks to find the data. With the block index facility, TPFDF searches the prime block to find the direct reference to the block that contains the data.

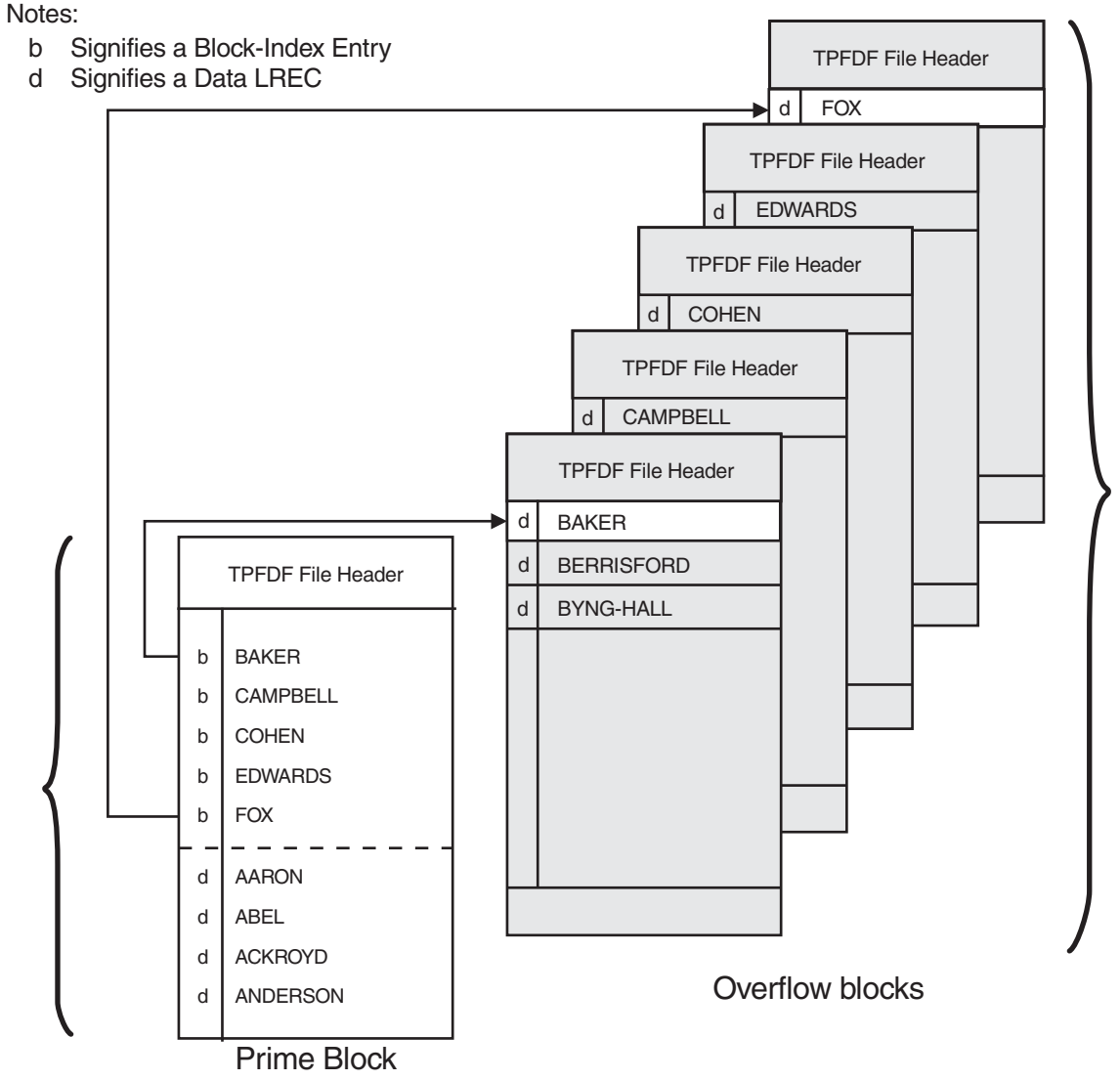

Figure 7. TPFDF Block Index Facility

B+Tree Index Support

B+Tree index support is similar to block index support. Like block indexing, B+Tree index support uses index LRECs (also known as technical LRECs or TLRECs) to identify the first data LREC contained in each block of a subfile. Unlike block indexing, these TLRECs are not maintained in the prime block of the subfile. Instead, these TLRECs are maintained in a separate B+Tree structure.

