DB2/400

CBSI COMPLETE BUSINESS SOLUTIONS (INDIA) PRIVATE LIMITED

SYSTEMS INTEGRATION PARTNERS

DB2/400 Index

Chapter	Description	Page No.
1...	THE DATA MANAGER	
	OS/400 - The Operating System	1.1
	System setup	1.2
	Field, Record and File definitions	1.2
	Files	1.3
	Open a file in other systems	1.4
	Read and write flow	1.4
2.	SOURCE FILES	
	Using source files	2.1
	Creating a source file	2.1
	IBM-supplied-source files	2.2
	Source file attributes	2.3
	Using the source entry utility	2.4
	Using source files in a program	2.4
	Creating an object using a source file	2.5
	Creating an object from source statements in a batch job	2.6
	Determining which source file member was used to create an object	2.6

3. PHYSICAL FILES

Structure fundamentals..... 3.1

Data members.....3.2

Types of files.....3.2

Database files..... 3.3

Source files.....3.4

Device files.....3.4

DDM files..... 3.5

Save files.....3.5

Describing database files

Records in database files can be

described in two ways..... 3.6

Programs can use file description

in two ways.....3.6

Dictionary described data.....3.8

Methods of describing data to

the system.....3.8

Methods of describing a database file

to the system..... 3.8

Describing the record format..... 3.9

Describing the access path..... 3.9

Naming conventions.....3.9

Describing database files using DDS.....3.10

Physical file keywords..... 3.10

File level keywords.....3.10

Record-format level keywords..... 3.11

Key-field level keywords..... 3.12

Example of defining a Physical file

using DDS..... 3.13
Example of using a logical file
using DDS..... 3.15
Field reference file..... 3.18
Record Format sharing limitation..... 3.25
Access path for the file..... 3.26
Arranging key fields using the SRTSEQ
parameter..... 3.30
Arranging key fields in ascending or
descending sequence..... 3.32
Using more than one key field..... 3.33
Preventing duplicate key values..... 3.35
Arranging duplicate keys..... 3.37
Using existing access path specifications..... 3.42
Database files and member attributes
- Introduction..... 3.42
FILE and MBR parameters..... 3.42
DTAMBRS Parameter..... 3.42
SRCFILE and SRCMBR parameters..... 3.43
FILETYPE parameter..... 3.43
MAXMBRS parameter..... 3.43
UNIT parameter..... 3.43
UNIT parameter tips..... 3.44
FRCRATIO parameter..... 3.44
FRCRATIO parameter tip..... 3.45
FRCACCPTH parameter..... 3.45
FRCACCPTH parameter tips..... 3.45
LVLCHK parameter..... 3.45
Level Check Example..... 3.45

MAINT parameter..... 3.46
MAINT parameter tips..... 3.47
RECOVERY parameter..... 3.48
RECOVERY parameter tip..... 3.48
SHARE parameter..... 3.48
WAITFILE and WAITRCD parameters..... 3.49
AUT parameter..... 3.49
SYSTEM parameter..... 3.49
TEXT parameter..... 3.49
CCSID parameter..... 3.49
SRTSEQ parameter..... 3.49
LANGIG parameter..... 3.50
Creating a physical file..... 3.51
CRTPF FILE..... 3.51
Physical file and member attributes..... 3.51
EXPDATE parameter..... 3.51
Size of the physical file member..... 3.52
Storage allocation..... 3.53
Method of allocating storage..... 3.53
RCDLEN parameter..... 3.53
DLTPCT parameter..... 3.54
REUSEDLT parameter..... 3.54
ALWUPD and ALWDLT parameters..... 3.55
SRCTYPE parameter..... 3.55

4. LOGICAL FILES

Record format definition / physical
file selection..... 4.1

Describing logical file record formats..... 4.1
Logical file keywords..... 4.1
File level keywords..... 4.2
Record format level keywords..... 4.2
Join level keywords..... 4.3
Field level keywords..... 4.3
Key field level keywords..... 4.4
Select / Omit level keywords..... 4.4
Describing field use for logical files..... 4.7
Deriving new fields from existing fields..... 4.9
Describing access paths for logical files..... 4.12
Selecting and omitting records using logical files..... 4.13
Using existent access paths..... 4.17
Creating a logical file..... 4.18
Creating a logical file with more than one
record format..... 4.19
Concept of join..... 4.22
Characteristics of a join logical file..... 4.22
Basic concepts of joining two physical files..... 4.24
Reading a join logical file..... 4.29
Matching records in Primary and secondary
files (Different cases)..... 4.31
Setting up a join logical file..... 4.38
Using more than one field to join files..... 4.39
Reading duplicate records in secondary files.....4.42
Using join fields whose attributes are different..... 4.44
Describing fields that never appear in the
record format..... 4.46
Application where it can be used..... 4.48

Specifying key fields in join logical files..... 4.48
Specifying select / omit statements in
join logical files..... 4.50
Joining three or more physical files..... 4.51
Joining a physical file to itself..... 4.53
Using default data for missing records
from secondary files..... 4.55
A complex join logical file..... 4.58
Summary of rules for join logical files
Requirements..... 4.62
Join fields..... 4.63
Fields in join logical files..... 4.64

5. MANAGING DATABASE FILES

Introduction..... 5.1
Member operations common to all database files..... 5.1
Adding members to files..... 5.1
Changing member attributes..... 5.2
Renaming members..... 5.2
Removing members from files..... 5.2
Initializing data in a physical file member..... 5.2
Clearing data from physical file members..... 5.4
Reorganizing data in physical file members..... 5.4
Displaying records in a physical file member..... 5.4

6. CHANGING DATABASE FILE DESCRIPTIONS AND ATTRIBUTES

Effect of changing fields in a file description..... 6.1
Changing a physical file description and attributes.....6.2
Changing a logical file description and attributes..... 6.5

7. USING DATABASE ATTRIBUTES AND CROSS - REFERENCE

Displaying attributes for a file..... 7.1
Displaying the relationships between the files on the system.....7.2
Displaying the files used by programs..... 7.2
Displaying the system cross-reference files..... 7.2

8. PHYSICAL FILE CONSTRAINTS

Unique constraint..... 8.1
Primary key constraint..... 8.1
Adding unique and primary key constraints..... 8.2
Removing constraints..... 8.2
Working with physical file constraints..... 8.3
Displaying check pending constraints..... 8.5
Processing check pending constraints..... 8.6
Physical file constraint considerations and limitations..... 8.8

9. REFERENTIAL INTEGRITY

Introducing referential integrity and constraints.....9.1
Referential integrity terminology..... .. 9.1
A simple referential integrity example..... 9.3
Creating a referential constraint..... 9.3
Constraint rules..... 9.4
Defining the parent file.....9.6
Defining the dependent file.....9.8
Verifying the referential constraints.....9.8
Referential integrity enforcement.....9.8
Foreign key enforcement..... 9.8
Parent key enforcement.....9.9
Check pending..... 9.10
AS/400 functions affected by referential integrity.....9.14
Referential constraint considerations and limitations... 9.17
Constraint cycles..... 9.18

10. HANDLING DATABASE FILE ERRORS IN A PROGRAM

NOTES..... 10.1

THE DATA MANAGER

OS/400 - THE OPERATING SYSTEM

OS/400, the software operating system for the AS/400 machine, together with the hardware , does all data management, This comprises creating, storing, and accessing data for applications, and means data for or from :

- Database
- Tape
- Another computer system's database
- Another computer system's devices
- Diskette
- Spool queues when needed
- Printers

The command language, CL, and the data definition language, DDS, ~~are used to create Data definition language, DDS~~ , are used to create the object definitions data management uses to run the system for you. Your program or display does not run the system- it only makes request of the system to do work for it. You use the SEU (Source Entry Utility) to enter the statements to create these objects. The CL commands have parameters from which you choose to define the attributes of the objects (for example, how much space the system should allocate for a data file when first created and how much additional space to grab when a user tries to expand the file beyond the original definition by adding one too many records to it). All of this is prompted by the system; you do not have to memorize DDS or CL in order to use either language to create your working environment.

SYSTEM SETUP

Although the Data Manager is built in , you must tell it how you plan to use the system. You define the database in enough detail to get started. Using CL, you "create" device "files" as well as the database "files" according to your own specifications. You do this once for the system rather than once per program as in other systems. You give a name to every object you create; it is this name a program will use to find the object; be it a printer, another system, or a database item . Later on, when your program is using these files of data , the Data manager will do the all quering and answering of data requests, the invoking of application of application programs when input has arrived from devices or database, the device control itself, all the spooling of input and output to synchronize the use of system devices, and work storage allocation and management. In other words, the data manager does all the navigation through the system that used to be within a user program, in a way that share resources and thus allows more than one data user to access a pathway to and from a data type.

All users are controlled by the system rather than the other way around. This is a significant change from other systems you may know. Not only does it open up the database to concurrent applications without the risk of bottlenecks, but it also opens the system up over time to changes in environment and use. There is never the need to redesign the database in order to add a new application because the database is not designed for or by an application but by the system itself, and program access is optimized as a function of system load, not programmer ingenuity. The word is shared, and it applies over space and over time. No new use or to interface to data will disable an older use or interface is an object, self-contained and independent, managed as one of many system objects according to the system's and object's priorities.

FIELD, RECORD AND FILE DEFINITIONS

A record is the unit of transfer from file to system to program. At a minimum, you define to the system, using DDS, the length of the record In bytes. If that is all you do, the

system will hand you the entire record on a data request, without breaking it down or doing specific field editing or verification. With added field definition the system will, with your request for data:

- Rearrange fields as specified In your data request.
- Edit fields differently for different data requests or output devices.
- Check input data field by field, according to prespecified field definition parameters.
- Sequence input to programs or output to screens by fields selected at run time
- Hide certain fields from a program's or output to screens by fields elected at run time
- Map data fields into your program or onto your screen differently from the actual Record format.

A record type is defined within a FILE, which is an unordered set of like records that is sequenced only if you specify. Sequencing of records occurs at usage time and according to the path control you use to access the data. retrieval sequence can be through a run time choice of "logical view", the so-called LOGICAL FILE of the AS/400, or through SQL or AS/QUERY or other inquiry type programming language requests, or by time-of-arrival of the record into the database. Unlike in systems left over from the era of batch programming, you should never read every record into your program and personally sift out the records you want to see, discarding the ones you want. This would be a terrible tax on the system and on the programmer.

FILES

You use FILE definitions to define to the system the sets of data it will house and to define each device on which the data will be presented. It is in these file definitions that the method of data presentation and retrieval is determined. For displays, for instance, you define as many FILES as you have output or input formats for each day type:

The data manager provides all the data security and locking/sharing, and the FILE definitions provide the presentation sifting and editing and layouts for each new

application. You never have to be concerned with what is already running on the system or with entanglements with other users data requests. These are major stumbling blocks to fast application growth on older systems.

OPEN A FILE IN OTHER SYSTEMS

On earlier systems a program had to "open" a file before requesting data. The open statement in a COBOL or other program was the place of connection: Execution of OPEN in my program caused the program to lock to a data file-"file" meaning both hardware medium and data type. In order to connect my program had to house the exact file specifications, down to exact record and field definitions, which had to match exactly the physical format of the data and its storage medium. To err here in the program was to cause a bug, meaning the program wouldn't work . This is called early binding.

The AS system binds later. The system doesnot have to know the exact length of fields or records until it actually does a read for you. This means you can supply such information outside your program, using FILE objects that resides outside of the program, which you select for your program's use some time before an actual record is needed. Opens are still used in COBOL programs but they serve a different purpose now. They signal the system to prepare the paths for data retrieval based on the logical and physical views thus you have invoked this run time. In this way OPEN speeds up performance on random retrieval by having the path ready when the program makes a singular request for data. It sees to it that the FILE objects is ready to use, the data is available , and shifting out of chosen records for program viewing is already done. The open of a DDM FILE also verifies that the path to another system , including modems, controllers, and lines , is ready to use and that the data there is available . It also does a large part of security clearance.

READ AND WRITE FLOW

The read and write commands cause an entire chain of events. On a write to display operation:

1. The screen is cleared.
2. The data from buffer is formatted according to record formats in the display FILE object.
3. The formatted data is written to display.
4. The input-capable fields on display are readied as defined in the display file object; the keyboard is unlocked.

On a read operation.

1. The keyboard is unlocked.
2. Input fields are validated per DDS for the display file.
3. Response attributes and data pass to the application.
4. The screen clears per DDS
5. The keyboard locks, awaiting application okay to data.

On write / read the system does the above two requests, locking the display between the write and read operations.

INVITE is a special display command that talks to groups of terminals as a unit, waiting for a response from any terminal. It allows the application program to continue until any screen signals that data has been entered for processing by this application , at which point the application is switched to its read sequence. The user's input area will show the device name and the results of the operations well as data.

This kind of operation requires multithreaded and reentrant application and system code, but you have probably guessed by now only that this is the only code that the AS/400 creates, since it could not efficiently handle all the locking and sharing required by database usage otherwise. This means all work areas are provided by the data manager and that your program requests are all queued, prioritized, and scheduled by the system, as is all work on the system , including work by and for the system itself. The subsystem is a user just as you are, but of a higher priority most times.

SOURCE FILES

Using source files

A source file is used [a command alone cannot supply sufficient information for creating an object] It contains input data needed to create some types of objects.

To create a logical file, you must use a source file containing (DDS).

To create the following objects, source files are required:

- High-level language programs
- control language programs
- Logical files
- Intersystem communications function files
- Commands
- Translate tables

To create the following objects, source files can be used, but are not required:

- Physical file
- Display file
- Printer file

A source file can be a database file, diskette file, tape file. An inline data file. A source database file is simply another type of database file. You can use a source database file like you would use any other database files on the system.

Creating a source file

To create a source file, you can use the create source physical file (CRTSRCPF), create physical file(CRTPF), or create logical file(CRTLF) command.

Normally you will use the CRTSRCPF command to create a source file, because many of the parameters default to values that you usually want for a source file.

The CRTSRCPF command creates a physical file, but with attributes appropriate for source physical file. For example, the default record length for a source file is 92(80 for

the source data field, 6 for the source sequence number field, and 6 for the source date field)

IBM-Supplied source files

For your convenience, the OS/400 program and other licensed programs provide a source file for each type of source. These source files are:

File Name	Library name	Used to create
QBASSRC	QGPL	Basic programs
QCBLSRC	QGPL	System/380 compatible COBOL
QCSRC	QGPL	C Programs
QCLSRC	QGPL	CL programs
QCMDSRC	QGPL	Command definition statements
QFTNSRC	QGPL	FORTRAN Programs
QDDSSRC	QGPL	Files
QFMTSRC	QGPL	sort source
QLBLSRC	QGPL	COBOL/400 programs
QS36SRC	#LIBRARY	system/36 compatible COBOL programs
QAPLISRC	QPLI	PL/I Programs
QPLISRC	QGPL	PL/I Programs
QREXSRC	QGPL	Procedures language 400/REXX
QRPGSRC	QRPG	RPG/400 Programs
QARPGSRC	QRPG38	system/38 environment RPG
QRPG2SRC	#RPGLIB	system/36 compatible RPG II
QS36SRC	#LIBRARY	system/36 compatible RPG II(after install)
QPASSRC	QPAS	Pascal programs
QTBLSRC	QGPL	Translation table
QTXTSRC	QPDA	Text

You can either add your source members to these files or create your own source files.

The IBM-supplied source files are created with the file names used for the corresponding create command (for example, the CRTCLPGM command uses the QCLSRC file name as the default). Additionally, the IBM-supplied programmer menu uses the same default names. If you create your own source files, do not place them in the same library as the IBM-supplied source files.

Source File Attributes

Source files usually have the following attributes:

- A record length of 92 characters (this includes a 6-byte sequence number, a 6-byte date, and 80 bytes of source).
- Keys (sequence numbers) that are unique even though the access pathn does not specify unique keys. You are not required to specify a key for a source file. Default source files are created without keys (arrival sequence access path). A source file created with an arrival sequence access path requires less storage space and reduces save/restore time in comparison to a source file for which a keyed sequence access path is specified.
- More than one member.
- Member names that are the same as the names of the objects that are created using them.
- The same record format for all records.
- Relatively few records in each member compared to most data files.

Some restrictions are:

- The source sequence number must be used as a key, if a key is specified.
- The key, if one is specified, must be in ascending sequence.
- The access path cannot specify unique keys.
- The ALTSEQ keyword is not allowed in DDS for source files.

- The first field must be a 6-digit sequence number field containing zoned decimal data and two decimal digits.
- The second field must be a 6-digit date field containing zoned decimal data and zero decimal digits.
- All fields following the second field must be zoned decimal or character.

Using the Source Entry Utility (SEU)

You can use the Source Entry Utility (SEU), to enter and change source in a source file. If you use SEU to enter source in a database file, SEU adds the sequence number and date fields to each source record.

If you use SEU to update a source file, you can add records between existing records. For example, if you add a record between records 0003.00 and 0004.00, the sequence number of the added record could be 0003.01. SEU will automatically arrange the newly added statements in this way.

When records are first placed in a source file, the date field is all zoned decimal zeros (unless DDS is used with the DFT keyword specified). If you use SEU, the date field changes in a record when you change the record.

Using Source Files in a Program

You can process a source file in your program. You can use the external definition of the source file and do any input/output operations for that file, just as you would for any other database file.

Source files are externally described database files. As such, when you name a source file in your program and compile it, the source file description is automatically included in your program printout.

For example, assume you wanted to read and update records for a member called FILEA in the source file QDDSSRC. When you write the program to process this file, the system will include the SRCSEQ, SRCDAT, and SRCDTA fields from the source file.

Note: You can display the fields defined in a file by using the Display File Field Description command (DSPFFD).

The program processing the FILEA member of the QDDSSRC file could:

- Open the file member (just like any other database file member).
- Read and update records from the source file (probably changing the SRCDTA field where the actual source data is stored).
- Close the source file member (just like any other database file member).

Creating an Object Using a Source File

You can use a create command to create an object using a source file. If you create an object using a source file, you can specify the name of the source file on the create command.

For example, to create a CL program, you use the Create Control Language Program (CRTCLPGM) command. A create command specifies through a SRCFILE parameter where the source is stored.

The create commands are designed so that you do not have to specify source file name and member name if you do the following:

- Use the default source file name for the type of object you are creating.
- Give the source member the same name as the object to be created.

For example, to create the CL program PGMA using the command defaults, you would simply type:

CRTCLPGM PGM(PGMA)

Creating an Object from Source Statements in a Batch Job

If your create command is contained in a batch job, you can use an inline data file as the source file for the command. However, inline data files used as a source file should not exceed 10,000 records.

Determining Which Source File Member Was Used to Create an Object

When an object is created from source, the information about the source file, library, and member is held in the object. The date/time that the source member was last changed before object creation is also saved in the object.

The information in the object can be displayed with the Display Object Description (DSPOBJD) command and specifying DETAIL(*SERVICE).

This information can help you in determining which source member was used and if the existing source member was changed since the object was created.

You can also ensure that the source used to create an object is the same as the source that is currently in the source member using the following commands:

- The Display File Description (DSPFD) command using TYPE(*MBR). This display shows both date/times for the source member. The Last source update date/time value should be used to compare to the Source file date/time value displayed from the DSPOBJD command.

- The Display Object Description (DSPOBJD) command using DETAIL(*SERVICE). This display shows the date/time of the source member used to create the object.

*Note: If you are using the data written to output files to determine if the source and object dates are the same, then you can compare the ODSRCD (source date) and ODSRCT (source time) fields from the output file of the DSPOBJD DETAIL(*SERVICE) command to the MBUPDD (member update date) and MBUPDT (member update time) fields from the output file of the DSPFD TYPE(*MBR) command.*

PHYSICAL FILES

STRUCTURE FUNDAMENTALS

If there is any one AS/400 concept that is the key to unlocking a basic understanding of application development , it is the concept of AS/400 file structure. It's not that the concept is difficult to grasp it's just that there are few facts to grasp.

On AS/400 all files are described at four levels.

- object level description
- file level description
- record level description
- field level description

1. Object level description : A file in AS/400 object whose object type is *file. The AS/400 maintains the same object description information for a file as it does for any other object on the system.
2. File level description : The second level of description the system maintains for file objects is a file level description. The file description is created along with the file when you execute a command CRTxxxF. It describes the attributes or characters of a particular file and is embedded within the file itself .
3. Record level description : The third level of descriptive information the system maintains for files is the record level description. This level describes various, if more than one, record formats that exist in the file. A record format describes a set of fields that make a record. All files have at least one record format.
4. Field level description : The final level of descriptive information the system maintains for files is the field level description. Field descriptions do not exist for all types of files; tape files diskette files, DDM files, and save files have no description because they have no fields. For the remaining files- physical, logical, source, display, printer and ICF- a description of each field and field attributes is maintained. You can use the DSPFFD command to display or print the field level description for a file.

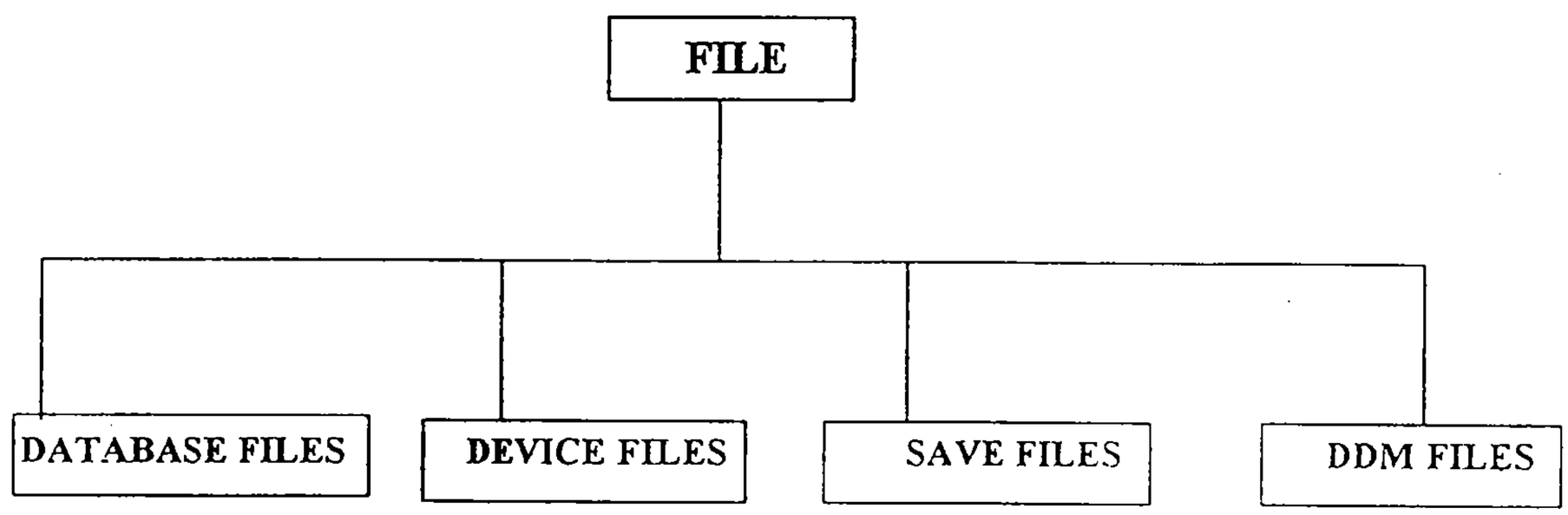
DATA MEMBERS

We now need consider a particular organization element that applies only to database and source files, the two type of files that actually contain records of data.

AS/400 database and source files can have no data members. If you create a physical file and specify MBR(*NONE), the file will be created without any data member for records. If you try to add records to that file, the system will issue an error stating that no data members exist. you would have to use the ADDPFM command to create a data member in the file before you could add records to the file.

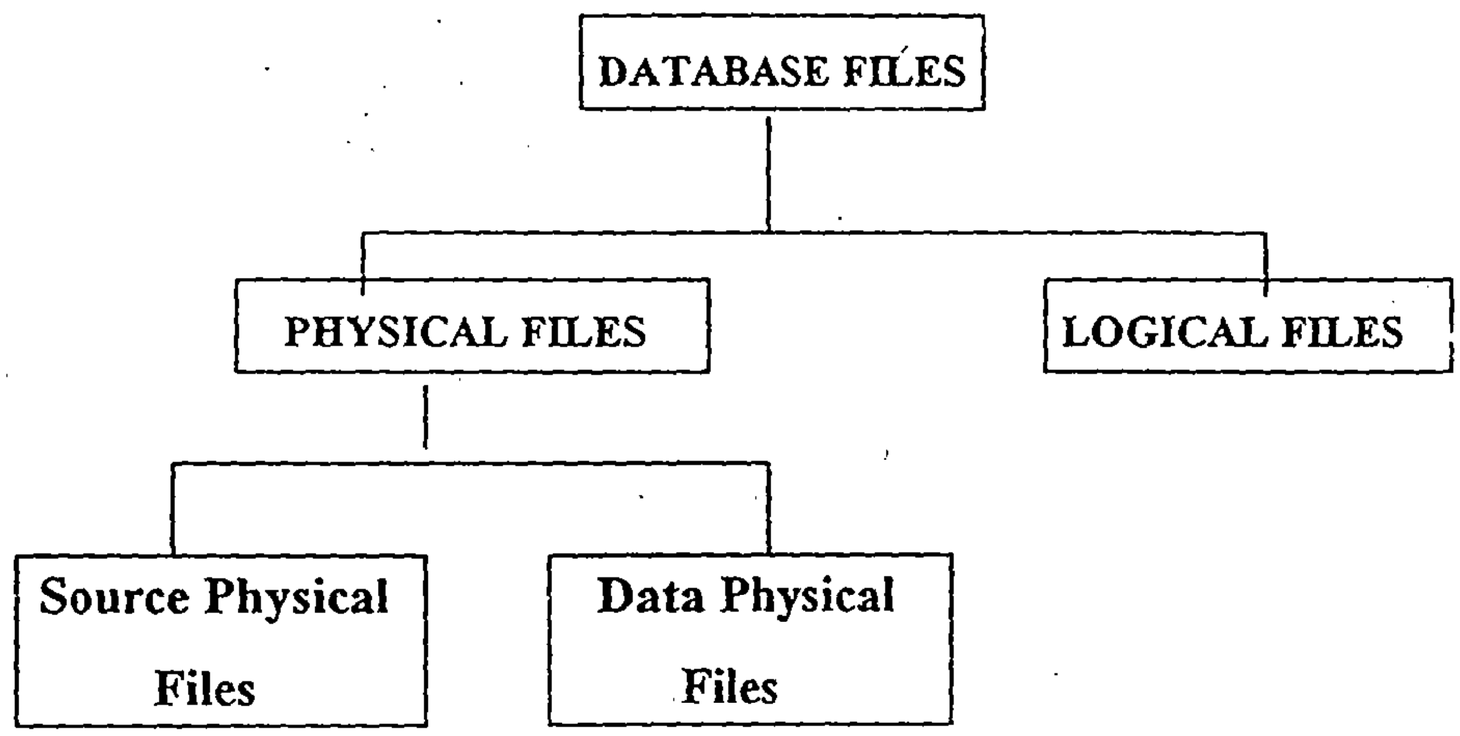
At the other end of the scale is the fact that you can have multiple data members in a file. A source file offers a good example. Each source member is a different member in the file. When you create a new source member, you are actually creating another data member in this physical source file.

TYPES OF FILES



NOTE: DDM means Distributed Data Management

DATABASE FILES



Database files are AS/400 objects that actually contain data or provide access to data. Two types of files are considered database files-physical files and logical files. A physical file denoted as TYPE(*FILE) and ATTR(PF), has file, record and filed-level descriptions and can be created with or without using externally described sourse specifications, physical files so called because they contain actual data. Can have only one record format. The data entered into the physical file is assigned a relative record based on arrival sequance. You can view the data that exists in a specific data member of a file using the DSPPFM (Display physical file member) command.

A logical file denoted as TYPE(*FILE) is created in conjuction with physical files to determine how the data will be presented to the requestor. Logical files contain no data but instead are used to specify key fields, select/omit logic field selection, or field manipulation. The key fields serve to specify the access path to use for accessing the actual data records that reside in physical files. Logical files must be externally described using DDS and can be used only in conjuction with externally described physical file.

SOURSE FILES

A sourse file like QRPGSRC where RPG sourse members are maintained, is simply a customized form of a physical file and as such, sourse files are denoted as TYPE(*FILE) and ATTR(PF). All sourse files created using the CRTSRCPF command have the same record format and thus the same fields. When you use the CRTSRCPF command, the system creates a physical file that allows multiple data members. Each program source is one physical file member. When you edit a particular sourse member you are simply editing a specific data member in a file.

DEVICE FILES

Device files contain no actual data. They are files whose descriptions provide introduction about how an application is to use particular devices. The device file contain information valid for the device type the application is accessing. The types of device files are display, printer, tape, diskette, and ICE.

Display files, denoted by the system as TYPE(*FILE) and ATTR(DSPF) provide specific information relating to how an application can interact with a workstation. While a display file contains no data, the display file contain various records formats that represents the screens the application will present to the workstation.

Printer files, denoted by the system as TYPE (*TYPE) and ATTR(PRTF) provide specific information relating to how an application can spool data for output to a writer. The print file can be created with a maximum record specified and one format to be used with a HLL program and program-described printing or the print file can be created from external sourse statements that define the formats to be used for printing.

Tape files denoted by the system as TYPE (*TYPE) and ATTR(PRTF) provide specific information relating to how an application can read or write data using tape media. The description of the tape file contains information such as the device name for tape read/write operations, the specific tape volume will requested , the density of the tape file to be processed, the record and block length to be used.

Diskette files denoted by the system as TYPE (*FILE) and ATTR(DKTF), are identical to tape filesexcept that these files support diskette devices. Diskette files have attributes that describe volume to be used and record and block length.

DDM Files

DDM (Distributed Data Management) files denoted by the as TYPE(*FILE) and ATTR(DDMF), are objects that represent files that exist an a remote system. For instance , if your customer file exists on a re mote system , you can create a DDM file on the local system that specifically point to that customer file on the remote system. DDM file provide you with an interface that you can compile programs using the file, read records, write records, and update records, while the system handles the communications.

SAVE FILES

Save files denoted by the as TYPE(*FILE) and ATTR(SAVE), are a special form of file designied specifically to handle save\restore data. You cannot determine the file ,record, and field level descriptions for a save file. The system creates a specific descriptions used for all save files to make them compatible with save\restore operations.

DESCRIBING DATABASE FILES

Records In Database Files Can Be Described In Two Ways.

1. Field Level Description.
2. Record Level Description

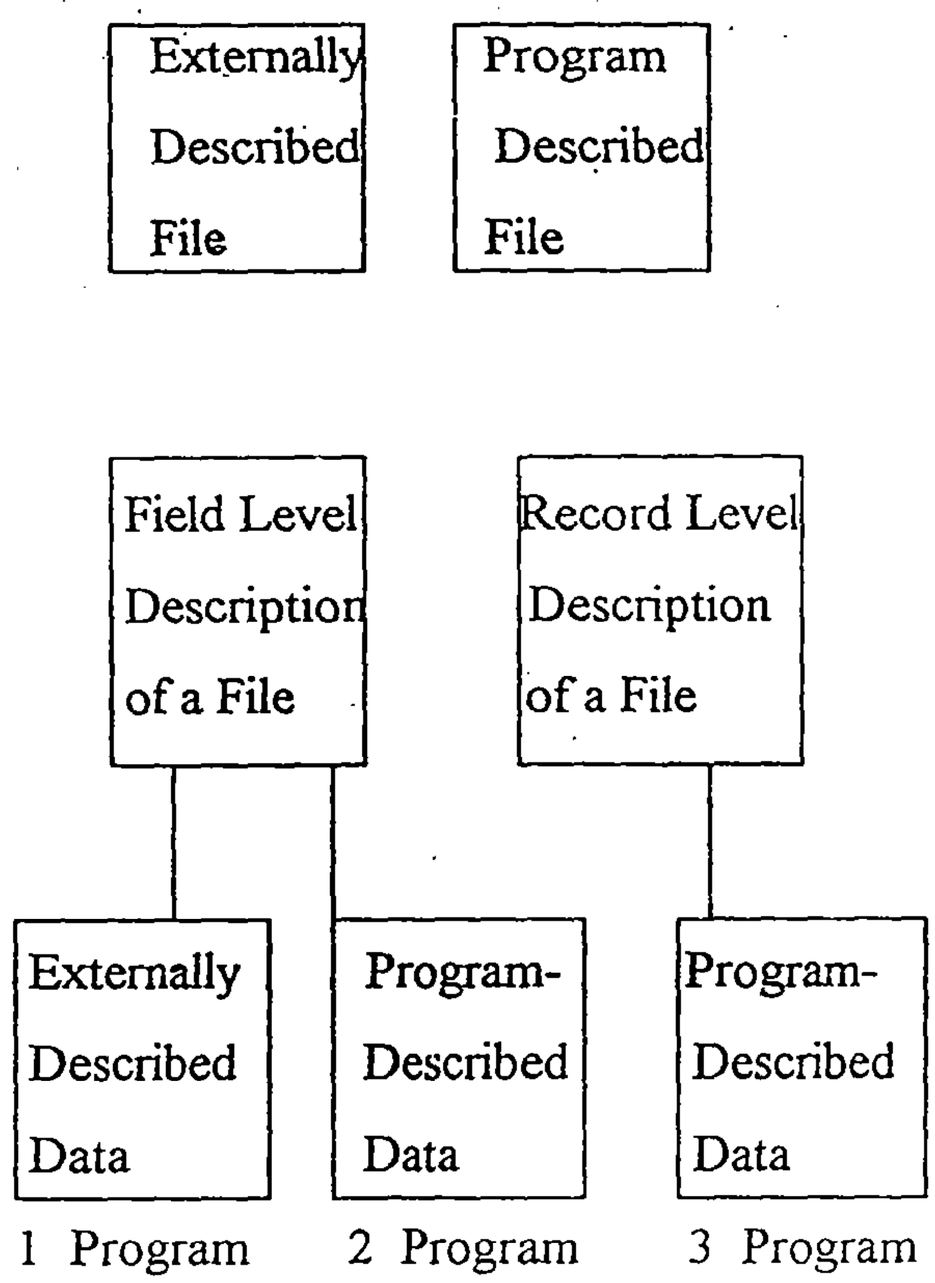
- Field Level Description : The fields in the record are described to the system . Some of the things you can describe for each field include NAME, LENGTH, DATA TYPES, VALIDITY CHECKS, AND TEXT DESCRIPTION. Database files that are created with field level description are referred to as externally described files.

- Record Level description : Only the length of the record in the file is described to the system. The system doesnot no about fields in the file. These database files are referred to as program described files.

Programs Can Use File Description In Two Ways.

- The program uses the field level descriptions that are part of the file. Because the field descriptions are external to the program itself, the term externally described data, is used.
- The program uses the fields that are described in the program itself; therefore, the data is called program described data. Fields in files that are only described to the record level must be described in the program using the file.

Note: *Programs can use either externally described or program descibed files . If you choose to describe a file to field level the system can do more for you, when you compile your program, the system can extract information from an externally described file and automatically include field information in your programs. Therefore you need not code field information in each program that uses the file.*



1. The program uses the field level description of a file that is defined to the system. At compilation time, the language compiler copies the external description of the file into the program.
2. The program uses a file that is described to the field level to the system, but it doesnot use the actual field descriptions. At compilation time, the language compiler doesnot copy the external description of the file into the program. the field attributes used in the program must be the same as the field attributes in the external description.

3. The program uses the file that is used only to the record level to the system. The fields in thefile must be described in the program. externally described files can also be described in a program.

Dictionary Described Data

A program-described file can be dictionary-described. You can describe the record format information using interactive data definition utility(IDDU). Even though the file is program-described, AS/400 Query, client access , and data file utility(DFU) will use the record format description stored in the data dictionary.

An externally described file can also be dictionary described \. You can use IDDU to describe a file using IDDU. The file created is an externally described file. You can also move into the data dictionary the file description stored in an eternally described file. The system always ensures that the definitions in the data dictionary and the description stored in the externally described file are identical.

Methods Of Describig Data to the System

If you want to describe a file just to the record level, you can use the record length(RCDLEN) parameter on the create physical file (CRTPF) and create source physical file (CRTSRCPF) commands.

If you want to describe your file to the feld level, several methods can be used to describe data to the database system: IDDU, SQL* commands or data description specifications (DDS).

Describing A Database File To The System

When you describe a database file to the system you describe the two major parts of that file: The record format and the access path.

Describing the record format :

The record format describes the order of the fields in each record. The record format also describes each field in detail including: length, datatype, validity checks, text desription, and other information.

A physical file can have only one record format. The record format in a physical file describes the way the data is actually stored.

A logical file contains no data. Logical files are used to arrange data from one or more physical files into different formats and sequences.

Alogical file record format can change the length and data type of fields stoed in physical files. The system does the necessary conversion between the physical field description and the logical file field description.

Describing Acess Path :

An acess path describes the order in which records are to be retrieved. When you describe an acesspath, you describe whether it will be keyed sequence or arrival sequence acess path.

Naming Conventions

The file name, record format name, and field name can be aslong as 10 characters and must follow all system naming convention, but you should keep in mind that some high level languages have more restsrictive naming conventions than the system does.

In addition, names must be unique as follows:

- Field names must be unique in a record format.
- Record format names and member names must be unique in a file.
- File names must be unique in a library.

Describing Database Files Using DDS

When you describe a database file using DDS, you can describe information at the file level, record format, join, field, key, and select/omit levels:

- File level DDS give the system information about the entire file.
- Record format levels DDS give the system information about a specific record format in the file. For example, when you describe a logical file record format, you can specify the physical file that it is based on.
- Join level DDS give the system information about the physical files used in a join logical file.
- Field level DDS give the system level information about individual fields in the record format.
- Key field level DDS give the system information about the key fields for the file.
- Select/Omit field level DDS give the system information about which records are to be returned to the program when processing the file. Select/Omit specification apply to logical files only.

PHYSICAL FILE KEY WORDS

Keywords used in physical files have four distinct levels. They are:

- File level keywords.
- Record format level keywords.
- Field level keywords.
- Key-field keywords.

There are 35 keywords associated with physical files. These keywords are used to provide further qualification to the file, record format, field or key field. Some of more common keywords are:

File level keywords.

FCFO *First changed first out :*

when duplicate keys exist in a file this keyword is used to specify that the record that was changed earliest is to be retrieved first.

FIFO *first in first out :*

when duplicate keys exist in a file this keyword is used to specify that the record that was written earliest is to be retrieved first.

LIFO *Last in first out :*

When duplicate keys exist in afile this keyword is used to specify that the record that was written recently was to retrieve first.

REF *Reference :*

Used to indicate the name of another file from which field references whenever indicated are to be retrieved.

UNIQUE *Unique key :*

Used to indicate the name of another file from which field references whenever indicated are to be retrieved.

Record-format level keywords

TEXT *Text :*

Used to assign descriptive text for the record format.

FORMAT

This keyword references another database file on the AS/400 from which all the fields will be shared.

Field Level Keywords.

ALIAS *Alias name :*

Used to indicate alternate name for the named field.

CHECK

This keyword does not affect the physical file itself. When a display file references this physical file field, the check keyword will be copied to provide default validity checking.

COMP

This keyword does not affect the physical file itself. When a display file references this physical file field, the COMP keyword will be copied to provide default validity checking.

COLHDG *Column heading :*

Used to provide descriptive text about the named field. Many IBM Utilities like Query/400, DFU select these headings

DATFMT *Date Format :*

Used to define the date format inwhich date is to be stored in the named field.

DFT *Default :*

This keyword lets you retrieve the field reference from another field in a file other than that specified in the REF keyword

Key field level keywords

ABSVAL *Absolute value :*

This keyword tells the AS/400 to ignore the sign of a numeric field when creating an index for the physical file.

DESCEND *Descending keyword :*

This keyword tells the AS/400 to sort the field in descending sequence when creating an index for the physical file.

UNSIGNED *Unsigned keyword :*

This keyword tells the AS/400 to ignore the sign of a numeric field when creating an index for the physical file.

SIGNED *Signed keyword :*

This keyword tells the AS/400 to consider the sign of a numeric field when creating an index for the physical file.

Example Of Describing A Physical File Using Dds.

1. File level entries (optional). The UNIQUE keyword is used to indicate that the value of the key field in each record in the file must be unique. Duplicate key values are not allowed in this file.

2. Record format level entries. The record format name is specified, along with an optional text description.

3. Field level entries. The field names and field lengths are specified, along with an optional text description for each field.

4. Key field level entries (optional). The field names used as key fields are specified.

5. Comment (optional).

```
A* ORDER HEADER FILE (ORDHDRP)
A      5
A              1                          UNIQUE
A      2       R ORDHDR                   TEXT('Order header record')
A      3         CUST            5S 0     TEXT('Customer number')
A                ORDER           5S 0     TEXT('Order number')
A      .
A      .
A              K CUST
A      4       K ORDER
```

Figure 1. DDS for a Physical File (ORDHDRP)

The following example shows a physical file ORDHDRP (an order header file), which has an arrival sequence access path without key fields specified, and the DDS necessary to describe that file.