Instead of reading all the LRECs in a subfile until it locates the desired LREC, TPFDF can go directly from the B+Tree structure to the block containing the requested LREC. LRECs are then searched sequentially in that block.

The B+Tree structure, maintained by TPFDF, is transparent to the application program. The database administrator must specify that a file is using B+Tree indexing in the DSECT and DBDEF statements. TPFDF then maintains technical

logical records (TLRECs) in the B+Tree structure associated with the subfile. Every TLREC in the lowest level of the B+Tree structure points to a data block of the subfile.

For more information about B+Tree indexing, see *TPFDF Database Administration*.

Structure of a File That Uses B+Tree Indexing

A data file that uses B+Tree indexing has a B+Tree index file associated with it. The data file consists of data blocks that contain LRECs. The B+Tree index file consists of node blocks that contain TLRECs.

Figure 8 shows data file GR91SR, which uses B+Tree index file IR73SR. Data file GR91SR shows nine data blocks. B+Tree index file IR73SR shows a root node and three leaf nodes.

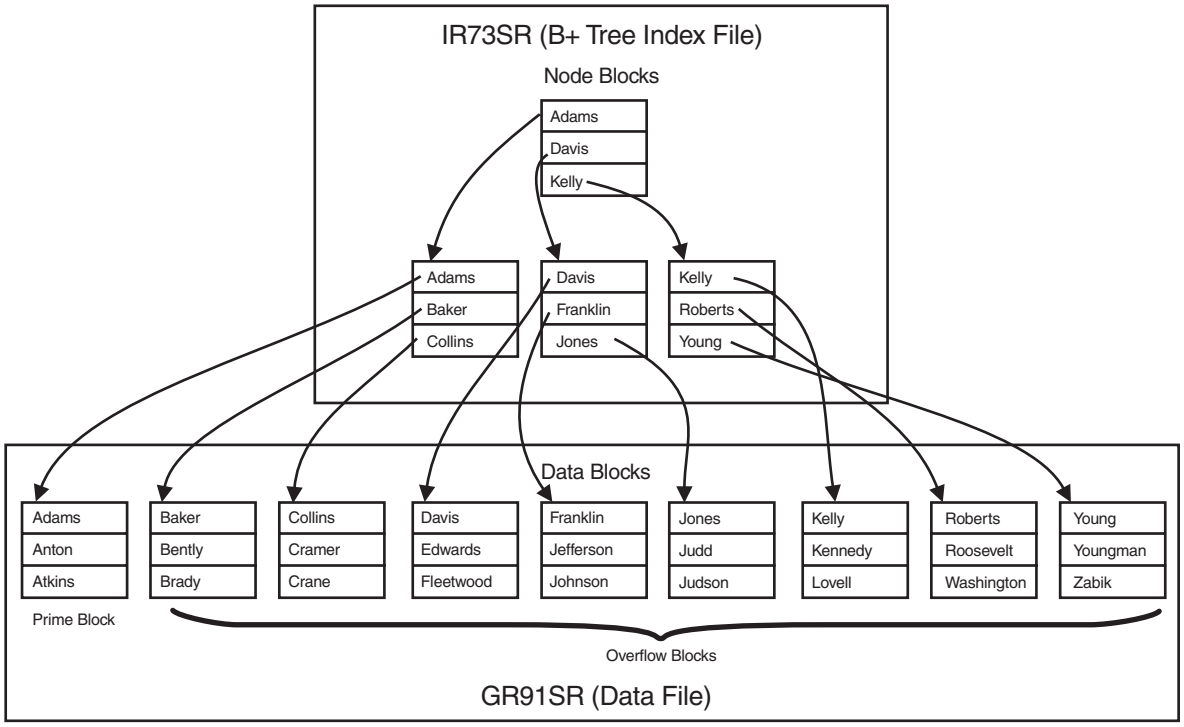

Figure 8. Sample B+Tree File

TPFDF File Types

There are three types of physical files in TPFDF:

- Fixed files
- Miscellaneous files
- Pool files.