Purchase

Customer Number (CUST)	Order Number (ORDER)	Order Date (ORDATE)	Order Number (CUSORD)	Shipping Instructions (SHPVIA)	Order Status (ORDSTS)	State (STATE)
Packed Decimal Length 5 0 Dec Pos	Packed Decimal Length 5 0 Dec Pos	Packed Decimal Length 6 0 Dec Pos	Packed Decimal Length 15 0 Dec Pos	Character Length 15	Character Length 1	Character Length 2

```
A* ORDER HEADER FILE (ORDHDRP)
A       R ORDHDR          TEXT('Order header record')
A         CUST      5P 0  TEXT('Customer Number')
A         ORDER     5P 0  TEXT('Order Number')
A         ORDATE    6P 0  TEXT('Order Date')
A         CUSORD    15P 0 TEXT('Customer Order No.')
A         SHPVIA    15A   TEXT('Shipping Instr')
A         ORDSTS    1A    TEXT('Order Status')
A         OPRNME    10A   TEXT('Operator Name')
A         ORDAMT    9S 2  TEXT('Order Amount')
A         CUTYPE    1A    TEXT('Customer Type')
A         INVNBR    5S 0  TEXT('Invoice Number')
A         PRTDAT    6L    TEXT('Printed Date')
A         SEQNBR    5S 0  TEXT('Sequence Number')
A         OPNSTS    1A    TEXT('Open Status')
A         LINES     3S 0  TEXT('Order Lines')
```

```
A        ACTMTH  2S 0   TEXT('Accounting Month')
A        ACTYR   2S 0   TEXT('Accounting Year')
A        STATE   2A     TEXT('State')
```

The R in position 17 indicates that a record format is being defined. The record format name ORDHDR is specified in positions 19 through 28.

You make no entry in positions 17 when your describing a field; a blank in position 17 along with a name in positions 19 through 28 indicates a field name.

The data type is defined in position 35. The valid data types are :

Entry	Meaning
A	Character
P	Packed decimal
S	Zoned decimal
B	Binary
F	Floating point
H	Hexadecimal
L	Date
T	Time
Z	Timestamp

Example Of Using A Logical File Using Dds

The DDS for a logical file must be in the following order (Figure 2):

1. File level entries (optional). In this example, the UNIQUE key word indicates that for this file the key value for each record must be unique; no duplicate key values are allowe.

2. Record format level entries. In this example, the record format name, the associated physical file, and an optional text description are specified.

3. Field level entries (optional). In this example, each field name used in the record format is specified.
4. Key field level entries (optional). In this example, the Order field is used as a key field.

5. Select/omit field level entries (optional). In this example, all records whose Opnsts field contains a value of N are omitted from the file's access path. That is, programs reading records from this file will never see a record whose Opnsts field contains an N value.

6. Comment.

```
A* ORDER HEADER FILE (ORDHDRP)
A      6
A      1                          UNIQUE
A      2      R ORDHDR            PFILE(ORDHDRP)
A      3        ORDER             TEXT('Order number')
A               CUST              TEXT('Customer number')
A               .
A               .
A               .
A      4        K ORDER
A               O OPNSTS    5     CMP(EQ 'N')
A               S                 ALL
```

Figure 2. DDS for a Simple Logical File (ORDHDRL)

A logical file must be created after all physical files on which it is based are created. The PFILE keyword in the previous example, is used to specify the physical file or files on which the logical file is based.

Record formats in a logical file can be:

- A new record format based on fields from a physical file
- The same record format as in a previously described physical or logical file.

Fields in the logical file record format must either appear in the record format of at least one of the physical files or be derived from the fields of the physical files on which the logical file is based.

Field Reference File

If a field was already described in an existing file, and you want to use that field description in a new file you are setting up, you can request the system to copy that description into your new file description. The DDS keywords REF and REFFLD allow you to refer to a field description in an existing file. This helps reduce the effort of coding DDS statements. It also helps ensure that the field attributes are used Consistently in all files that use the field.

In addition, you can create a physical file for the sole purpose of using its field descriptions. That is, the file does not contain data; it is used only as a reference for the field descriptions for other files.

This type of file is known as a field reference file. A field reference file is a physical file containing no data, just field descriptions.

You can use a field reference file to simplify record format descriptions and to ensure field descriptions are used consistently. You can define all the fields you need for an application or any group of files in a field reference file. You can create a field reference file using DDS and the Create Physical File (CRTPF) command.

After the field reference file is created, you can build physical file record formats from this file without describing the characteristics of each field in each file. When you build physical files, all you need to do is refer to the field reference file (using the REF and REFFLD keywords) and specify any changes. Any changes to the field descriptions and keywords specified in your new file override the descriptions in the field reference file.

In the following example, a field reference file named DSTREFP is created for distribution applications. Figure 3 shows the DDS needed to describe DSTREFP.

```
A* FIELD REFERENCE FILE (DSTREFP)
A       R DSTREF            TEXT('Field reference file')
A*
A* FIELDS DEFINED BY CUSTOMER MASTER RECORD  (CUSMST)
A         CUST       5S 0   TEXT('Customer numbers')
A                           COLHDG('CUSTOMER' 'NUMBER')
A         NAME       20A    TEXT('Customer name')
A         ADDR       20A    TEXT('Customer address')
A         CITY       20A    TEXT('Customer city')
A         STATE      2A     TEXT('State abbreviation')
A                           CHECK(MF)
A         CRECHK     1A     TEXT('Credit check')
A                           VALUES('Y' 'N')
A         SEARCH     6A 0   TEXT('Customer name search')
A                           COLHDG('SEARCH CODE')
A         ZIP        5S 0   TEXT('Zip code')
A                           CHECK(MF)
A         CUTYPE     15A    COLHDG('CUSTOMER' 'TYPE')
A                           RANGE(1 5)
A*
A* FIELDS DEFINED BY ITEM MASTER RECORD (ITMAST)
A         ITEM       5S     TEXT('Item number')
A                           COLHDG('ITEM' 'NUMBER')
A                           CHECK(M10)
A         DESCRP     18A    TEXT('Item description')
A         PRICE      5S 2   TEXT('Price per unit')
A                           EDTCDE(J)
A                           CMP(GT 0)
A                           COLHDG('PRICE')
```

```
A         ONHAND  5S 0   TEXT('On hand quantity')
A                        EDTCDE(Z)
A                        CMP(GE 0)
A                        COLHDG('ON HAND')
A         WHSLOC  3A     TEXT('Warehouse location')
A                        CHECK(MF)
A                        COLHDG('BIN NO')
A         ALLOC   R      REFFLD(ONHAND *SRC)
A                        TEXT('Allocated quantity')
A                        CMP(GE 0)
A                        COLHDG('ALLOCATED')
A* FIELDS DEFINED BY ORDER HEADER RECORD (ORDHDR)
A         ORDER   5A 0   TEXT('Order number')
A                        COLHDG('ORDER' 'NUMBER')
A         ORDATE  6L 0   TEXT('Order date')
A                        EDTCDE(Y)
A                        COLHDG('DATE' 'ORDERED')
A         CUSORD  15S    TEXT('Cust purchase ord no.')
A                        COLHDG('P.O.' 'NUMBER')
A         SHPVIA  15A    TEXT('Shipping instructions')
A         ORDSTS  1A     TEXT('Order status code')
A                        COLHDG('ORDER' 'STATUS')
A         OPRNME  R      REFFLD(NAME *SRC)
A                        TEXT('Operator name')
A                        COLHDG('OPERATOR NAME')
A         ORDAMT  9S 2   TEXT('Total order value')
A                        COLHDG('ORDER' 'AMOUNT')
A*
A         INVNBR  5S 0   TEXT('Invoice number')
```

```
A                          COLHDG('INVOICE' 'NUMBER')
A         PRTDAT   6L 0    EDTCDE(Y)
A                          COLHDG('PRINTED' 'DATE')
A         SEQNBR   5S 0    TEXT('Sequence number')
A                          COLHDG('SEQ' 'NUMBER')
A         OPNSTS   1A      TEXT('Open status')
A                          COLHDG('OPEN' 'STATUS')
A         LINES    3A 0    TEXT('Lines on invoice')
A                          COLHDG('TOTAL' 'LINES')
A         ACTMTH   2S 0    TEXT('Accounting month')
A                          COLHDG('ACCT' 'MONTH')
A         ACTYR    2A 0    TEXT('Accounting year')
A                          COLHDG('ACCT' 'YEAR')
A*
A* FIELDS DEFINED BY ORDER DETAIL/LINE ITEM RECORD
A*(ORDDTL)
A         LINE     3S 0    TEXT('Line no. this item')
A                          COLHDG('LINE' 'NO')
A         QTYORD   3S 0    TEXT('Quantity ordered')
A                          COLHDG('QTY' 'ORDERED'
A                          CMP(GE 0)
A         EXTENS   6S 2    TEXT('Ext of QTYORD x PRICE')
A                          EDTCDE(J)
A                          COLHDG('EXTENSION')
A*
A* FIELDS DEFINED BY ACCOUNTS RECEIVABLE
A         ARBAL    8S 2    TEXT('A/R balance due')
A                          EDTCDE(J)
A* WORK AREAS AND OTHER FIELDS THAT OCCUR IN MULTIPLE
```

```
A* PROGRAMS
A          STATUS   12A    TEXT('status description')
```

Figure 3. DDS for a Field Reference File (DSTREFP)

Assume that the DDS in Figure 1-3 is entered into a source file FRSOURCE;the member name is DSTREFP. To then create a field reference file, use the Create Physical File (CRTPF) command as follows:

```
CRTPF FILE(DSTPRODLB/DSTREFP)
SRCFILE(QGPL/FRSOURCE) MBR(*NONE)
TEXT('Distribution field reference file')
```

The parameter MBR(*NONE) tells the system not to add a member to the file (because the field reference file never contains data and therefore does not need a member).

To describe the physical file ORDHDRP by referring to DSTREFP, use the following DDS (Figure 4):

```
A* ORDER HEADER FILE (ORDHDRP) - PHYSICAL FILE RECORD
A* DEFINITION
A                          REF(DSTREFP)
A       R ORDHDR           TEXT('Order header record')
A         CUST       R
A         ORDER      R
A         ORDATE     R
A         CUSORD     R
A         SHPVIA     R
A         ORDSTS     R
```

```
A       OPRNME R
A       ORDAMT R
A       CUTYPE  R
A       INVNBR  R
A       PRTDAT  R
A       SEQNBR  R
A       OPNSTS  R
A       LINES   R
A       ACTMTH R
A       ACTYR   R
A       STATE   R
```

Figure 4. DDS for a Physical File (ORDHDRP) Built from a Field Reference File

The REF keyword (positions 45 through 80) with DSTREFP (the field reference file name) specified indicates the file from which field descriptions are to be used. The R in position 29 of each field indicates that the field description is to be taken from the reference file.

When you create the ORDHDRP file, the system uses the DSTREFP file to determine the attributes of the fields included in the ORDHDR record format. To create the ORDHDRP file, use the Create Physical File (CRTPF) command. Assume that the DDS in Figure 4 was entered into a source file QDDSSRC; the member name is ORDHDRP.

A record format can be described once in either a physical or a logical file (except a join logical file) and can be used by many files. When you describe a new file, you can specify that the record format of an existing file is to be used by the new file. This can help reduce the number of DDS statements that you would normally code to describe a record format in a new file and can save auxiliary storage space.

The file originally describing the record format can be deleted without affecting the files sharing the record format. After the last file using the record format is deleted, the system automatically deletes the record format description.

The following shows the DDS for two files. The first file describes a record format, and the second shares the record format of the first:

```
A       R RECORD1        PFILE(CUSMSTP)
A         CUST
A         NAME
A         ADDR
A         SEARCH
A       K CUST
```

Figure 5. DDS for a Logical File (CUSMSTL)

```
A       R RECORD1        PFILE(CUSMSTP)
A                        FORMAT(CUSMSTL)
A       K NAME
```

Figure 6. DDS for a Logical File (CUSTMSTL1) Sharing a Record Format

The example shown in Figure 5 shows file CUSMSTL, in which the fields Cust, Name, Addr, and Search make up the record format. The Cust field is specified as a key field.

The DDS in Figure 6 shows file CUSTMSTL1, in which the FORMAT keyword names CUSMSTL to supply the record format. The record format name must be RECORD1, the same as the record format name shown in Figure 5. Because the files are sharing the same format, both files have fields Cust, Name, Addr, and Search in the record format. In file CUSMSTL1, a different key field, Name is specified.

The following restrictions apply to shared record formats:

- A physical file cannot share the format of a logical file.
- A join logical file cannot share the format of another file, and another file cannot share the format of a join logical file.
- A view cannot share the format of another file, and another file cannot share the format of a view. (In SQL, a view is an alternative representation of data from one or more tables. A view can include all or some of the columns contained in the table or tables on which it is defined.)

If the original record format is changed by deleting all related files and creating the original file and all the related files again, it is changed for all files that share it. If only the file with the original format is deleted and re-created with a new record format, all files previously sharing that file's format continue to use the original format.

If a logical file is defined but no field descriptions are specified and the FORMAT keyword is not specified, the record format of the first physical file (specified first on the PFILE keyword for the logical file) is automatically shared. The record format name specified in the logical file must be the same as the record format name specified in the physical file.

To find out if a file shares a format with another file, use the RCDFMT parameter on the Display Database Relations (DSPDBR) command.

Record Format Sharing Limitation: A record format can only be shared by **32K** objects. Error messages are issued when you reach the limitation.
You may encounter this limitation in a circumstance where you are duplicating the same database object multiple times.

ACCESS PATH FOR THE FILE

An access path describes the order in which records are to be retrieved. Records in a physical or logical file can be retrieved using an arrival sequence access path or a keyed sequence access path. For logical files, you can also select and omit records based on the value of one or more fields in each record.

ARRIVAL SEQUENCE ACCESS PATH

The arrival sequence access path is based on the order in which the records arrive and are stored in the file. For reading or updating, records can be accessed:

- Sequentially, where each record is taken from the next sequential physical position in the file.
- Directly by relative record number, where the record is identified by its position from the start of the file.

An externally described file has an arrival sequence access path when no key fields are specified for the file.

An arrival sequence access path is valid only for the following:

- Physical files
- Logical files in which each member of the logical file is based on only one physical file member
- Join logical files
- Views

KEYED SEQUENCE ACCESS PATH

A keyed sequence access path is based on the contents of the key fields as defined in DDS. This type of access path is updated whenever records are added or deleted, or

when records are updated and the contents of a key field is changed. The keyed sequence access path is valid for both physical and logical files. The sequence of the records in the file is defined in DDS when the file is created and is maintained automatically by the system.

Key fields defined as character fields are arranged based on the sequence defined for EBCDIC characters. Key fields defined as numeric fields are arranged based on their algebraic values, unless the UNSIGNED (unsigned value) or ABSVAL (absolute value) DDS keywords are specified for the field. Key fields defined as DBCS are allowed, but are arranged only as single bytes based on their bit representation.

EXAMPLE OF DDS FOR PHYSICAL FILE WITH ABSVAL

```
A       R RECORD1
A         CUST
A         NAME
A         ADDR
A         SEARCH
A       K CUST            ABSVAL
```

Arranging Key Fields Using an Alternative Collating Sequence: Keyed fields that are defined as Character fields can be arranged based either on the sequence for EBCDIC characters or on an alternative collating sequence. Consider the following records:

DDS For Employee Physical File (EMPDET)

```
A       R RECORD
A         EMPNAME       10A
A         DEPTNBR       2S
A         EMPNBR        5S
```

Record	Empname	Deptnbr	Empnbr
1	Jones, Mary	45	23318
2	Smith, Ron	45	41321
3	JOHNSON, JOHN	53	41322
4	Smith, ROBERT	27	56218
5	JONES, MARTIN	53	62213

If the Empname is the key field and is a character field, using the sequence for **EBCDIC** characters, the records would be arranged as follows:

```
A      R RECORD          PFILE(EMPDET)
A      K EMPNAME
```

Record	Empname	Deptnbr	Empnbr
1	Jones, Mary	45	23318
3	JOHNSON, JOHN	53	41322
5	JONES, MARTIN	53	62213
2	Smith, Ron	45	41321
4	Smith, ROBERT	27	56218

Notice that the **EBCDIC** sequence causes an unexpected sort order because the lowercase characters are sorted before uppercase characters. Thus, Smith, Ron sorts before Smith, ROBERT. An alternative collating sequence could be used to sort the records when the records were entered using uppercase and lowercase as shown in the following example:

Record	Empname	Deptnbr	Empnbr
3	JOHNSON, JOHN	53	41322
5	JONES, MARTIN	53	62213
1	Jones, Mary	45	23318
4	Smith, ROBERT	27	56218
2	Smith, Ron	45	41321

To use an alternative collating sequence for a character key field, specify the **ALTSEQ DDS** keyword, and specify the name of the table containing the alternative collating sequence. When setting up a table, each 2-byte position in the table corresponds to a character. To change the order in which a character is sorted, change its 2-digit value to the same value as the character it should be sorted equal to.

EXAMPLE USING **ALTSEQ** KEYWORD

```
A                                   ALTSEQ(TABLELIB/TABLE1)
A       R RECORD
A         EMPNAME                   10A
A         DEPTNBR          2S
A         EMPNBR           5S
A       K.EMPNAME
```

NOTE:

1. *ALTSEQ keyword names a table (which was created with the crttbl command) to be used as an 'Alternate collating sequence' for sequencing the keys in the index.*
2. *ALTSEQ keyword is only valid for files with keyed access path.*
3. *The access path is changed to the table Table1.*

Arranging Key Fields Using the SRTSEQ Parameter: You can arrange key fields containing character data according to several sorting sequences available with the SRTSEQ parameter. Consider the following records:

Record	Empname	Deptnbr	Empnbr
1	Jones, Marilyn	5	3318
2	Smith, Ron	5	1321
3	JOHNSON, JOHN	53	41322
4	Smith, ROBERT	27	56218
5	JONES, MARTIN	53	62213
6	Jones, Martin	8	9231

If the Empname field is the key field and is a character field, the *HEX sequence (the EBCDIC sequence) arranges the records as follows:

Record	Empname	Deptnbr	Empnbr
1	Jones, Marilyn	45	23318
6	Jones, Martin	08	29231
3	JOHNSON, JOHN	53	41322
5	JONES, MARTIN	53	62213
2	Smith, Ron	45	41321
4	Smith, ROBERT	27	56218

Notice that with the ***HEX** sequence, all lowercase characters are sorted before the uppercase characters. Thus, Smith, Ron sorts before Smith, ROBERT, and JOHNSON, JOHN sorts between the lowercase and uppercase Jones. You can use the ***LANGIDSHR** sort sequence to sort records when the records were entered using a

mixture of uppercase and lowercase. The ***LANGIDSHR** sequence, which uses the same collating weight for lowercase and uppercase characters, results in the following:

Record	Empname	Deptnbr	Empnbr
3	JOHNSON, JOHN	53	41322
1	Jones, Marilyn	45	23318
5	JONES, MARTIN	53	62213
6	Jones, Martin	08	29231
4	Smith, ROBERT	27	56218
2	Smith, Ron	45	41321

Notice that with the ***LANGIDSHR** sequence, the lowercase and uppercase characters are treated as equal. Thus, JONES, MARTIN and Jones, Martin are equal and sort in the same sequence they have in the base file. While this is not incorrect, it would look better in a report if all the lowercase Jones preceded the uppercase JONES. You can use the ***LANGIDUNQ** sort sequence to sort the records when the records were entered using an inconsistent uppercase and lowercase. The ***LANGIDUNQ** sequence, which uses different but sequential collating weights for lowercase and uppercase characters, results in the following:

Record	Empname	Deptnbr	Empnbr
3	JOHNSON, JOHN	53	41322
1	Jones, Marilyn	45	23318
6	Jones, Martin	08	29231
5	JONES, MARTIN	53	62213
4	Smith, ROBERT	27	56218
2	Smith, Ron	45	41321

The *LANGIDSHR and *LANGIDUNQ sort sequences exist for every language supported in your system. The LANGID parameter determines which *LANGIDSHR or *LANGIDUNQ sort sequence to use. Use the SRTSEQ parameter to specify the sort sequence and the LANGID parameter to specify the language.

Arranging Key Fields in Ascending or Descending Sequence: Key fields can be arranged in either ascending or descending sequence. Consider the following records:

Record	Empnbr	Clsnbr	Clsnam	Cpdate
1	56218	412	Welding I	032188
2	41322	412	Welding I	011388
3	64002	412	Welding I	011388
4	23318	412	Welding I	032188
5	41321	412	Welding I	051888
6	62213	412	Welding I	032188

If the Empnbr field is the key field, the two possibilities for organizing these records are:

- In ascending sequence, where the order of the records in the access path is:

Record	Empnbr	Clsnbr	Clsnam	Cpdate
4	23318	412	Welding I	032188
5	41321	412	Welding I	051888
2	41322	412	Welding I	011388
1	56218	412	Welding I	032188
6	62213	412	Welding I	032188

- In descending sequence, where the order of the records in the access path is:

EXAMPLE USING DESCEND KEYWORD

```
A      R RECORD          PFILE(EMPPF)
A      K EMPNAME         DESCEND
```

Record	Empnbr	Clsnbr	Clsnam	Cpdate
3	64002	412	Welding I	011388
6	62213	412	Welding I	032188
1	56218	412	Welding I	032188
2	41322	412	Welding I	011388
5	41321	412	Welding I	051888
4	23318	412	Welding I	032188

When you describe a key field, the default is ascending sequence. However, you can use the DESCEND DDS keyword to specify that you want to arrange a key field in descending sequence.

Using More Than One Key Field: You can use more than one key field to arrange the records in a file. The key fields do not have to use the same sequence. For example, when you use two key fields, one field can use ascending sequence while the other can use descending sequence. Consider the following records:

Record	Order	Ordate	Line	Item	Qtyord	Extens
1	52218	063088	01	88682	425	031875
2	41834	062888	03	42111	30	020550
3	41834	062888	02	61132	4	021700
4	52218	063088	02	40001	62	021700
5	41834	062888	01	00623	50	025000

If the access path uses the Order field, then the Line field as the key fields, both in ascending sequence, the order of the records in the access path is:

Record	Order	Ordate	Line	Item	Qtyord	Extens
5	41834	062888	01	00623	50	025000
3	41834	062888	02	61132	4	021700
2	41834	062888	03	42111	30	020550
1	52218	063088	01	88682	425	031875
4	52218	063088	02	40001	62	021700

If the access path uses the key field Order in ascending sequence, then the Line field in descending sequence, the order of the records in the access path is:

EXAMPLE USING DESCEND KEYWORD

```
A     R RECORD1          PFILE(ORDPF)
A     K ORDER
A     K.LINE             DESCEND
```

Record	Order	Ordate	Line	Item	Qtyord	Extens
2	41834	062888	03	42111	30	020550
3	41834	062888	02	61132	4	021700
5	41834	062888	01	00623	50	025000
4	52218	063088	02	40001	62	021700
1	52218	063088	01	88682	425	031875

when a record has key fields whose contents are the same as the key field in another record in the same file, then the file is said to have records with duplicate key values.

However, the duplication must occur for all key fields for a record if they are to be called duplicate key values.

For example, if a record format has two key fields Order and Ordate, duplicate key values occur when the contents of both the Order and Ordate fields are the same in two or more records. These records have duplicate key values:

Order	Ordate	Line	Item	Qtyord	Extens
41834	062888	03	42111	30	020550
41834	062888	02	61132	04	021700
41834	062888	01	00623	50	025000

Using the Line field as a third key field defines the file so that there are no duplicate keys:

(First Key Field) (Second Key Field) (Third Key Field)

Order	Ordate	Line	Item	Qtyord	Extens
41834	062888	03	42111	300	20550
41834	062888	02	61132	04	21700
41834	062888	01	00623	50	25000

A logical file that has more than one record format can have records with duplicate key values, even though the record formats are based on different physical files. That is, even though the key values come from different record formats, they are considered duplicate key values.

Preventing Duplicate Key Values: The AS/400 database management system allows records with duplicate key values in your files. However, you may want to prevent duplicate key values in some of your files. For example, you can create a file where

the key field is defined as the customer number field. In this case, you want the system to ensure that each record in the file has a unique customer number

You can prevent duplicate key values in your files by specifying the UNIQUE keyword in DDS. With the UNIQUE keyword specified, a record cannot be entered or copied into a file if its key value is the same as the key value of a record already existing in the file. You can also use unique constraints to enforce the integrity of unique keys.

If records with duplicate key values already exist in a physical file, the associated logical file cannot have the UNIQUE keyword specified. If you try to create a logical file with the UNIQUE keyword specified, and the associated physical file contains duplicate key values, the logical file is not created. The system sends you a message stating this and sends you messages (as many as 20) indicating which records contain duplicate key values.

When the UNIQUE keyword is specified for a file, any record added to the file cannot have a key value that duplicates the key value of an existing record in the file, regardless of the file used to add the new record.

For example, two logical files LF1 and LF2 are based on the physical file PF1. The UNIQUE keyword is specified for LF1. If you use LF2 to add a record to PF1, you cannot add the record if it causes a duplicate key value in LF1.

If any of the key fields allow null values, null values that are inserted into those fields may or may not cause duplicates depending on how the access path was defined at the time the file was created. The *INCNULL parameter of the UNIQUE keyword indicates that null values are included when determining whether duplicates exist in the unique access path. The *EXCNULL parameter indicates that null values are not included when determining whether duplicate values exist.

The following shows the DDS for a logical file that requires unique key values:

```
A* ORDER TRANSACTION LOGICAL FILE (ORDFILL)
A                                UNIQUE
A      R ORDHDR                  PFILE(ORDHDRP)
A      K ORDER
A*
A      R ORDDTL                  PFILE(ORDDTLP)
A      K ORDER
A      K LINE
```

In this example, the contents of the key fields (the Order field for the ORDHDR record format, and the Order and Line fields for the ORDDTL record format) must be unique whether the record is added through the ORDHDRP file, the ORDDTLP file, or the logical file defined here. With the Line field specified as a second key field in the ORDDTL record format, the same value can exist in the Order key field in both physical files. Because the physical file ORDDTLP has two key fields and the physical file ORDHDRP has only one, the key values in the two files do not conflict.

Arranging Duplicate Keys: If you do not specify the UNIQUE keyword in DDS, you can specify how the system is to store records with duplicate key values, should they occur. You specify that records with duplicate key values are stored in the access path in one of the following ways:

- Last-in-first-out (LIFO). When the LIFO keyword is specified (1), records with duplicate key values are retrieved in last-in-first-out order by the physical sequence of the records.

Below is an example of DDS using the LIFO keyword.

```
A* ORDERP2
A                    1        LIFO
A       R ORDER2
A       .
A       .
A       .
A       K ORDER
```

- First-in-first-out (FIFO). If the FIFO keyword is specified, records with duplicate key values are retrieved in first-in-first-out order by the physical sequence of the records.

- First-changed-first-out (FCFO). If the FCFO keyword is specified, records with duplicate key values are retrieved in first-changed-first-out order by the physical sequence of the keys.

No specific order for duplicate key fields (the default). When the FIFO, FCFO, or LIFO keywords are not specified, no guaranteed order is specified for retrieving records with duplicate keys. No specific order for duplicate key fields allows more access path sharing, which can improve performance.

Note:

1. ***When a simple- or multiple-format logical file is based on more than one physical file member, records with duplicate key values are read in the order in which the files and members are specified on the DTAMBRS parameter on the Create Logical File (CRTLF) or Add Logical File Member (ADDLFM) command.***

2. *The LIFO or FIFO order of records with duplicate key values is not determined by the sequence of updates made to the contents of the key fields, but solely by the physical sequence of the records in the file member. Assume that a physical file has the FIFO keyword specified (records with duplicate keys are in first-in-first-out order), and that the following shows the order in which records were added to the file:*

Order Records Were Added to File	key Value
1	A
2	B
3	C
4	C
5	D

The sequence of the access path is (FIFO, ascending key):

RecordNummber	Key Value
1	A
2	B
3	C
4	C
5	D

Records 3 and 4, which have duplicate key values, are in FIFO order That is, because record 3 was added to the file before record 4, it is read before record 4. This would become apparent if the records were read in descending order. This could be done by

creating a logical file based on this physical file, with the DESCEND keyword specified in the logical file.

The sequence of the access path is (FIFO, descending key):

RecordNumber	keyValue
5	D
3	C
4	C
2	B
1	A

If physical record 1 is changed such that the key value is C, the sequence of the access path for the physical file is (FIFO, ascending key):

RecordNumber	KeyValue
2	B
1	C
3	C
4	C
5	D

Finally, changing to descending order, the new sequence of the access path for the logical file is (FIFO, descending key):

RecordNumber	KeyValue
5	D
1	C
3	C
4	C
2	B

After the change, record 1 does not appear after record 4, even though the contents of the key field were updated after record 4 was added.

The FCFO order of records with duplicate key values is determined by the sequence of updates made to the contents of the key fields. In the example above, after record 1 is changed such that the key value is C, the sequence of the access path (FCFO, ascending key only) is:

RecordNumber	KeyValue
2	B
3	C
4	C
1	C
5	D

For FCFO, the duplicate key ordering can change when the FCFO access path is rebuilt or when a rollback operation is performed. In some cases, your key field can change but the physical key does not change. In these cases, the FCFO ordering does not change, even though the key field has changed. For example, when the index ordering is changed to be based on the absolute value of the key, the FCFO ordering does not change. The physical value of the key does not change even though your key changes from negative to positive. Because the physical key does not change, FCFO ordering does not change.

If the reuse deleted records attribute is specified for a physical file, the duplicate key ordering must be allowed to default or must be FCFO. The reuse deleted records attribute is not allowed for the physical file if either the key ordering for the file is FIFO or LIFO, or if any of the logical files defined over the physical file have duplicate key ordering of FIFO or LIFO.

Using Existing Access Path Specifications

You can use the DDS keyword REFACCPTH to use another file's access path specifications. When the file is created, the system determines which access path to share. The file using the REFACCPTH keyword does not necessarily share the access path of the file specified in the REFACCPTH keyword. The REFACCPTH keyword is used to simply reduce the number of DDS statements that must be specified. That is, rather than code the key field specifications for the file, you can specify the REFACCPTH keyword.

When the file is created, the system copies the key field and select/omit specifications from the file specified on the REFACCPTH keyword to the file being created.

Database File and Member Attributes: Introduction

When you create a database file, database attributes are stored with the file and members. You specify attributes with database command parameters. For discussions on specifying these attributes and their possible values,

File Name and Member Name (FILE and MBR) Parameters

You name a file with the FILE parameter in the create command. You also name the library in which the file will reside. When you create a physical or logical file, the system normally creates a member with the same name as the file. You can, however, specify a member name with the MBR parameter in the create commands. You can also choose not to create any members by specifying MBR(*NONE) in the create command.

Note: The system does not automatically create a member for a source physical file.

Physical File Member Control (DTAMBRS) Parameter

You can control the reading of the physical file members with the DTAMBRS parameter of the Create Logical File (CRTLF) command. You can specify:

- The order in which the physical file members are to be read.
- The number of physical file members to be used.

Source File and Source Member (SRCFILE and SRCMBR) Parameters

The SRCFILE and SRCMBR parameters specify the names of the source file and members containing the DDS statements that describe the file being created. If you do not specify a name:

- The default source file name is QDDSSRC.
- The default member name is the name specified on the FILE parameter.

Database File Type (FILETYPE) Parameter

A database file type is either data (*DATA) or source (*SRC). The Create Physical File (CRTPF) and Create Logical File (CRTLF) commands use the default data file type (*DATA).

Maximum Number of Members Allowed (MAXMBRS) Parameter

The MAXMBRS parameter specifies the maximum number of members the file can hold. The default maximum number of members for physical and logical files is one, and the default for source physical files is *NOMAX.

Where to Store the Data (UNIT) Parameter

The parameter can still be coded; its presence will not cause an error. It will be ignored.

The system finds a place for the file on auxiliary storage. To specify where to store the file, use the UNIT parameter. The UNIT parameter specifies:

- The location of data records in physical files.
- The access path for both physical files and logical files.

The data is placed on different units if:

- There is not enough space on the unit.
- The unit is not valid for your system.

An informational message indicating that the file was not placed on the requested unit is sent when file members are added. (A message is not sent when the file member is extended.)

Unit Parameter Tips

In general, you should not specify the UNIT parameter. Let the system place the file on the disk unit of its choosing. This is usually better for performance, and relieves you of the task of managing auxiliary storage.

If you specify a unit number and also an auxiliary storage pool, the unit number is ignored. For more information about auxiliary storage pools,

Frequency of Writing Data to Auxiliary Storage (FRCRATIO) Parameter

You can control when database changes are written to auxiliary storage using the force write ratio (FRCRATIO) parameter on either the create, change, or override database file commands. Normally, the system determines when to write changed data from main storage to auxiliary storage. Closing the file (except for a shared close) and the force-end-of-data operation forces remaining updates, deletions, and additions to auxiliary storage. If you are journaling the file, the FRCRATIO parameter should normally be *NONE.

FRCRATIO Parameter Tip

Using the FRCRATIO parameter has performance and recovery considerations for your system.

Frequency of Writing the Access Path (FRCACCPTH) Parameter

The force access path (FRCACCPTH) parameter controls when an access path is written to auxiliary storage. FRCACCPTH(*YES) forces the access path to auxiliary storage whenever the access path is changed. This reduces the chance that the access path will need to be rebuilt should the system fail.

FRCACCPTH Parameter Tips

Specifying FRCACCPTH(*YES) can degrade performance when changes occur to the access path. An alternative to forcing the access path is journaling the access path.

Check for Record Format Description Changes (LVLCHK) Parameter

When the file is opened, the system checks for changes to the database file definition. When the file changes to an extent that your program may not be able to process the file, the system notifies your program. The default is to do level checking. You can specify if you want level checking when you:

- Create a file.
- Use a change database file command.

You can override the system and ignore the level check using the Override with Database File (OVRDBF) command.

Level Check Example

For example, assume you compiled your program two months ago and, at that time, the file the program was defined as having three fields in each record. Last week another programmer decided to add a new field to the record format, so that now each record would have four fields. The system notifies your program, when it tries to open the file, that a significant change occurred to the definition of the file since the last time the program was compiled. This notification is known as a record format level check.

Current Access Path Maintenance (MAINT) Parameter

The MAINT parameter specifies how access paths are maintained for closed files. While a file is open, the system maintains the access paths as changes are made to the data in the file. However, because more than one access path can exist for the same data, changing data in one file might cause changes to be made in access paths for other files that are not currently open (in use).

The three ways of maintaining access paths of closed files are:

- Immediate maintenance of an access path means that the access path is maintained as changes are made to its associated data, regardless if the file is open. Access paths used by referential constraints will always be in immediate maintenance.

- Rebuild maintenance of an access path means that the access path is only maintained while the file is open, not when the file is closed; the access path is rebuilt when the file is opened the next time. When a file with rebuild maintenance is closed, the system stops maintaining the access path. When the file is opened again, the access path is totally rebuilt. If one or more programs has opened a specific file member with rebuild maintenance specified, the system maintains the access path for that member until the last user closes the file member.

- Delayed maintenance of an access path means that any maintenance for the access path is done after the file member is opened the next time and while it remains open. However, the access path is not rebuilt as it is with rebuild maintenance. Updates to the access path are collected from the time the member is closed until it is opened again. When it is opened, only the collected changes are merged into the access path.

If you do not specify the type of maintenance for a file, the default is immediate maintenance.

MAINT Parameter Tips

The type of access path maintenance to specify depends on the number of records and the frequency of additions, deletions, and updates to a file while the file is closed.

You should use delayed maintenance for files that have relatively few changes to the access path while the file members are closed. Delayed maintenance reduces system overhead by reducing the number of access paths that are maintained immediately. It may also result in faster open processing, because the access paths do not have to be rebuilt.

You may want to specify immediate maintenance for access paths that are used frequently, or when you cannot wait for an access path to be rebuilt when the file is opened. You may want to specify delayed maintenance for access paths that are not used frequently, if infrequent changes are made to the record keys that make up the access path.

In general, for files used interactively, immediate maintenance results in good response time. For files used in batch jobs, either immediate, delayed, or rebuild maintenance is adequate, depending on the size of the members and the frequency of changes.

Access Path Recovery (RECOVER) Parameter

After a failure, changed access paths that were not forced to auxiliary storage or journaled cannot be used until they are rebuilt. The RECOVER parameter on the Create Physical File (CRTPF), the Create Logical File (CRTLF), and the Create Source Physical File (CRTSRCPF) commands specifies when that access path is to be rebuilt. Access paths are rebuilt either during the initial program load (IPL), after the IPL, or when a file is opened.

RECOVER Parameter Tip

A list of files that have access paths that need to be recovered is shown on the Edit Rebuild of Access Paths display during the next initial program load (IPL) if the IPL is in manual mode. You can edit the original recovery option for the file by selecting the desired option on the display. After the IPL is complete, you can use the Edit Rebuild of Access Paths (EDTRBDAP) command to set the sequence in which access paths are rebuilt. If the IPL is unattended, the Edit Rebuild of Access Paths display is not shown and the access paths are rebuilt in the order determined by the RECOVER parameter. You only see the *AFTIPL and *NO (open) access paths.

File Sharing (SHARE) Parameter

The database system lets multiple users access and change a file at the same time. The SHARE parameter allows sharing of opened files in the same job.

For example, sharing a file in a job allows programs in the job to share a file's status, record position, and buffer. Sharing files in a job can improve performance by reducing:

- The amount of storage the job needs.
- The time required to open and close the file.

Locked File or Record Wait Time (WAITFILE and WAITRCD) Parameters

When you create a file, you can specify how long a program should wait for either the file or a record in the file if another job has the file or record locked. If the wait time ends before the file or record is released, a message is sent to the program indicating that the job was not able to use the file or read the record.

Public Authority (AUT) Parameter

When you create a file, you can specify public authority. Public authority is the authority a user has to a file (or other object on the system) if that user does not have specific authority for the file or does not belong to a group with specific authority for the file.

System on Which the File Is Created (SYSTEM) Parameter

You can specify if the file is to be created on the local system or a remote system that supports distributed data management (DDM).

File and Member Text (TEXT) Parameter

You can specify a text description for each file and member you create. The text data is useful in describing information about your file and members.

Coded Character Set Identifier (CCSID) Parameter

You can specify a coded character set identifier (CCSID) for physical files. The CCSID describes the encoding scheme and the character set for character type fields contained in this file.

Sort Sequence (SRTSEQ) Parameter

You can specify the sort sequence for a file. The values of the SRTSEQ parameter along with the CCSID and LANGID parameters determine which sort sequence table

the file uses. You can set the SETSEQ parameter for both the physical and the logical files.

You can specify:

- System supplied sort sequence tables with unique or shared collating weights. There are sort sequence tables for each supported language.
- Any user-created sort sequence table.
- The hexadecimal value of the characters in the character set.
- The sort sequence of the current job or the one specified in the ALTSEQ parameter.

The sort sequence table is stored with the file, except when the sort sequence is *HEX.

Language Identifier (LANGID) Parameter

You can specify the language identifier that the system should use when the SRTSEQ parameter value is *LANGIDSHR or *LANGIDUNQ. The values of the LANGID, CCSID, and SRTSEQ parameters determine which sort sequence table the file uses. You can set the LANGID parameter for physical and logical files.

You can specify any language identifier supported on your system, or you can specify that the language identifier for the current job be used.

CREATING A PHYSICAL FILE

1. If you are using DDS, enter DDS for the physical file into a source file. This can be done using the AS/400 Application Development Tools source entry utility (SEU).
2. Create the physical file. You can use the Create Physical File (CRTPF) command, or the Create Source Physical File (CRTSRCPF) command.

The following command creates a one-member file using DDS and places it in a library called DSTPRODLB.

CRTPF FILE(DSTPRODLB/ORDHDRP)
TEXT('Order header physical file')

As shown, this command uses defaults. For the SRCFILE and SRCMBR parameters, the system uses DDS in the source file called QDDSSRC and the member named ORDHDRP (the same as the file name). The file ORDHDRP with one member of the same name is placed in the library DSTPRODLB.

PHYSICAL FILE AND MEMBER ATTRIBUTES

Some of the attributes you can specify for physical files and members on the Create Physical File (CRTPF), Create Source Physical File (CRTSRCPF), Change Physical File (CHGPF), Change Source Physical File (CHGSRCPF), Add Physical File Member (ADDPFM), and Change Physical File Member (CHGPFM) commands include (command names are given in parentheses):

Expiration Date

EXPDATE Parameter. This parameter specifies an expiration date for each member in the file (ADDPFM, CHGPFM, CRTPF, CHGPF, CRTSRCPF, and CHGSRCPF commands). If the expiration date is past, the system operator is notified when the file is

opened. The system operator can then override the expiration date and continue, or stop the job. Each member can have a different expiration date, which is specified when the member is added to the file.

Size of the Physical File Member

SIZE Parameter. This parameter specifies the maximum number of records that can be placed in each member (CRTPF, CHGPF, CRTSRCPF, AND CHGSRCPF commands). The following formula can be used to determine the maximum:

R + (I * N)

where:

R is the starting record count

I is the number of records (increment) to add each time

N is the number of times to add the increment

The defaults for the SIZE parameter are:

R 10,000

I 1,000

N 3 (CRTPF command)

499 (CRTSRCPF command)

For example, assume that R is a file created for 5000 records plus 3 increments of 1000 records each. The system can add 1000 to the initial record count of 5000 three times to make the total maximum 8000. When the total maximum is reached, the system operator either stops the job or tells the system to add another increment of records and continue. When increments are added, a message is sent to the system history log. When the file is extended beyond its maximum size, the minimum extension is 10% of the current size, even if this is larger than the specified increment.

Instead of taking the default size or specifying a size, you can specify *NOMAX.

Storage Allocation

ALLOCATE Parameter. This parameter controls the storage allocated for members when they are added to the file (CRTPF, CHGPF, CRTSRCPF, and CHGSRCPF commands). The storage allocated would be large enough to contain the initial record count for a member. If you do not allocate storage when the members are added, the system will automatically extend the storage allocation as needed. You can use the ALLOCATE parameter only if you specified a maximum size on the SIZE parameter. If SIZE(*NOMAX) is specified, then ALLOCATE(*YES) cannot be specified.