Fixed Files

The database administrator defines each fixed file as consisting of a fixed number of blocks on a specific part of DASD. When a fixed file has been allocated, it contains only prime blocks, so it contains exactly as many blocks as there are subfiles in the file.

When an application program adds LRECs to a subfile in a fixed file, the prime block may become full. When this happens, TPFDF obtains a free pool block and creates a chain from this to the fixed prime block that overflowed.

Fixed files are suitable when the number of subfiles is known in advance and it is unlikely that many LRECs will overflow into pool blocks.

Miscellaneous Files

A miscellaneous file is a type of fixed file. The database administrator normally defines some miscellaneous fixed files on DASD with record types #MISCS, #MISCL, and #MISC4. The last character in the name identifies the size used for prime blocks in the subfile.

Use miscellaneous files for small amounts of data that require a fixed file.

Pool Files

Pool files consist of pool blocks that are used as prime blocks and overflow blocks in a file. Pool blocks can be *short-term* or *long-term*. Short-term pool blocks are likely to be reused by a TPF system or ALCS environment in seconds or minutes. Long-term pool blocks have an indefinite lifetime.

Many TPFDF macros, functions, commands, and indexing support use long-term pool files. To use long-term pool files for permanent storage, save the file address of each prime block in a fixed file. You cannot use short-term pool files for permanent storage.

The TPFDF/Distributed Data Access (DDA) Feature

The TPFDF/DDA feature propagates data from a TPFDF database (hierarchical) to a DATABASE 2 (DB2) database (relational).

The database administrator defines the correlation between fields in the hierarchical database (index keys, index levels, and so on) and fields in the relational database (data row, column, and so on). Extracting fields from the TPFDF file (or files) to build the SQL data row is defined centrally for any TPFDF file that contains data to be propagated.

When any file-modification command is detected, the central definition is checked to determine what action to take. This action can be as simple as ignoring LRECs with a particular ID (by not defining any actions for that ID) or can be as complicated as extracting index keys at each indexing level together with multiple fields in other files.

Figure 9 shows an SQL row built from fields in an index file (1) , a field in its detail file (2) , and a field in a different detail file (3). The extracted data can be a complete field or part of a field.
Figure 10 on page 20 shows a simple one-to-one relationship between the TPFDF

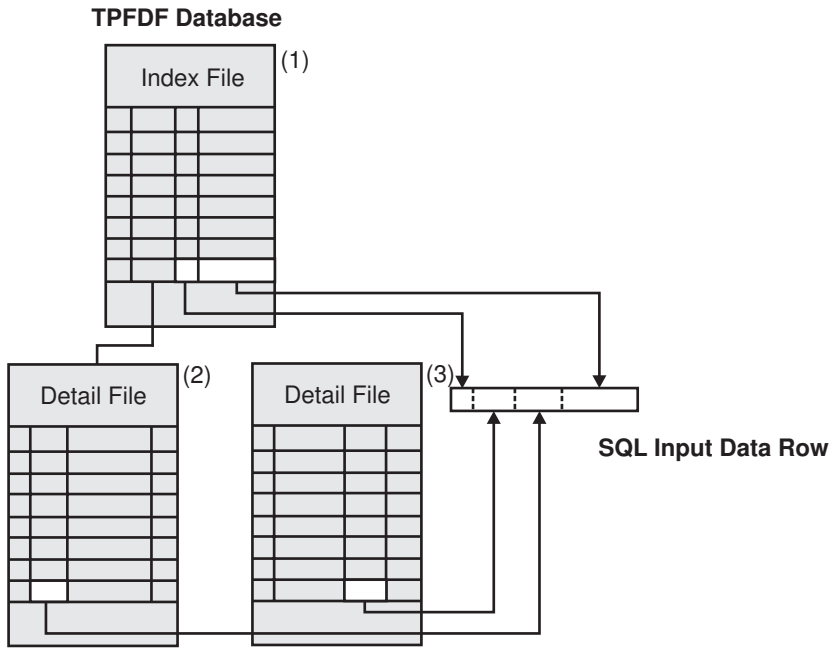

Figure 9. Data Extraction: From Index Files and Multiple Files

LREC data and the DATABASE 2 SQL data row.