Method of Allocating Storage

CONTIG Parameter. This parameter controls the method of allocating physical storage for a member (CRTPF and CRTSRCPF commands). If you allocate storage, you can request that the storage for the starting record count for a member be contiguous. That is, all the records in a member are to physically reside together. If there is not enough contiguous storage, contiguous storage allocation is not used and an informational message is sent to the job that requests the allocation, at the time the member is added. Note: When a physical file is first created, the system always tries to allocate its initial storage contiguously. The only difference between using CONTIG(*NO) and CONTIG(*YES) is that with CONTIG(*YES) the system sends a message to the job log if it is unable to allocate contiguous storage when the file is created. No message is sent when a file is extended after creation, regardless of what you specified on the CONTIG parameter.

Record Length

RCDLEN Parameter. This parameter specifies the length of records in the file (CRTPF and CRTSRCPF commands). If the file is described to the record level only, then you specify the RCDLEN parameter when the file is created. This parameter cannot be

specified if the file is described using DDS, IDDU, or SQL (the system automatically determines the length of records in the file from the field level descriptions).

Deleted Records

DLTPCT Parameter. This parameter specifies the percentage of deleted records a file can contain before you want the system to send a message to the system history log (CRTPF, CHGPF, CRTSRCPF, and CHGSRCPF commands). When a file is closed, the system checks the member to determine the percentage of deleted records. If the percentage exceeds that value specified in the DLTPCT parameter, a message is sent to the history log. One reason you might want to know when a file reaches a certain percentage of deleted records is to reclaim the space used by the deleted records. After you receive the message about deleted records, you could run the Reorganize Physical File Member (RGZPFM) command to reclaim the space. You can also specify to bypass the deleted records check by using the *NONE value for the DLTPCT parameter. *NONE is the default for the DLTPCT parameter.

REUSEDLT Parameter. This parameter specifies whether deleted record space should be reused on subsequent write operations (CRTPF and CHGPF commands). When you specify *YES for the REUSEDLT parameter, all insert requests on that file try to reuse deleted record space. Reusing deleted record space allows you to reclaim space used by deleted records without having to issue a RGZPFM command. When the CHGPF command is used to change a file to reuse deleted records, the command could take a long time to run, especially if the file is large and there are already a lot of deleted records in it.

It is important to note the following:

- The term arrival order loses its meaning for a file that reuses deleted record space. Records are no longer always inserted at the end of the file when deleted record space is reused.

- If a new physical file is created with the reuse deleted record space attribute and the file is keyed, the FIFO or LIFO access path attribute cannot be specified for the physical file, nor can any keyed logical file with the FIFO or LIFO access path attribute be built over the physical file.

- You cannot change an existing physical file to reuse deleted record space if there are any logical files over the physical file that specify FIFO or LIFO ordering for duplicate keys, or if the physical file has a FIFO or LIFO duplicate key ordering.
- Reusing deleted record space should not be specified for a file that is processed as a direct file or if the file is processed using relative record numbers.

*NO is the default for the REUSEDLT parameter.

Physical File Capabilities

ALWUPD and ALWDLT Parameters. File capabilities are used to control which input/output operations are allowed for a database file independent of database file authority. For more information about database file capabilities and authority, see Chapter 4, "Database Security."

Source Type

SRCTYPE Parameter. This parameter specifies the source type for a member in a source file (ADDPFM and CHGPFM commands). The source type determines the syntax checker, prompting, and formatting that are used for the member. If the user specifies a

unique source type (other than AS/400 supported types like COBOL and RPG), the user must provide the programming to handle the unique type.

If the source type is changed, it is only reflected when the member is subsequently opened; members currently open are not affected.

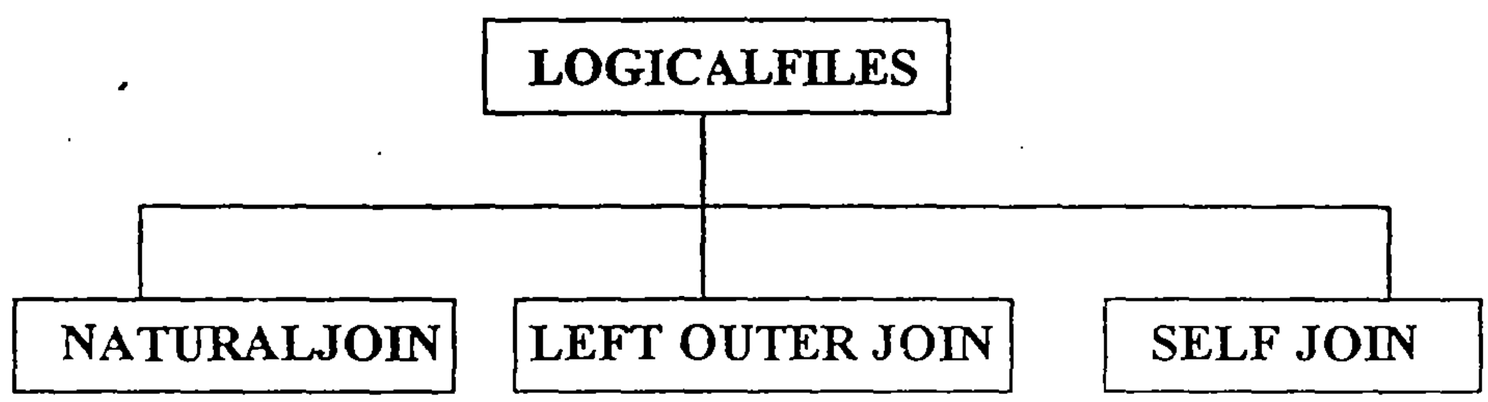
LOGICAL FILES

RECORD FORMAT DEFINITION / PHYSICAL FILE SELECTION

To define a logical file, you must select the record formats to be used and the physical files to be referenced. You can use the record format found in the physical file, or you can use a new record formats. If you use the thysical file record format, every field in the record format is accessible through the logical file. If you create a new record format. You must specify which fields will exists in the logical file . A logical file field must either reference a field in the physical file record format or be derived by using concatenation or substring functions.

Because the logical file does not contain any data, it must know any which physical file to request for the requested data. You use the DDS PFILE keywords to select the physical file referenced by the logical file record format.

DESCRIBING LOGICAL FILE RECORD FORMATS



LOGICAL FILE KEYWORDS :

Keywords used in logical files have six distinct levels. They are :

- File level keywords
- Record format level keywords
- Join level keywords (for join logical files only)

- Field level keywords
- Key field keywords
- Select / Omit level keywords

There are over 40 keywords associated with logical files. These keywords are used to provide further qualification to the file, record format, field, key field or select/omit condition. Some of the more common keywords are :

File Level Keywords:

DYNSLT (Dynamic Selection) :

This keyword tells the AS/400 to perform select/omit testing for the index associated with the logical file at processing time(dynamically).

JDFLT (Join default values) :

This keyword is for join logical files only. It tells the AS/400 that when joining records together, if a join to a secondary record does not occur, the system will provide default values for fields retrieved from the secondary file.

Record Format Level Keywords:

JFILE (Joined Files) :

This keyword is for join logical files only. It specifies all the physical files that make up the join-logical file.

PFILE (Physical File):

This keyword specifies all the physical files that make up a logical file. This is used with single or multiple format logical files. PFILE is not allowed with join-logical file.

Join Level Keywords:

JDUPSEQ (Join Duplicate Sequence) :

This keyword specifies the order in which duplicate records are arranged for a join-logical file.

JFLD (Join Fields) :

This keyword specifies how physical files are to be joined together.

JOIN

This keyword specifies the two physical files that will be joined together.

Field Level Keywords:

CONCAT (Concatenate) :

This keyword lets you take two or more physical file field names and concatenate them into a single field within the logical file.

JREF (Join Reference) :

This keyword qualifies the file name from which a field is to be retrieved.

RENAME

This keyword renames a field name to some other name.

SST (Sub string):

This keyword extracts a character string that is a subset of an existing field.

Key Field Level Keywords:

SIGNED

This keyword tells the AS/400 to consider the sign of a numeric field, when creating index for the logical file.

UNSIGNED

This keyword tells the AS/400 to treat numeric fields as a string of binary data, not numbers when creating the index for the logical file.

Select / Omit Level Keywords:

ALL

This keyword tells the action to be taken after all the Select/Omit specifications have been processed.

COMP (Comparison) :

This keyword is used to define a comparison test to see whether a record should be included/excluded from the index.

RANGE

This keyword instructs the AS/400 to include/exclude records in the index based on the range of values entered.

VALUES

This affects how records are included / excluded from the index created for this logical file.

For every logical file record format described with DDS, you must specify a record format name and either the PFILE keyword (for simple and multiple format logical files), or the JFILE keyword (for join logical files). The file names specified on the PFILE or JFILE keyword are the physical files that the logical file is based on. A simple or multiple-format logical file record format can be specified with DDS in any one of the following ways:

1. In the simple logical file record format, specify only the record format name and the PFILE keyword. The record format for the only (or first) physical file specified on the PFILE keyword is the record format for the logical file. The record format name specified in the logical file must be the same as the record format name in the only (or first) physical file.

```
A       R ORDDTL        PFILE(ORDDTLP)
A*
```

Figure L.F.1. Simple Logical File

2. In the following example, you describe your own record format by listing the field names you want to include. You can specify the field names in a different order, rename fields using the RENAME keyword, combine fields using the CONCAT keyword, and use specific positions of a field using the SST keyword. You can also override attributes of the fields by specifying different attributes in the logical file.

```
A       R ORDHDR        PFILE(ORDHDRP)
A       ORDER
A       CUST
A       SHPVIA
A*
```

Figure L.F.2. Simple Logical File with Fields Specified

In the following example, the file name specified on the FORMAT keyword is the name of a database file. The record format is shared from this database file by the logical file being described. The file name can be qualified by a library name. If a library name is not specified, the library list is used to find the file. The file must exist when the file you are describing is created. In addition, the record format name you specify in the logical file must be the same as one of the record format names in the file you specify on the FORMAT keyword.

```
A       R CUSRCD        PFILE(CUSMSTP)
A                       FORMAT(CUSMSTL)
```

In the following example, a program needs:

- The fields placed in a different order
- A subset of the fields from the physical file
- The data types changed for some fields
- The field lengths changed for some fields

You can use a logical file to make these changes.

Logical

File

FIELD D	FIELD A	FIELD C
Data type:	*Data type:*	*Data type:*
Zoned decimal	Zoned decimal	Zoned decimal
Length: 10,0	Length: 8,2	Length: 5,0

Physical

File

FIELD A	FIELD B	FIELD C	FIELD D
Data type:	*Data type:*	*Data type:*	*Data type:*
Zoned decimal	Character	Binary	Character
Length: 8,2	Length: 32	Length: 2,0	Length: 10

For the logical file, the DDS would be:

```
A       R LOGREC          PFILE(PF1)
A         D    10S 0
A         A    8S 2
A         C    5S 0
```

For the physical file, the DDS would be:

```
A       R PHYREC
A         A    8S 2
A         B    32A
A         C    2B 0
A         D    10S 0
```

When a record is read from the logical file, the fields from the physical file are changed to match the logical file description. If the program updates or adds a record, the fields are changed back. For an add or update operation using a logical file, the program must supply data that conforms with the format used by the logical file.

DESCRIBING FIELD USE FOR LOGICAL FILES

You can specify that fields in database files are to be input-only, both (input/output), or neither fields. Do this by specifying one of the following in position 38:

Entry	Meaning
Blank	For simple or multiple format logical files, defaults to B (both) For join logical files, defaults to I (input only)
B	Both input and output allowed; not valid for join logical files
I	Input only (read only)
N	Neither input nor output; valid only for join logical files

Note: The usage value (in position 38) is not used on a reference function. When another file refers to a field (using a REF or REFFLD keyword) in a logical file, the usage value is not copied into that file.

Both

A both field can be used for both input and output operations. Your program can read data from the field and write data to the field. Both fields are not valid for join logical files, because join logical files are read-only files.

Input Only

An input only field can be used for read operations only. Your program can read data from the field, but cannot update the field in the file. Typical cases of input-only fields are key fields (to reduce maintenance of access paths by preventing changes to key field values), sensitive fields that a user can see but not update (for example, salary), and fields for which either the translation table (TRNTBL) keyword or the substring (SST) keyword is specified.

If your program updates a record in which you have specified input-only fields, the input-only fields are not changed in the file. If your program adds a record that has input-only fields, the input-only fields take default values (DFT keyword).

Neither

A neither field is used neither for input nor for output. It is valid only for join logical files. A neither field can be used as a join field in a join logical file, but your program cannot read or update a neither field.

Use neither fields when the attributes of join fields in the physical files do not match. In this case, one or both join fields must be defined again. However, you cannot include these redefined fields in the record format (the application program does not see the redefined fields.) Therefore, redefined join fields can be coded N so that they do not appear in the record format. A field with N in position 38 does not appear in the buffer used by your program. However, the field description is displayed with the Display File Field Description (DSPFFD) command.

Neither fields cannot be used as select/omit or key fields.

DERIVING NEW FIELDS FROM EXISTING FIELDS

Fields in a logical file can be derived from fields in the physical file the logical file is based on or from fields in the same logical file. For example, you can concatenate, using the CONCAT keyword, two or more fields from a physical file to make them appear as one field in the logical file. Likewise, you can divide one field in the physical file to make it appear as multiple fields in the logical file with the SST keyword.

Concatenated Fields

Using the CONCAT keyword, you can combine two or more fields from a physical file record format to make one field in a logical file record format. For example, a physical file

record format contains the fields Month, Day, and Year. For a logical file, you concatenate these fields into one field, Date.

The field length for the resulting concatenated field is the sum of the lengths of the included fields (unless the fields in the physical file are binary or packed decimal, in which case they are changed to zoned decimal). The field length of the resulting field is automatically calculated by the system. A concatenated field can have:

- Column headings
- Validity checking
- Text description
- Edit code or edit word (numeric concatenated fields only)

Note: This editing and validity checking information is not used by the database management system but is retrieved when field descriptions from the database file are referred to in a display or printer file.

When fields are concatenated, the data types can change (the resulting data type is automatically determined by the system).

00101A	**MONTH**	
00102A	**DAY**	
00103A	**YEAR**	
00104A	**DATE**	**CONCAT(MONTH DAY YEAR)**

In this example, the logical file record format includes the separate fields of Month, Day, and Year, as well as the concatenated Date field. Any of the following can be used:

- A format with the separate fields of Month, Day, and Year
- A format with only the concatenated Date field
- A format with the separate fields Month, Day, Year and the concatenated Date field

concatenated fields can also be used as key fields and select /omit fields.

Substring Fields

You can use the SST keyword to specify which fields (character,hexadecimal, c zoned decimal) are in a substring. (You can also use substring with a packed field in a hysical file by specifying S (zoned decimal) as the data type in the logical file.) For e ample, assume you defined the Date field in physical file PF1 as 6 characters in length. ou can describe the logical file with three fields, each 2 characters in length. You can use e SST keyword to define MM as 2 characters starting in position 1 of the Date field, I D as 2 characters starting in position 3 of the Date field, and YY as 2 characters sta ing in position 5 of the Date field.

The following shows the field descriptions in DDS for these substring fields. T e SST keyword is used to specify the field to substring.

```
A      R REC1              PFILE(PF1)
A
A      MM          I       SST(DATE 1 2)
A      DD          I       SST(DATE 3 2)
A      YY          I       SST(DATE 5 2)
```

Note: The starting position of the substring is specified according to its positio n the field being operated on (Date), not according to its position in the file. The I n the Usage column indicates input-only.

Substring fields can also be used as key fields and select/omit fields.

Renamed Fields

You can name a field in a logical file differently than in a physical file using the RE AME keyword. You might want to rename a field in a logical file because the program w

written using a different field name or because the original field name does not conform to the naming restrictions of the high-level language you are using.

```
A       R CUSRCD        PFILE(CUSMSTP)
A       FLD1            RENAME(F1)
```

DESCRIBING ACCESS PATHS FOR LOGICAL FILES

The access path for a logical file record format can be specified in one of three ways:

1. Keyed sequence access path specification. Specify key fields after the last record or field level specification. The key field names must be in the record format. For join logical files, the key fields must come from the first, or primary, physical file.

```
A       R CUSRCD        PFILE(CUSMSTP)
A       K ARBAL
A       K CRDLMT
A*
```

2. Arrival sequence access path specification. Specify no key fields. You can specify only one physical file on the PFILE keyword (and only one of the physical file's members when you add the logical file member).

```
A       R CUSRCD        PFILE(CUSMSTP)
```

3. Previously defined keyed-sequence access path specification (for simple and multiple format logical files only). Specify the REFACCPTH keyword at the file level to identify a previously created database file whose access path and select/omit specifications are to be copied to this logical file. You cannot specify individual key or select/omit fields with the REFACCPTH keyword.

Note: Even though the specified file's access path specifications are used, the system determines which file's access path, if any, will actually be shared. The system always tries to share access paths, regardless of whether the REFACCPTH keyword is used.

```
                                REFACCPTH(DSTPRODLIB/ORDHDRL)
A       R CUSRCD                PFILE(CUSMSTP)
```

When you define a record format for a logical file that shares key field specifications of another file's access path (using the DDS keyword, REFACCPTH), you can use any fields from the associated physical file record format. These fields do not have to be used in the file that describes the access path. However, all key and select/omit fields used in the file that describes the access path must be used in the new record format.

SELECTING AND OMITTING RECORDS USING LOGICAL FILES

The system can select and omit records when using a logical file. This can help you to exclude records in a file for processing convenience or for security.

The process of selecting and omitting records is based on comparisons identified in position 17 of the DDS Form for the logical file, and is similar to a series of comparisons coded in a high-level language program.

For example, in a logical file that contains order detail records, you can specify that the only records you want to use are those in which the quantity ordered is greater than the quantity shipped. All other records are omitted from the access path. The omitted records remain in the physical file but are not retrieved for the logical file. If you are adding records to the physical file, all records are added, but only selected records that match the select/omit criteria can be retrieved using the select/omit access path.

In DDS, to specify select or omit, you specify an S (select) or O (omit) in position 17 of the DDS form. You then name the field (in positions 19 through 28) that will be used in the selection or omission process. In positions 45 through 80 you specify the comparison.

Note: Select/omit specifications appear after key specifications (if keys are specified).

Records can be selected and omitted by several types of comparisons:

- **VALUES.** The contents of the field are compared to a list of not more than 100 values. If a match is found, the record is selected or omitted. In the following example, a record is selected if one of the values specified in the VALUES keyword is found in the Itmnbr field.

```
A       S ITMNBR          VALUES(301542 306902 382101 422109 +
A                         431652 486592 502356 556608 590307)
A
```

- **RANGE.** The contents of the field are compared to lower and upper limits. If the contents are greater than or equal to the lower limit and less than or equal to the upper limit, the record is selected or omitted. In the following example, all records with a range 301000 through 599999 in the Itmnbr field are selected.

```
A       S ITMNBR          RANGE(301000 599999)
A
```

- **CMP.** The contents of a field are compared to a value or the contents of another field. Valid comparison codes are EQ, NE, LT, NL, GT, NG, LE, and GE. If the

comparison is met, the record is selected or omitted. In the following example, a record is selected if its Itmnbr field is less than or equal to 599999:

```
A        S ITMNBR              CMP(LE 599999)
```

The value for a numeric field for which the CMP, VALUES, or RANGE keyword is specified is aligned based on the decimal positions specified for the field and filled with zeros where necessary. If decimal positions were not specified for the field, the decimal point is placed to the right of the farthest right digit in the value. For example, for a numeric field with length 5 and decimal position 2, the value 1.2 is interpreted as 01.20 and the value 100 is interpreted as 100.00.

The status of a record is determined by evaluating select/omit statements in the sequence you specify them. If a record qualifies for selection or mission, subsequent statements are ignored.

Normally the select and omit comparisons are treated independently from one another; the comparisons are ORed together. That is, if the select or omit comparison is met, the record is either selected or omitted. If the condition is not met, the system proceeds to the next comparison. To connect comparisons together, you simply leave a space in position 17 of the DDS Form. Then, all the comparisons that were connected in this fashion must be met before the record is selected or omitted. That is, the comparisons are ANDed together.

The fewer comparisons, the more efficient the task is. So, when you have several select/omit comparisons, try to specify the one that selects or omits the most records first. In the following examples, few records exist for which the Rep field is JSMITH. The examples show how to use DDS to select all the records before 1988 for a sales representative named JSMITH in the state of New York.

All give the same results with different efficiency (in this example, 3 is the most efficient).

```
A      S ST          CMP(EQ 'NY')       1
A       REP          CMP(EQ 'JSMITH')
A       YEAR         CMP(LT 88)
A*


A      O YEAR        CMP(GE 88)         2
A      S ST          CMP(EQ 'NY')
A       REP          CMP(EQ 'JSMITH')
A*


A      O REP         CMP(NE 'JSMITH')   3
A      O ST          CMP(NE 'NY')
A      S YEAR        CMP(LT 88)
A*
```

Figure L. F .3. Three Ways to Code Select/Omit Function

1. All records must be compared with all of the select fields St, Rep, and Year before they can be selected or omitted.

2. All records are compared with the Year field. Then, the records before 1988 have to be compared with the St and Rep fields.

3. All records are compared with the Rep field. Then, only the few for JSMITH are compared with the St field. Then, the few records that are left are compared to the Year field.

Access Path Select/Omit

With access path select/omit, the access path only contains keys that meet the select/omit values specified for the logical file. When you specify key fields for a file, an access path is kept for the file and maintained by the system when you add or update records in the physical file(s) used by the logical file. The only index entries in the access path are those that meet the select/omit values.

Dynamic Select/Omit

With dynamic select/omit, when a program reads records from the file, the system only returns those records that meet the select/omit values. That is, the actual select/omit processing is done when records are read by a program, rather than when the records are added or changed. However, the keyed sequence access path contains all the keys, not just keys from selected records. Access paths using dynamic select/omit allow more access path sharing, which can improve performance.

To specify dynamic select/omit, use the dynamic selection (DYNSLT) keyword. With dynamic select/omit, key fields are not required.

If you have a file that is updated frequently and read infrequently, you may not need to update the access path for select/omit purposes until your program reads the file. In this case, dynamic select/omit might be the correct choice.

USING EXISTING ACCESS PATHS

When two or more files are based on the same physical files and the same key fields in the same order, they automatically share the same keyed sequence access path. When access

paths are shared, the amount of system activity required to maintain access paths and the amount of auxiliary storage used by the files is reduced.

When a logical file with a keyed sequence access path is created, the system always tries to share an existing access path.

CREATING A LOGICAL FILE

Before creating a logical file, the physical file or files on which the logical file is based must already exist.

To create a logical file, take the following steps:

Type the DDS for the logical file into a source file. This can be done using SEU or another method.

```
A* ORDER HEADER LOGICAL FILE (ORDHDRL)
A        R ORDHDR          PFILE(ORDHDRP)
A        K ORDER
```

1. This file uses the key field Order (order number) to define the access path. The record format is the same as the associated physical file ORDHDRP. The record format name for the logical file must be the same as the record format name in the physical file because no field descriptions are given.
2. Create the logical file. You can use the Create Logical File (CRTLF) command.

The following shows how the CRTLF command could be typed:

CRTLF FILE(DSTPRODLB/ORDHDRL)

TEXT('Order header logical file')

As shown, this command uses some defaults. For example, because the SRCFILE and SRCMBR parameters are not specified, the system used DDS from the IBM-supplied

source file QDDSSRC, and the source file member name is ORDHDRL (the same as the file name specified on the CRTLF command). The file ORDHDRL with one member of the same name is placed in the library DSTPRODLB.

CREATING A LOGICAL FILE WITH MORE THAN ONE RECORD FORMAT

A multiple format logical file lets you use related records from two or more physical files by referring to only one logical file. Each record format is always associated with one or more physical files. You can use the same physical file in more than one record format.

```
A* ORDER DETAIL FILE (ORDDTLP) - PHYSICAL FILE RECORD
A*DEFINITION
A                          REF(DSTREF)
A       R ORDDTL           TEXT('Order detail record')
A         CUST     R
A         ORDER    R
A         LINE     R
A         ITEM     R
A         QTYORD   R
A         DESCRP   R
A         PRICE    R
A         EXTENS   R
A         WHSLOC   R
A         ORDATE   R
A         CUTYPE   R
A         STATE    R
A         ACTMTH   R
A         ACTYR    R
A
```

Figure L.F.4. DDS for a Physical File (ORDDTLP) Built from a Field Reference File

```
A* ORDER HEADER FILE (ORDHDRP) - PHYSICAL FILE RECORD
A*DEFINITION
A                          REF(DSTREFP)
A       R ORDHDR           TEXT('Order header record')
A         CUST       R
A         ORDER      R
A         ORDATE     R
A         CUSORD     R
A         SHPVIA     R
A         ORDSTS     R
A         OPRNME     R
A         ORDMNT     R
A         CUTYPE     R
A         INVNBR     R
A         PRTDAT     R
A         SEQNBR     R
A         OPNSTS     R
A         LINES      R
A         ACTMTH     R
A         ACTYR      R
A         STATE      R
```

Figure L.F.5. DDS for a Physical File (ORDHDRP) Built from a Field Reference File

The following example shows how to create a logical file ORDFILL with two record formats. One record format is defined for order header records from the physical file ORDHDRP; the other is defined for order detail records from the physical file ORDDTLP. (Figure L.F.4 shows the DDS for the physical file ORDDTLP, Figure L.F.5 shows the DDS for the physical file ORDHDRP, and Figure 3-8 shows the DDS for the logical file ORDFILL.)

The logical file record format ORDHDR uses one key field, Order, for sequencing; the logical file record format ORDDTL uses two keys fields, Order and Line, for sequencing.

```
A* ORDER TRANSACTION LOGICAL FILE (ORDFILL)
A        R ORDHDR          PFILE(ORDHDRP)
A        K ORDER
A*
A        R ORDDTL          PFILE(ORDDTLP)
A        K ORDER
A        K LINE
```

Figure L.F.5. DDS for the Logical File ORDFILL

To create the logical file ORDFILL with two associated physical files, use a Create Logical File (CRTLF) command like the following:

CRTLF FILE(DSTPRODLB/ORDFILL)
TEXT('Order transaction logical file')

The DDS source is in the member ORDFILL in the file QDDSSRC. The file ORDFILL with a member of the same name is placed in the DSTPRODLB library.

The access path for the logical file member ORDFILL arranges records from both the ORDHDRP and ORDDTLP files. Record formats for both physical files are keyed on Order as the common field. Because of the order in which they were specified in the logical file description, they are merged in Order sequence with duplicates between files retrieved first from the header file ORDHDRP and second from the detail file ORDDTLP.

CONCEPT OF JOIN

A join is generally defined as the ability to logically concatenate information from records in multiple files based on a relationship between the values in the records. An ideal join facility would link files in a single record format,thus allowing an application program to get all the data from the related records in a single read operation. In addition with a single record format it is desirable to be able to use any field in the record format as a key in select/omit criteria,regardless of which underlying physical file the field comes from.

An ideal join would also allow files tobe linked by specifying logically related fields, without requiring that those fields be key fields in either the physical file or the resulting join file.

In addition anideal join would allow either inner join or outer join. An inner join includes only records from the two files that match. An outer join includes all records from both the files regardless of there is a match;default values are used to fill the fields from the missing record.

A subste of outer join is left outer join. A left outer join includes all records from the first file in the join, even if no matching record exists in the second file. Records from the second file are included in the join only if a corresponding record exists in the primary file.

CHARACTERISTICS OF JOIN LOGICAL FILE

Now that you know the features that the ideal join facility should provide,let's see how the AS/400 join logical file compares. The primary differences between an ideal join and an AS/400 join logical file are that an AS/400 join logical file does not support a full outer join , and it allows only fields contained in the first file to be used as key fields join logical file.

These are the charterstic features of AS/400 join logical files.

Join logical files can be based on 2 to 32 physical files (Thus one AS/400 join logical file acutally specifies up to 31 two file join operations.) These physical files may be in key or arrival sequence, and predefined keys need not be used to link files. The same physical file can be spedcified as a based- on file more than once, so a file can be joined to itself.

Join logical files support inner and left outer join only. Outer join is not supported.

Select/omit criteria can be specified for any field in a join logical family.

Key fields, if specified, must come from the primary file.

Join logical files are read-only files and cannot be used to update files.

The Benefits of join logical files fall into three categories : Increased productivity, more flexible database design, and improved performance.

Increased productivity . HLL programs that process multiple files for displays and reports can be simplified by using join logical files. Instead of having to open and read several files (or record format), the program can open a single join logical file so that only on HLL read is required to get all of the data.

More Flexible database design. Because join logical files provede more ways to access data in physical files, it is possible for physical files to conform more closely to normalized database design, Greater database nomalization means that databases can be designed to further reduce " Fact " redundancy. In addition , fuly normalized files allow more complex inquireis into the database, although physical files or nonjoin logical files remain the only means of updating the database. Also, because a join logical file can provide access to the database that would otherwise require database modification, the database is more insulatedfrom changes in program design than before.

Improved performance. Because HLL programs require fewer database read operations (.g., READ CHAIN), performance is improved. A Single HLL read operation caused multiple VMC (vertical Microcode) reads, which is more efficient and incurs less overhead that multiple HLL reads. In addition, if a program has fewer open data paths, the Job's PAG (Process Access Group) size is reduced, which means more efficient use of main storage and faster program loading.

Basic Concepts of Joining Two Physical Files

A join logical file is a logical file that combines (in one record format) fields from two or more physical files. In the record format, not all the fields need to exist in all the physical files.

The following example illustrates a join logical file that joins two physical files. This example is used for the five cases discussed in **Example 1.**

JLF

Employee Number
Name
Salary

PF1	**PF2**
Employee Number	Employee Number
Name	Salary

In this example, employee number is common to both physical files (PF1 and PF2), but name is found only in PF1, and salary is found only in PF2.

With a join logical file, the application program does one read operation (to the record format in the join logical file) and gets all the data needed from both physical files. Without the join specification, the logical file would contain two record formats, one based on PF1 and the other based on PF2, and the application program would have to do two read operations to get all the needed data from the two physical files. Thus, join provides more flexibility in designing your database.

However, a few restrictions are placed on join logical files:

- You cannot change a physical file through a join logical file. To do update, delete, or write (add) operations, you must create a second multiple format logical file and use it to change the physical files. You can also use the physical files, directly, to do the change operations.
- You cannot use DFU to display a join logical file.
- You can specify only one record format in a join logical file.
- The record format in a join logical file cannot be shared.
- A join logical file cannot share the record format of another file.
- Key fields must be fields defined in the join record format and must be fields from the first file specified on the JFILE keyword (which is called the primary file).
- Select/omit fields must be fields defined in the join record format, but can come from any of the physical files.
- Commitment control cannot be used with join logical files.

The following shows the DDS for Example 1:

```
JLF

A       R JOINREC          JFILE(PF1 PF2)
A       J                  JOIN(PF1 PF2)
A                          JFLD(NBR NBR)
A         NBR              JREF(PF1)
A         NAME
A         SALARY
A       K NBR


PF1

A       R REC1
A         NBR        10S
A         NAME       20A
A       K NBR


PF2

A       R REC2
A         NBR        10S
A         SALARY     7S 2
A       K NBR
```

Figure J.F.1. DDS Example for Joining Two Physical Files

The following describes the DDS for the join logical file in Example 1

The record level specification identifies the record format name used in the join logical file.

R Identifies the record format. Only one record format can be placed in a join logical file.

JFILE Replaces the PFILE keyword used in simple and multiple-format logical files. You must specify at least two physical files. The first file specified on the JFILE keyword is the primary file. The other files specified on the JFILE keyword are secondary files.

The join specification describes the way a pair of physical files is joined. The second file of the pair is always a secondary file, and there must be one join specification for each secondary file.

J Identifies the start of a join specification. You must specify at least one join specification in a join logical file. A join specification ends at the first field name specified in positions 19 through 28 or at the next J specified in position 17.

JOIN Identifies which two files are joined by the join specification. If only two physical files are joined by the join logical file, the JOIN keyword is optional.

JFLD Identifies the join fields that join records from the physical files specified on the JOIN. JFLD must be specified at least once for each join specification. The join fields are fields common to the physical files. The first join field is a field from the first file specified on the JOIN keyword, and the second join field is a field from the second file specified on the JOIN keyword.

Join fields, except character type fields, must have the same attributes (data type, length, and decimal positions). If the fields are character type fields, they do not need to have the same length. If you are joining physical file fields that do not have the same attributes, you can redefine them for use in a join logical file.

The field level specification identifies the fields included in the join logical file.

Field names Specifies which fields (in this example, Nbr, Name, and Salary) are used by the application program. At least one field name is required. You can specify any field names from the physical files used by the logical file. You can also use keywords like RENAME, CONCAT, or SST as you would in simple and multiple format logical files.

JREF In the record format (which follows the join specification level and precedes the key field level, if any), the field names must uniquely identify which physical file the field comes from. In this example, the Nbr field occurs in both PF1 and PF2. Therefore, the JREF keyword is required to identify the file from which the Nbr field description wil lbe used.

The key field level specification is optional, and includes the key field names for the join logical file.

K Identifies a key field specification. The K appears in position 17. Key field specifications are optional.

NOTE:

Key field names

Key field names (in this example, Nbr is the only key field) are optional and make the join logical file an indexed (keyed sequence) file.

Without key fields, the join logical file is an arrival sequence file. In join logical files, key fields must be fields from the primary file, and the key field name must be specified in positions 19 through 28 in the logical file record format.

The select/omit field level specification is optional, and includes Select/Omit field names for the join logical file.

S or O Identifies a select or omit specification. The S or O appears in position 17. Select/omit specifications are optional. Select/omit field names Only those records meeting the select/omit values will be returned to the program using the logical file. Select/omit fields must be specified in positions 19 through 28 in the logical file record format.

Reading a Join Logical File

The following cases describe how the join logical file in Figure 3-9 in topic 1.3.4.1 presents records to an application program.

The PF1 file is specified first on the JFILE keyword, and is therefore the primary file. When the application program requests a record, the system does the following:

1. Uses the value of the first join field in the primary file (the Nbr field in PF1).

2. Finds the first record in the secondary file with a matching join field (the Nbr field in PF2 matches the Nbr field in PF1).

3. For each match, joins the fields from the physical files into one record and provides this record to your program. Depending on how many records are in the physical files, one of the following conditions could occur:

A. For all records in the primary file, only one matching record is found in the secondary file. The resulting join logical file contains a single record for each record in the primary file as in (Case1).

B. For some records in the primary file, no matching record is found in the secondary file.

If you specify the JDFTVAL keyword:

- For those records in the primary file that have a matching record in the secondary file, the system joins to the secondary, or multiple secondaries. The result is one or more records for each record in the primary file.

- For those records in the primary file that do not have a matching record in the secondary file, the system adds the default value fields for the secondary file and continues the join operation. You can use the DFT keyword in the physical file to define which defaults are used. See "Record Missing in Secondary File; JDFTVAL Keyword Not Specified (Case 2A)"

Note: If the DFT keyword is specified in the secondary file, the value specified for the DFT keyword is used in the join. The result would be at least one join record for each primary record.

- If a record exists in the secondary file, but the primary file has no matching value, no record is returned to your program. A second join logical file can be used that reverses the order of primary and secondary files to determine if secondary file records exist with no matching primary file records.

If you do not specify the JDFTVAL keyword:

- If a matching record in a secondary file exists, the system joins to the secondary, or multiple secondaries. The result is one or more records for each record in the primary file.

- If a matching record in a secondary file does not exist, the system does not return a record.

Note: When the JDFTVAL is not specified, the system returns a record only if a match is found in every secondary file for a record in the primary file.

In the following examples, cases 1 through 4 describe sequential read operations, and case 5 describes reading by key.

Matching Records in Primary and Secondary Files (Case 1)

Assume that a join logical file is specified as in Figure L.F.1

PF1		PF2	
235	Anne	235	1700.00
440	Doug	440	950.50
500	Mark	500	2100.00
729	Sue	729	1400.90

The program does four read operations and gets the following records:

NBR	NAME	BASIC
235	Anne	1700.00
440	Doug	950.50
500	Mark	2100.00
729	Sue	1400.90

Record Missing in Secondary File; JDFTVAL Keyword Not Specified (Case 2A)

Assume that a join logical file is specified as in Figure J.F.2 and that there are four records in PF1 and three records in PF2, as follows:

PF1		PF2		
235	Anne	235	1700.00	No record was
440	Doug	440	950.50	found for number
500	Mark	729	1400.90	500 in PF2
729	Sue			

```
A       R JOINREC       JFILE(PF1 PF2)
A       J               JOIN(PF1 PF2)
A                       JFLD(NBR NBR)
A         NBR           JREF(PF1)
A         NAME
A         SALARY
A       K NBR
```

Figure J.F.2 DDS for Joining Two Physical Files

With the join logical file shown in Figure J.F.2, the program reads the join logical file and gets the following records:

RESULT FROM THE JLF

235	Anne	1700.00
440	Doug	950.50
729	Sue	1400.90

If you do not specify the **JDFTVAL** keyword and no match is found for the join field in the secondary file, the record is not included in the join logical file.

Record Missing in Secondary File; JDFTVAL Keyword Specified (Case 2B)

Assume that a join logical file is specified as in Figure L.F.2, except that the JDFTVAL keyword is specified, as shown in the following DDS:

JLF

```
A                    JDFTVAL
A     R JOINREC      JFILE(PF1 PF2)
A     J              JOIN(PF1 PF2)
A                    JFLD(NBR NBR)
A       NBR          JREF(PF1)
A       NAME
A       SALARY
A     K NBR
```

The program reads the join logical file and gets the following records:

RESULT FROM THE JLF

235	Anne	1700.00	A record for number 500
440	Doug	950.50	is returned if JDFTVAL is
500	Mark	0000.00	specified (but the SALARY
729	Sue	1400.90	is 0).

With JDFTVAL specified, the system returns a record for 500, even though the record is missing in PF2. Without that record, some field values can be missing in the join record (in this case, the Salary field is missing).

With JDFTVAL specified, missing character fields normally use blanks; missing numeric fields use zeros. However, if the DFT keyword is specified for the field in the physical file, the default value specified on the DFT keyword is used.

Secondary File Has More Than One Match for a Record in the PrimaryFile (Case 3)

Assume that a join logical file is specified as in Figure 3-9 in topic 1.3.4.1, and that four records in PF1 and five records in PF2, as follows:

PF1		PF2		
235	Anne	235	1700.00	Duplicate record
440	Doug	235	1500.00	was found in PF2
500	Mark	440	950.50	for number 235.
729	Sue	500	2100.00	
		729	1400.90	

The program gets five records:

JLF

			Order of records received
235	Anne	1700.00	for 235 is unpredictable
235	Anne	1500.00	unless you specify the
440	Doug	950.50	JDUPSEQ keyword.
500	Mark	2100.00	
729	Sue	1400.90	

Extra Record in Secondary File (Case 4)

Assume that a join logical file is specified as in Figure L.F.2, and that four records are contained in PF1 and five records in PF2, as follows:

PF1		PF2		
235	Anne	235	1700.00	Record for
440	Doug	301	1500.00	number 301 is
500	Mark	440	950.50	only in PF2.
729	Sue	500	2100.00	
		729	1400.90	

The program reads the join logical file and gets only four records, which would be the same even if JDFTVAL was specified (because a record must always be contained in the primary file to get a join record):

JLF

235	Anne	1700.00
440	Doug	950.50
500	Mark	2100.00
729	Sue	1400.90

Random Access (Case 5)

Assume that a join logical file is specified as in Figure L.F.1. Note that the join logical file has key fields defined.

This case shows which records would be returned for a random access read operation using the join logical file.

Assume that PF1 and PF2 have the following records:

PF1		PF2		
				No record was found for number 500 in PF2.
235	Anne	235	1700.00	
440	Doug	440	950.50	Record for number 984 is only in PF2.
500	Mark	729	1400.90	
729	Sue	984	878.25	
997	Tim	997	331.00	Duplicate records were found for number 997 in PF2.
		997	555.00	

The program can get the following records:

Given a value of 235 from the program for the Nbr field in the logical file, the system supplies the following record:

235 Anne 1700.00

Given a value of 500 from the program for the Nbr field in the logical file and with the JDFTVAL keyword specified, the system supplies the following record:

500 Mark 0.00

Note: If the JDFTVAL keyword was not specified in the join logical file, no record would be found for a value of 500 because no matching record is contained in the secondary file.

Given a value of 984 from the program for the Nbr field in the logical file, the system supplies no record and a no record found exception occurs because record 984 is not in the primary file.

Given a value of 997 from the program for the Nbr field in the logical file, the system returns one of the following records:

997 Tim 331.00

or

997 Tim 555.00

Which record is returned to the program cannot be predicted. To specify which record is returned, specify the JDUPSEQ keyword in the join logical file.

Notes:

1. *With random access, the application programmer must be aware that duplicate records could be contained in PF2, and ensure that the program does more than one read operation for records with duplicate keys. If the program were using sequential access, a second read operation would get the second record.*

2. *If you specify the JDUPSEQ keyword, the system can create a separate access path for the join logical file (because there is less of a chance the system will find an existing access path that it can share). If you omit the JDUPSEQ keyword, the system can share the access path of another file. (In this case, the system would share the access path of PF2.)*

Setting Up a Join Logical File

To set up a join logical file, do the following:

1. Find the field names of all the physical file fields you want in the logical file record format. (You can display the fields cortained in files using the Display File Field Description [DSPFFD] command.)
2. Describe the fields in the record format. Write the field names in a vertical list. This is the start of the record format for the join logical file.