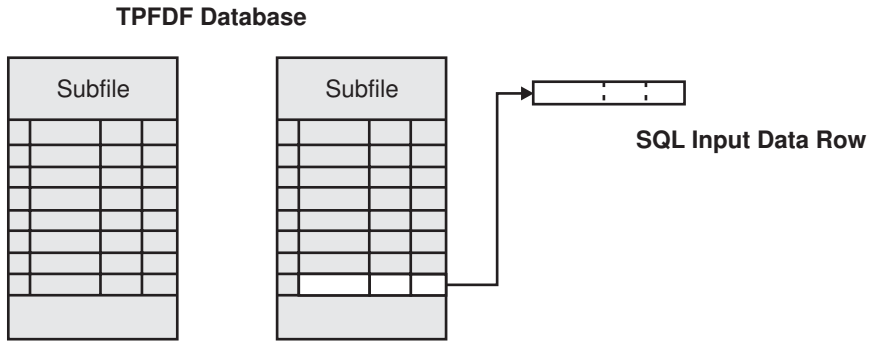

Figure 10. Data Extraction: One-to-One Relationship between the Data Fields and Rows

TPFDF Implementation Considerations

Implementing TPFDF is essentially the same process as implementing any new application program for the TPF system or ALCS environment. It consists of:

- Installing TPFDF programs
- Installing TPFDF macro definitions
- Adding records to the database for TPFDF use.

See *TPFDF Installation and Customization* for more information.

There are also some implementation considerations that are specific to TPFDF. These include:

- Appointing a TPFDF database administrator with experience in designing and implementing databases. The role of the database administrator includes:
  - Assisting the application programmer in designing a database to meet the requirements of the application program
  - Advising the application programmer about how to code TPFDF macros for maximum performance
  - Acting as a coordinator for system and application programmers in implementing the database.
- Defining procedures to design, implement, and manage TPFDF files. These procedures include:
  - Defining the administrative process for developing the TPFDF database
  - Clarifying application programmer, system programmer, and database administrator responsibilities for this process.
- Assessing database migration options. These include:
  - Migrating the entire database to a TPFDF database
  - Retaining a traditional database for existing application programs while implementing a TPFDF database for new application programs
  - Writing new application programs and modifying existing application programs to use both traditional and TPFDF databases.

TPFDF can operate in a multiprocessor environment. TPFDF supports:

- The loosely coupled facility of the TPF High Performance Option (HPO) feature
- The multiple database function (MDBF) of the TPF HPO feature.

System Requirements

The following describes the system requirements for the TPFDF product.

Hardware Requirements

The TPFDF product requires the same hardware configuration as the operating system. It requires additional storage for the TPFDF application programs and tables.

Software Requirements

TPF Version 4 Release 1 (TPF 4.1) users of the TPFDF product must apply all APARs that support your level of the TPFDF product.

ALCS Version 2 Release 1 users of the TPFDF product must apply all APARs that support your version of the TPFDF product. To use C functions, B^+Tree, and global modification support, ALCS users must also install the associated programming requirements for C language support

Note: For more information about the TPFDF APARs that you need to apply to your system, see the *TPFDF Program Directory* and the *Memo to Current Licensees of IBM TPF Database Facility, TPF 4.1 and ALCS*.

Migration and Coexistence

Application programs that use traditional databases can coexist with TPFDF application programs without modification.

The TPFDF product includes structured programming macros (SPMs). These macros can coexist with TPF SPMs. For more information about SPMs, see *TPFDF and TPF Structured Programming Macros*.

If you are upgrading from earlier releases of the TPFDF product, you only need to reassemble the segments that contain DBDEF macro statements. You do not need to reassemble application programs.

To use the optional Distributed Data Access (DDA) feature you must:

- Reassemble the DBDEFs
- Reassemble application programs that use the DBMOD macro.

Index

T

U

Z

File Number: S370/30XX-20
Program Number: 5706-196

Printed in U.S.A.

GH31-0177-02