Note: You can specify the field names in any order. If the same field names appear in different physical files, specify the name of the physical file on the JREF keyword for those fields. You can rename fields using the RENAME keyword, and concatenate fields from the same physical file using the CONCAT keyword. A subset of an existing character, hexadecimal, or zoned decimal field can be defined using the SST keyword. The substring of a character or zoned decimal field is a character field, and the substring of a hexadecimal field is also a hexadecimal field. You can redefine fields: changing their data type, length, or decimal positions.

3. Specify the names of the physical files as parameter values on the JFILE keyword. The first name you specify is the primary file. The others are all secondary files. For best performance, specify the secondary files with the least records first after the primary file.

4. For each secondary file, code a join specification. On each join specification, identify which pair of files are joined (using the JOIN keyword; optional if only one secondary file), and identify which fields are used to join the pair (using the JFLD keyword; at least one required in each join specification).

5. Optionally, specify the following:

A. The JDFTVAL keyword. Do this if you want to return a record for each record in the primary file even if no matching record exists in a secondary file

B. The JDUPSEQ keyword. Do this for fields that might have duplicate values in the secondary files. JDUPSEQ specifies on which field (other than one of the join fields) to sort these duplicates, and the sequence that should be used.

C. Key fields. Key fields cannot come from a secondary file. If you omit key fields, records are returned in arrival sequence as they appear in the primary file.

D. Select/omit fields. In some situations, you must also specify the dynamic selection (DYNSLT) keyword at the file level.

Using More Than One Field to Join Files (Example 2)

You can specify more than one join field to join a pair of files. The following shows the fields in the logical file and the two physical files.

JLF

Part Number

Color

Price

Quantity on Hand

PF1	PF2
PartNumber	PartNumber
Color	Color
Price	Quantity on Hand
Vendor	Warehouse

The DDS for these files is as follows:

JLF

```
A       R JOINREC  JFILE(PF1 PF2)
A       J          JOIN(PF1 PF2)
A                  JFLD(PTNBR PTNBR)
A                  JFLD(COLOR COLOR)
A         PTNBR    JREF(PF1)
A         COLOR    JREF(PF1)
A         PRICE
A         QUANTOH
```

PF1

```
A       R REC1
A       PTNBR      4S
```

```
A     COLOR     20A
A     PRICE           7S 2
A     VENDOR    40A

PF2.

A     R REC2
A     PTNBR     4S
A     COLOR     20A
A     QUANTOH   5S 0
A     WAREHSE   30A
```

Assume that the physical files have the following records:

PF1				PF2			
100	Black	22.50	ABC Corp.	100	Black	23	ABC Corp.
100	White	20.00	Ajax Inc.	100	White	15	Ajax Inc.
120	Yellow	3.75	ABC Corp.	120	Yellow	120	ABC Corp.
187	Green	110.95	ABC Corp.	187	Green	0	ABC Corp.
187	Red	110.50	ABC Corp.	187	Red	2	ABC Corp.
190	Blue	40.00	Ajax Inc.	190	White	2	Ajax Inc.

If the file is processed sequentially, the program receives the following records:

JLF

100	Black	22.50	23
100	White	20.00	15
120	Yellow	3.75	102
187	Green	110.95	0
187	Red	110.50	2

Note that no record for part number 190, color blue, is available to the program, because a match was not found on both fields in the secondary file. Because JDFTVAL was not specified, no record is returned.

Reading Duplicate Records in Secondary Files (Example 3)

Sometimes a join to a secondary file produces more than one record from the secondary file. When this occurs, specifying the JDUPSEQ keyword in the join specification for that secondary file tells the system to base the order of the duplicate records on the specified field in the secondary file.

The DDS for the physical files and for the join logical file are as follows:

```
JLF
A       R JREC      JFILE(PF1 PF2)
A       J           JOIN(PF1 PF2)
A                   JFLD(NAME1 NAME2)
A                   JDUPSEQ(TELEPHONE)
A         NAME1
A         ADDR
A         TELEPHONE
PF1
A       R REC1
A         NAME1     10A
A         ADDR      20A
PF2
A       R REC2
A         NAME2     10A
A         TELEPHONE     8S
```

Figure J.F.3. DDS Example Using the JDUPSEQ Keyword

The physical files have the following records:

PF1		PF2	
Anne	120 1st St.	Anne	555-1111
Doug	40 Pillsbury	Anne	555-6666
Mark	2 Lakeside Dr.	Anne	555-2222
		Doug	555-5555

The join logical file returns the following records:

JLF

Anne	120 1st St.	555-1111	Anne's telephone
Anne	120 1st St.	555-2222	numbers are in
Anne	120 1st St.	555-6666	ascebdubg order.
Doug	40 Pillsbury	555-5555	

The program reads all the records available for Anne, then Doug, then Mark. Anne has one address, but three telephone numbers. Therefore,1 there are three records returned for Anne.

The records for Anne sort in ascending sequence by telephone number because the JDUPSEQ keyword sorts in ascending sequence unless you specify *DESCEND as the keyword parameter.

The following example shows the use of *DESCEND in DDS:

```
JLF
A       R JREC      JFILE(PF1 PF2)
A       J           JOIN(PF1 PF2)
A                   JFLD(NAME1 NAME2)
A                   JDUPSEQ(TELEPHONE *DESCEND)
A         NAME1
A         ADDR
A         TELEPHONE
```

When you specify JDUPSEQ with *DESCEND, the records are returned as follows:

JLF			
Anne	120 1st St.	555-6666	Anne's telephone
Anne	120 1st St.	555-2222	numbers are in
Anne	120 1st St.	555-1111	descending order.
Doug	40 Pillsbury	555-5555	

Note: The JDUPSEQ keyword applies only to the join specification in which it is specified.

Using Join Fields Whose Attributes Are Different (Example 4)

Fields from physical files that you are using as join fields generally have the same attributes (length, data type, and decimal positions).

You can also use character type fields that have different lengths as join fields without requiring any redefinition of the fields.

For example, if the NAME1 Field of PF1 was 10 characters long and the NAME2 field of PF2 was 15 characters long, those fields could be used as join fields without redefining one of the fields.

The following is an example in which the join fields do not have the same attributes. The Nbr field in physical file PF1 and the Nbr field in physical file PF2 both have a length of 3 specified in position 34, but in the PF1 file the field is zoned (S in position 35), and in the PF2 file the field is packed (P in position 35). To join the two files using these fields as join fields, you must redefine one or both fields to have the same attributes.

The following illustrates the fields in the logical and physical files:

JLF

Employee Number

Name

Salary

PF1	**PF2**
Employee Number (zoned)	Employee Number (packed)
Name	Salary

The DDS for these files is as follows:

```
JLF
A       R JOINREC  JFILE(PF1 PF2)
A       J          JOIN(PF1 PF2)
A                  JFLD(NBR NBR)
A          NBR     S      JREF(2)
A          NAME
A          SALARY
PF1
A       R REC1
A          NBR                3S 0 <-Zoned
A          NAME       20A
A       K NBR
PF2
A       R REC2
A          NBR                3P 0 <-Packed
A          SALARY   7S 2
A       K NBR
```

Note: In this example, the Nbr field in the logical file comes from PF2, because JREF(2) is specified. Instead of specifying the physical file name, you can specify a relative file number on the JREF keyword; in this example, the 2 indicates PF2.

Because the Nbr fields in the PF1 and PF2 files are used as the join fields, they must have the same attributes. In this example, they do not. Therefore, you must redefine one or both of them to have the same attributes. In this example, to resolve the difference in the attributes of the two employee number fields, the Nbr field in JLF (which is coming from the PF2 file) is redefined as zoned (S in position 35 of JLF).

Describing Fields That Never Appear in the Record Format (Example 5)

A neither field (N specified in position 38) can be used in join logical files for neither input nor output. Programs using the join logical file cannot see or read neither fields. Neither fields are not included in the record format. Neither fields cannot be key fields or used in select/omit statements in the joined file. You can use a neither field for a join field (specified at the join specification level on the JFLD keyword) that is redefined at the record level only to allow the join, but is not needed or wanted in the programs

In the following example, the program reads the descriptions, prices, and quantity on hand of parts in stock. The part numbers themselves are not wanted except to bring together the records of the parts. However, because the part numbers have different attributes, at least one must be redefined.

JLF

Description

Price

Quantity on Hand

PF1	PF2
Description	Part Number
Part Number	Price
	Quantity on Hand

The DDS for these files is as follows:

```
JLF
A       R JOINREC          JFILE(PF1 PF2)
A       J                  JOIN(PF1 PF2)
A                          JFLD(PRTNBR PRTNBR)
A         PRTNBR   S N     JREF(1)
A         DESC
A         PRICE
A         QUANT
A       K DESC

PF1
A       R REC1
A         DESC     30A
A         PRTNBR   6P 0

PF2
A       R REC2
A         PRTNBR   6S 0
A         PRICE    7S 2
A         QUANT    8S 0
```

In PF1, the Prtnbr field is a packed decimal field; in PF2, the Prtnbr field is a zoned decimal field. In the join logical file, they are used as join fields, and the Prtnbr field from PF1 is redefined to be a zoned decimal field by specifying an S in position 35 at the field level. The JREF keyword identifies which physical file the field comes from.

However, the field is not included in the record format; therefore, N is specified in position 38 to make it a neither field. A program using this file would not see the field.

Application WhereIt Can Be Used

In this example, a sales clerk can type a description of a part. The program can read the join logical file for a match or a close match, and display one or more parts for the user to examine, including the description, price, and quantity. This application assumes that part numbers are not necessary to complete a customer order or to order more parts for the warehouse.

Specifying Key Fields in Join Logical Files (Example 6)

If you specify key fields in a join logical file, the following rules apply:

- The key fields must exist in the primary physical file.
- The key fields must be named in the join record format in the logical file in positions 19 through 28.
- The key fields cannot be fields defined as neither fields (N specified in position 38 for the field) in the logical file.

The following illustrates the rules for key fields:

```
JLF
A       R JOINREC          JFILE(PF1 PF2)
A       J                  JOIN(PF1 PF2)
A                          JFLD(NBR NUMBER)
A                          JFLD(FLD3 FLD31)
A         FLD1             RENAME(F1)
A         FLD2             JREF(2)
A         FLD3       35 N
A         NAME
A         TELEPHONE        CONCAT(AREA LOCAL)
A       K FLD1
A       K NAME
PF1
A       R REC1
A         NBR              4S
A         F1         20S
A         FLD2       7S 2
A         FLD3       40S
A         NAME       20A
PF2
A       R REC2
A         NUMBER 4S
A         FLD2       7S 2
A         FLD31      35A
A         AREA       3A
A         LOCAL      7A
```

The following fields cannot be key fields:

- Nbr (not named in positions 19 through 28)
- Number (not named in positions 19 through 28)
- F1 (not named in positions 19 through 28)
- Fld31 (comes from a secondary file)
- Fld2 (comes from a secondary file)
- Fld3 (is a neither field)
- Area and Local (not named in positions 19 through 28)
- Telephone (is based on fields from a secondary file)

Specifying Select/Omit Statements in Join Logical Files

If you specify select/omit statements in a join logical file, the following rules apply:

- The fields can come from any physical file the logical file uses (specified on the JFILE keyword).
- The fields you specify on the select/omit statements cannot be fields defined as neither fields (N specified in position 38 for the field).
- In some circumstances, you must specify the DYNSLT keyword when you specify select/omit statements in join logical files.

Joining Three or More Physical Files (Example 7)

- You can use a join logical file to join as many as 32 physical files. These files must be specified on the JFILE keyword. The first file specified on the JFILE keyword is the primary file; the other files are all secondary files.

- The physical files must be joined in pairs, with each pair described by a join specification. Each join specification must have one or more join fields identified.

The following shows the fields in the files and one field common to all the physical files in the logical file:

JLF

Name
Address
Telephone
Salary

PF1	**PF2**	**PF3**
Name	Name	Name
Address	Telephone	Salary

In this example, the Name field is common to all the physical files (PF1, PF2, and PF3), and serves as the join field.

The following shows the DDS for the physical and logical files:

```
JLF
A       R JOINREC          JFILE(PF1 PF2 P3)
A       J                  JOIN(PF1 PF2)
A                          JFLD(NAME NAME)
A       J                  JOIN(PF2 PF3)
A                          JFLD(NAME NAME)
A          NAME            JREF(PF1)
A          ADDR1
A          TELEPHONE
A          SALARY
A       K NAME
PF1
A       R REC1
A          NAME      10
A          ADDR      20
A       K NAME
PF2
A       R REC2
A          NAME            10
A          TELEPHONE       7
A       K NAME
PF3
A       R REC3
A          NAME      10
A          SALARY    9 2
A       K NAME
```

Assume the physical files have the following records:

PF1		PF2		PF3	
Anne	120 1st St.	Anne	555-1111	Anne	1700.00
Doug	40 Pillsbury	Doug	555-5555	Doug	950.00
Mark	2 Lakeside Dr.	Mark	555-0000	Mark	2100.00
Tom	335 Elm St.	Sue	555-3210		

The program reads the following logical file records:

JLF			
Anne	120 1st St.	555-1111	1700.00
Doug	40 Pillsbury	555-5555	950.00
Mark	2 Lakeside Dr.	555-0000	2100.00 [3]

No record is returned for Tom because a record is not found for him in PF2 and PF3 and the JDFTVAL keyword is not specified. No record is returned for Sue because the primary file has no record for Sue.

Joining a Physical File to Itself [SELF JOIN] (Example 8)

You can join a physical file to itself to read records that are formed by combining two or more records from the physical file itself. The following example shows how:

JLF

Employee Number

Name

Manager's Name

PF1

Employee Number

Name

Manager's Employee Number

The following shows the DDS for these files:

```
JLF
A                          JDFTVAL
A       R JOINREC          JFILE(PF1 PF1)
A       J                  JOIN(1 2)
A                          JFLD(MGRNBR NBR)
A          NBR             JREF(1)
A          NAME            JREF(1)
A          MGRNAME         RENAME(NAME)
A                          JREF(2)
PF1
A       R RCD1
A          NBR       3S
A          NAME      10A   DFT('none')
A          MGRNBR 3S
```

Notes:

1. *Relative file numbers must be specified on the JOIN keyword because the same file name is specified twice on the JFILE keyword. Relative file number 1 refers to the first physical file specified on the JFILE keyword, 2 refers to the second, and so forth.*
2. *With the same physical files specified on the JFILE keyword, the JREF keyword is required for each field specified at the field level.*

Assume the following records are contained in PF1:

PF1

235	Anne	440
440	Doug	729
500	Mark	440
729	Sue	888

The program reads the following logical file records:

PF1

235	Anne	Doug
440	Doug	Sue
500	Mark	Doug
729	Sue	none

Note that a record is returned for the manager name of Sue because the JDFTVAL keyword was specified. Also note that the value none is returned because the DFT keyword was used on the Name field in the PF1 physical file.

Using Default Data for Missing Records from Secondary Files (Example 9)

If you are joining more than two files, and you specify the JDFTVAL keyword, the default value supplied by the system for a join field missing from a secondary file is used to join to other secondary files. If the DFT keyword is specified in the secondary file, the value specified for the DFT keyword is used in the logical file.

The DDS for the files is as follows:

```
JLF
A                          JDFTVAL
A       R JRCD             JFILE(PF1 PF2 PF3)
A       J                  JOIN(PF1 PF2)
A                          JFLD(NAME NAME)
A       J                  JOIN(PF2 PF3)
A                          JFLD(TELEPHONE TELEPHONE)
A         NAME             JREF(PF1)
A         ADDR
A         TELEPHONE        JREF(PF2)
A         LOC

PF1
A       R RCD1
A         NAME     20
A         ADDR     40
A         COUNTRY        40
PF2

A       R RCD2
A         NAME           20
A         TELEPHONE      8        DFT('999-9999')
PF3

A       R RCD3
A         TELEPHONE      8
A         LOC                     30     DFT('No location assigned')
```

Assume that PF1, PF2, and PF3 have the following records:

PF1

Anne	120 1st St.
Doug	40 Pillsbury
Mark	2 Lakeside Dr.
Sue	120 Broadway

PF2

Anne	555-1234	No telephone number
Doug	555-2222	found for Mark
Sue	555-1144	in PF2.

PF3

555-1234	Room 312	No record for telephone
555-2222	Main Lobby	number 555-1144 was
999-9999	No telephone number	found in PF3.

With JDFTVAL specified in the join logical file, the program reads the following logical file records:

Anne	120 1st St	555-1234	Room 312
Doug	40 Pillsbury	555-2222	Main Lobby
Mark	2 Lakeside Dr	999-9999	No telephone number
Sue	120 Broadway	555-1144	No location assigned [3]

In this example, complete data is found for Anne and Doug. However, part of the data is missing for Mark and Sue

- PF2 is missing a record for Mark because he has no telephone number. The default value for the Telephone field in PF2 is defined as 999-9999 using the DFT keyword. In this example, therefore, 999-9999 is the telephone number returned when no telephone number is assigned. The JDFTVAL keyword specified in the join logical file

causes the default value for the Telephone field (which is 999-9999) in PF2 to be used to match with a record in PF3. (In PF3, a record is included to show a description for telephone number 999-9999.) Without the JDFTVAL keyword, no record would be returned for Mark.

- Sue's telephone number is not yet assigned a location; therefore, a record for 555-1144 is missing in PF3. Without JDFTVAL specified, no record would be returned for Sue. With JDFTVAL specified, the system supplies the default value specified on the DFT keyword in PF3 the Loc field (which is No location assigned).

A Complex Join Logical File (Example 10)

The following example shows a more complex join logical file. Assume the data is in the following three physical files:

VENDOR MASTER FILE (PF1)

```
A       R RCD1              TEXT('VENDOR INFORMATION')
A         VDRNBR  5S        TEXT('VENDOR NUMBER')
A         VDRNAM 25A        TEXT('VENDOR NAME')
A         STREET  15A       TEXT('STREET ADDRESS')
A         CITY    15A       TEXT('CITY')
A         STATE   2A        TEXT('STATE')
A         ZIPCODE 5S        TEXT('ZIP CODE')
A                           DFT('00000')
A         PAY               1A    TEXT('PAY TERMS')
```

ORDER FILE (PF2)

```
A       R RCD2              TEXT('VENDORS ORDER')
A         VDRNUM 5S 0       TEXT('VENDOR NUMBER')
A         JOBNBR  6         TEXT('JOB NUMBER')
```

```
A        PRTNBR  5S 0        TEXT('PART NUMBER')
A                            DFT(99999)
A        QORDER  3S 0        TEXT('QUANTITY ORDERED')
A        UNTPRC  6S 2        TEXT('PRICE')
```

PART FILE (PF3)

```
A      R RCD3                TEXT('DESCRIPTION OF PARTS')
A        PRTNBR        5S 0  TEXT('PART NUMBER')
A                            DFT(99999)
A        DESCR         25A   TEXT('DESCRIPTION')
A        UNITPRICE     6S 2  TEXT('UNIT PRICE')
A        WHSNBR        3S    TEXT('WAREHOUSEA*NUMBER')
A        PRTLOC        4A    TEXT('LOCATION OF PART')
A        QOHAND        5S    TEXT('QUANTITY ON HAND')
```

The join logical file record format should contain the following fields:

Vdrnam (vendor name)

Street, City, State, and Zipcode (vendor address)

Jobnbr (job number)

Prtnbr (part number)

Descr (description of part)

Qorder (quantity ordered)

Untprc (unit price)

Whsnbr (warehouse number)

Prtloc (location of part)

The DDS for this join logical file is as follows:

JOIN LOGICAL FILE (JLF)

```
A          1          DYNSLT
A          2          JDFTVAL
A        R RECORD1    JFILE(PF1 PF2 PF3)
A    3   J            JOIN(1 2)
A                     JFLD(VDRNBR VDRNUM)
A    4                JDUPSEQ(JOBNBR)
A    5   J            JOIN(2 3)
A    6                JFLD(PRTNBR PRTNBR)
A                     JFLD(UNTPRC UNITPRICE)
A    7     VDRNUM 5A  N TEXT('CHANGED ZONED TO CHAR')
A          VDRNAM
A          ADDRESS 8  CONCAT(STREET CITY STATE +
A                     ZIPCODE)
A          JOBNBR
A          PRTNBR  9  JREF(2)
A          DESCR
A          QORDER
A          UNTPRC
A          WHSNBR
A          PRTLOC
A   10   S VDRNAM     COMP(EQ 'SEWING COMPANY')
A        S QORDER     COMP(GT 5)
```

1. The DYNSLT keyword is required because the JDFTVAL keyword and select fields are specified.

2. The JDFTVAL keyword is specified to pick up default values in physical files.

3. First join specification.

4. The JDUPSEQ keyword is specified because duplicate vendor numbers occur in PF2.

5. Second join specification.

6. Two JFLD keywords are specified to ensure the correct records are joined from the PF2 and PF3 files.

7. The Vdrnum field is redefined from zoned decimal to character (because it is used as a join field and it does not have the same attributes in PF1 and PF2).

8. The CONCAT keyword concatenates four fields from the same physical file into one field.

9. The JREF keyword must be specified because the Prtnbr field exists in two physical files and you want to use the one in PF2.

10. The select/omit fields are Vdrnam and Qorder. (Note that they come from two different physical files.)

(*) A logical file with more than one record format
The attributes of the key fields must be same
field 1 must be equal (u attribute must be same)
field 2 must be equal (u attribute must be same).

SUMMARY OF RULES FOR JOIN LOGICAL FILES

Requirements

The principal requirements for join logical files are:

- Each join logical file must have:

  * Only one record format, with the JFILE keyword specified for it.
  * At least two physical file names specified on the JFILE keyword. (The physical file names on the JFILE keyword do not have to be different files.)
  * At least one join specification (J in position 17 with the JFLD keyword specified).
  * A maximum of 31 secondary files.
  * At least one field name with field use other than N (neither) at the field level.

- If only two physical files are specified for the JFILE keyword, the JOIN keyword is not required. Only one join specification can be included, and it joins the two physical files.

- If more than two physical files are specified for the JFILE keyword, the following rules apply:

  * The primary file must be the first file of the pair of files specified on the first JOIN keyword (the primary file can also be the first of the pair of files specified on other JOIN keywords).

Note: Relative file numbers must be specified on the JOIN keyword and any JREF keyword when the same file name is specified twice on the JFILE keyword.

* Every secondary file must be specified only once as the second file of the pair of files on the JOIN keyword. This means that for every secondary file on the JFILE keyword, one join specification must be included (two secondary files would mean two join specifications, three secondary files would mean three join specifications).

* The order in which secondary files appear in join specifications must match the order in which they are specified on the JFILE keyword.

Join Fields

The rules to remember about join fields are:

- Every physical file you are joining must be joined to another physical file by at least one join field. A join field is a field specified as a parameter value on the JFLD keyword in a join specification.

- Join fields (specified on the JFLD keyword) must have identical attributes (length, data type, and decimal positions) or be redefined in the record format of the join logical file to have the same attributes. If the join fields are of character type, the field lengths may be different.
- Join fields need not be specified in the record format of the join logical file (unless you must define one or both so that their attributes are identical).

- If you redefine a join field, you can specify N in position 38 (making it a neither field) to prevent a program using the join logical file from using the redefined field.

- The maximum length of fields used in joining physical files is equal to the maximum size of keys for physical and logical files

Fields in Join Logical Files

The rules to remember about fields in join logical files are:

- Fields in a record format for a join logical file must exist in one of the physical files used by the logical file or, if CONCAT, RENAME, TRNTBL, or SST is specified for the field, be a result of fields in one of the physical files.

- Fields specified as parameter values on the CONCAT keyword must be from the same physical file. If the first field name specified on the CONCAT keyword is not unique among the physical files, you must specify the JREF keyword for that field to identify which file contains the field descriptions you want to use.

- If a field name in the record format for a join logical file is specified in more than one of the physical files, you must uniquely specify on the JREF keyword which file the field comes from.

- Key fields, if specified, must come from the primary file. Key fields in the join logical file need not be key fields in the primary file.

- Select/omit fields can come from any physical file used by the join logical file, but in some circumstances the DYNSLT keyword is required.

- If specified, key fields and select/omit fields must be defined in the record format.

- Relative file numbers must be used for the JOIN and JREF keywords if the name of the physical file is specified more than once on the JFILE keyword.

MANAGING DATABASE FILES

INTRODUCTION

Before you perform any input or output operations on a file, the files must have ablest one member. As a general rule, database files have only one member, the one created when the file is created. The name of this member is the same as the file name, unless you give it a different name. Because most operations on database files assume that the member being used as the first member in the file, and because most files only have one member, you do not normally have to be concerned with, or specify, member names. If a file contains more than one member, each file serves as the subset of the data in the file. This allows you to classify the data easier.

Member Operations Common to all Database files

The system supplies a way for you to:

- Add new members to an existing file
- Change some attributes for an existing member
- Rename a member

Adding Members to files

You can add members to files in any of these ways:

- Automatically. When a file is created using the create physical file(CRTPF) or create Logical file (CRTLF) commands, the default is to automatically add a member to the newly created file. You can specify a different member name using the MBR parameter on the create database commands. if you do not want a member added when the file is created, specify *NONE on the MBR parameter.
- Specifically. after the file is created, you can add a member using the add physical file member (ADDPFM) or Add logical file member (ADDLFM) commands.
- Copy files (CPYF) command. If the member you are copying does not exist in the file being copied to, the member is added to the file by CPYF command.

Changing Member Attributes

You can use the change physical file ~~manager~~ member (CHGPFM) or change logical files member(CHGLFM) command to change certain attributes of a physical or a logical file member.

1. For a physical file member, you can change the following parameters: SRCTYPE, EXPDATE, SHARE and TEXT.
2. For a logical file member you can change the SHARE and TEXT parameters.

Note: You can use the change physical file and charge logical file (CHGLF) commands to change many other file attributes. Example, to change the maximum size of allowed for each member in the file, you would use the SIZE parameter on the CHGPF command.

Renaming Members

The rename member (RNMM) command changes the name of an existing a physical or logical file. The file name is not changed.

Removing Members from files

1. The Remove member (RMVM) command is used to remove the member and its contents. Both the member data and the member itself are removed.
2. After the member the removed, it can no longer be used by the system. This is different from clearing or deleting the data from the member.

Initializing Data in a physical file Member

To use relative record processing in a program, the database file must contain a number of record positions equal to the highest relative record number used in the program. Programs using relative-record-number processing sometimes requires that these records be initialized.

You can use the initialize physical file member (INZPFM) command to initialize members with one of two types of records:

- Default records
- Deleted records

  You specify which type of record you want using the RECORDS parameter on the initialize physical file member (INZPFM) command.

  If you initialize records using default records , the fields in each new record are initialized to the default field values defined when the file was created.
- If no default file value was, numeric are filled with zeros.
- Character fields are filled with blanks.
- variable -length character fields have a zero-length default value.
- The default value for null-capable fields is the null value.
- The default value for dates, times, and time stamps is the current date, time, or time stamp.
- Program described files have a default value of all blanks.

Note: *you can initialize one default record if the UNIQUE keyword is specified in DDS for the physical file member or any associated logical file members. Otherwise, you would create a series of duplicate key records.*

If the records are initialized to the default records, you can read a record by a relative record numberand change the data.

If the records were initialized to deleted records, you can change the data by adding a record using the relative record number of one of the deleted records.

Deleted records cannot be read; they only hold a place in the member. A deleted record can be changed by writing a new record over the deleted records.

Clearing data from physical Members

The clear physical file member (CLRPFM) command is used to remove the data from a physical file member. After a clear operation is complete, member description remains, but the data is gone.

Reorganizing the data in physical file members

You can use the reorganize physical file member(RGZPFM) command to:

- Remove deleted records to make the space occupied by them available for more records.
- Reorganize the records of a file in the order in which you normally access them sequentially, there by minimizing the time required to retrieve records. This is done using the KEYFILE parameter. This may be advantages for files that are primarily accessed in on
- order other than arrival sequence.
- A member can be reorganized using either of the following:
    - Key fields of the physical file
    - Key fields of a logical file based on the physical file
- Reorganize a source file member, insert new source sequence numbers, and reset the source data fields (using the srcopt and srcseq parameters on the reorganize physical file members).
- Reclaim space in the variable portion of the file that was previously used by variable-length fields in the physical file format and has that now become fragmented.

Displaying records in a physical file member

The Display physical file member (DSPPFM) command can be used to display the data in the physical file database file members by arrival sequence. The commands can be used for:

- Problem analysis
- Debugging

- Record inquiry

You can display source files or data files, regardless if they are keyed or arrival sequence. Records are displayed in arrival sequence, even if the file is a keyed file. You can page through the file, locate a particular record by record number, or shift the display to the right or left to see other parts of the record. You can also press a function key to show either character data or hexadecimal data on the display.

If you have query installed, you can use the start query (STRQRY) command to select and display records, too.

If you have SQL language installed, you can use the start SQL (STRSQL) command to interactively select and display records.

CHANGING DATABASE FILE DESCRIPTIONS AND ATTRIBUTES

Effect of changing fields in a file Description

When a program that uses externally described data is compiled, the compiler copies the file descriptions of the files into the compiled program. When you run a program, the system can verify that the record formats the program was compiled with are the same as the record formats currently defined for the file. The default is to do level checking. The systems assigns a unique level identifier for each record format when the file it is associated with is created. The system uses the information in the record format description to determine the level identifier. This information includes the total length of the record format, the record name, the number and order of fields defined, the data type, the size of the field, the field names, and the decimal positions on in the field. Changes to this information in a record format cause the level identifier to change.

The following DDS information has no effect on the level identifier and therefore, can be changed without recompiling the program that uses the files:

- TEXT keyword
- COLHDG keyword
- CHECK keyword
- EDTDCE keyword
- EDTWRD keyword
- REF keyword
- REFFLD keyword
- CMP, RANGE and VALUES keyword
- REFSHIFT keyword
- DFT keyword
- CCSID keyword
- ALWNULL keyword
- Join specifications and join keyword

- key fields
- Access path keyword
- Select/omit keyword

Keep in mind that even though changing the key or Select/omit fields will not cause a level check, the change may cause unexpected results in program using the new access path.

For example, changing the key field from the customer number to customer name changes the order in which the records are retrieved, and may cause unexpected problems in the programs processing the file.

If level checking is specified, the level identifier of the file to be used is compared to the level identifier of the file in your program when the file is opened. If the identifiers differ, a message is sent to the program to identified the changed condition and the changes may affect your programs. You can simply compile your program again so that the changes are included.

Changing a physical file description and attributes

Sometimes, when you make a change to a physical file description and then recreate the file, the level identifier can change. For example, the level identifier will change if you add a field to the file description, or change the rest of the existing field. If level identifier changes, you can compile the program again that use the physical file. After the programs are recompiled, they will use the new level check identifier.

You can avoid compiling again by creating a logical file that presents a data to your programs in the original record format of the physical file. Using this approach, the logical file has the same level check identifier as the physical file before the change.

For example, you decide to add a field to a physical file record format. You can avoid compiling your program again by doing the following:

1. Change the DDS and create a new physical file (FILEB in LIBA) to include the new field.

   CRTPF FILE(LIBA/FILEB) MBR(*NONE)

   FILEB does not have a member. (The old file FILEA is in library LIBA and has one member MBRA.)

2. Copy the member of the old physical file to the new physical file:

   CPYF FROMFILE(LIBA/FILEA) TOFILE(LIBA/FILEB)
   FROMMBR(*ALL) TOMBR(*FROMMBR)
   MBROPT(*ADD) FMTOPT(*MAP)

The member in the new physical file is automatically named the same as the member in the old physical file because FROMMBR(*ALL) and TOMBR(*FROMMBR) are specified. The FMTOPT parameter specifies to copy (*MAP) the data in the fields by field name.

3. Describe a new logical file (FILEC) that looks like the original physical file (the logical file record format does not include the new physical file field). Specify FILEB for the PFILE keyword. (When a level check is done, the level identifier in the logical file and the level identifier in the program match because FILEA and FILEC have the same format.)

4. Create the new logical file:

   CRTLF FILE(LIBA/FILEC)...

5. You can now do one of the following:

   a. Use an Override with Database File (OVRDBF) command in the appropriate jobs to override the old physical file referred to in the program with the logical file.

      OVRDBF FILE(FILEA) TOFILE(LIBA/FILEC)

   b. Delete the old physical file and rename the logical file to the name of the old physical file so the file name in the program does not have to be overridden.

DLTF FILE(LIBA/FILEA)

RNMOBJ OBJ(LIBA/FILEC) OBJTYPE(*FILE)

NEWOBJ(FILEA)

The following illustrates the relationship of the record formats used in the three files:

FILEA (old physical file)

FLDA FLDB FLDC FLDD

FILEB (new physical file)

FLDB1

FLDB1 was added to the record format.

FILEC (logical file)

FLDA FLDB FLDC FLDD

FILEC shares the record format of FILEA.

FLDB1 is not used in the record format for the logical file.

When you make changes to a physical file that cause you to create the file again, all logical files referring to it must first be deleted before you can delete and create the new physical file. After the physical file is re-created, you can re-create or restore the logical files referring to it. The following examples show two ways to do this:

Changing a logical file Description and attributes

As a general rule, when you make changes to the logical file that will cause a change to the level identifier, it is strongly recommended that you recompile the program that use the logical file. Sometimes you can make changes to a file that change the level identifier and which do not require you to recompile your program. However, in those situations you will be forced to run off level checking to run your program using the changed file. That is not the preferred method of operating. It increases the chances of incorrect data in the future.

To avoid recompiling, you can keep the current logical file and create a new logical file with the added field. Your programs refer to the old file, which still exists.

USING DATABASE ATTRIBUTES AND CROSS - REFERENCE

The AS/400 integrated database provides file attributes and cross - reference information. Some of the cross - reference information includes:

- The files used in a program
- The files that depend on other files for data or access paths
- File attributes
- The fields defined for a file
- Constraints associated with a file
- Key fields for a file

Displaying attributes for a file

You can use the display file description (DSPFD) command to display the file attributes for database files and device files. The information can be displayed, printed, or written to a database output file. The information supplied by this command includes:

- Basic attributes
- Basic attributes (*BASATR)
- Access path specifications (*ATR)
- Select/omit specifications
- Join logical file specifications
- Alternative collating sequence specifications
- Record format specifications
- Member attributes
- Spooling attributes
- Member lists
- File constraints (*CST)
- Triggers (*TRG)

Displaying the Relationships between files on the System

You can use the display database relations (DSPDBR) command to display the following information about the organization of your database:

- A list of database files(physical and logical) that use a specific record format.
- A list of database files(physical and logical) that depend on the specified file for data sharing.
- A list of members(physical and logical) that that depend on the specified Member for sharing data or sharing on access path.
- A list of physical files that are dependent files in a referential constraint relationship with a file.

This information can be displayed, printed or written to the database output file(OUTFILE).

Displaying the files used by programs

You can use the display program Reference (DSPPGMREF) command to determine which files, data areas, and other programs are used by a program. This information available for compiled programs only. The information cab be displayed, printed, or written to the database output file(OUTFILE).

Displaying the system cross-reference files

The system manages eight database files that contain:

- Basic database attribute information (QSYS/QADBXREF)
- Cross-reference information (QSYS/QADBFDEP) about all the database files on the system
- Database files field information (QSYS/QADBIFLD)
- Database file KEY field information (QSYS/QADBKFLD)
- Referential constraint file information (QSYS/QADBFCST)
- Referential constraint field information (QSYS/QADBCCST)
- SQL Package information(QSYS/QADBPKG)

- Remote database directory information (QSYS/QADBXRDBD)

You can use these files to determine basic attributes and database file requirements. To display the fields contained in these files, use the display file field description (DSPFFD) command.

PHYSICAL FILE CONSTRAINTS

The DB2/400 database supports the following physical file constraints:

- Referential constraints
- Primary key constraints
- Unique constraints

Primary key and unique constraints are useful alone and are often necessary when using referential constraints. Unique constraints let you create additional, enforced unique keys for a file beyond the file access path. Unique and primary key constraints are enforced during file insertions, updates, and deletions. Primary key and unique constraints are similar to logical files in that they provide a keyed access path.

Unique Constraint

A unique constraint identifies a field or set of fields in a database file that meets all the following:

- Is unique within the file
- Is in ascending order
- Can be null capable

A file can have multiple unique constraints, but there cannot be duplicate unique constraints. The same key fields, regardless of order, constitute a duplicate constraint.

Primary Key Constraint

A primary key constraint identifies a field or set of fields in a database file that meets all the following:

- Is unique within the file
- Is in ascending order
- cannot be null capable

Only one primary key constraint can be defined for a file. A primary key constraint is a unique key constraint with special attributes.

Adding Unique and Primary Key Constraints

The Add Physical File Constraint (ADDPFCST) command adds all types of physical file constraints. To add unique and primary key constraints, specify *UNQCST for the Type parameter for a unique constraint and *PRIKEY for a primary key constraint. When adding a primary key constraint, the specified key becomes the file's primary access path. If the file does not have a keyed access path that can be shared, one is created by the system.

The KEY parameter is required when adding unique and primary key constraints.

Removing Constraints

The Remove Physical File Constraint (RMVPFCST) command removes a constraint.
The full impact of the RMVPFCST command depends on what constraint you request be removed and certain conditions surrounding the constraint.

The TYPE parameter specifies the type of constraint to remove:

- Unique constraints TYPE(*UNQCST)
    - * All unique constraints except the primary key constraint are removed when CST(*ALL) is specified
    - * The named unique constraint for CST(constraint-name)

- Primary key constraints TYPE(*PRIKEY)
    - * The primary constraint is removed when CST(*ALL) is specified
    - * The named primary constraint for CST(constraint-name)

If you remove a primary key constraint or unique constraint and the associated access path is shared by a logical file, the ownership of the shared path is transferred to the logical file. If the access path is not shared, it is removed.

When you remove a primary key constraint with the RMVPFCST command, the system sends an inquiry message asking if the key specifications should be removed from the file. A reply of 'K' indicates keep the key specifications in the file. The file remains keyed. A reply of 'G' indicates the file will have an arrival sequence access path when the command completes.

When you remove a primary key constraint with the SQL ALTER TABLE, the inquiry message is not sent. The key specifications are always removed and the file will have an arrival sequence access path when the ALTER TABLE completes.

Working With Physical File Constraints

The Work with Physical File Constraints (WRKPFCST) command displays a list of constraints that includes the constraint name, file and library name, type, state, and whether it is in check pending. From this display you can change or remove a constraint. You can also display a list of the records that have caused a file to be put into check pending.

```
                    Work With Physical Fle constraints

Type options, press Enter.

2=Change     4=Remove     6=Display records in check pending


Opt  Constraint   File       Library    Type       State   Check
                                                           Pending
_    DEPTCST      EMPDIV7    EPPROD     *REFCST    EST/ENB   No
_    ACCTCST      EMPDIV7    EPPROD     *REFCST    EST/ENB   Yes
_    STAT84       EMPDIV7    EPPROD     *REFCST    DEF/ENB   No
_    FENSTER      REVSCHED   EPPROD     *REFCST    EST/DSB   No
_    IRSSTAT3     REVSCHED   EPPROD     *UNQCST
_    IFRNUMBEROREVSCHED EPPROD         *UNQCST
_    EVALDATE     QUOTOSCHEMEPPROD      *REFCST    EST/ENB   No
_    STKOPT       CANSCRONN9 EPPROD     *PRIKEY


Parameters for options 2, 4, 6 or command
===>
_____________________________________________________________________

F3=Exit      F4=Prompt    F5=Refresh   F12=Cancel   F15=Sort by

F16=Repeat position to    F17=Position to    F22=Display constraint name
```

Figure P.F.C.1. Work with Physical File Constraints Display

The display shows all the constraints defined for file specified in the WRKPFCST command. The display lists the constraint names, the file name, and the library name. The Type column identifies the constraint as referential, unique, or primary key. The State column indicates whether the constraint is defined or established and whether it is enabled or disabled. The final column contains the check pending status of the constraint. Unique

and primary key constraints do not have a state because they are always established and enabled.

With this display, you have the following options:

- Change (option 2) to any permissible state. For example, you can enable a constraint that is currently disabled. This option performs the same functions as the CHGPFCST command.

- Remove (option 4) a constraint. This option performs the same functions as the RMVPFCST command.

- Display (option 6) the records that are in check pending. This option performs the same functions as DSPCPCST command.

Displaying Check Pending Constraints

The Display Check Pending Constraints (DSPCPCST) command displays or prints the records that caused the constraint to be marked as check pending. Before using this command, use the Change Physical File Constraint (CHGPFCST) command to disable the constraint. This command may take a long time when the associated files have a large number of records. Figure P.F.C.2 shows the DSPCPCST command displays the records in check pending.

```
                                Display report
Width . . . . . 71
Column . . . . . 1
Control . . . . . _____
Line  ...+...1...+...2...+...3...+...4...+...5...+....
        PARTNUM   PARTID      QUANTITY
        ------    ------      --------
00001   25RFA1    WIDGET      3000
00002   32XGW3    GIZMO       32
  3 ***** *****E N D O F D A T A*****                          3

F3=Exit      F12=Cancel   F19=Left    F20=Right   F21=Split
```

Figure P.F.C.2. DSPCPCST Display

Processing Check Pending Constraints

The Edit Check Pending Constraints (EDTCPCST) command presents the Editing Check Pending Constraints display shown in Figure P.F.C.3

This display can help you manage and schedule the verification of physical file constraints placed in check pending.

When verifying a physical file constraint, the database must ensure that every record in the file meets the constraint definition. For instance, verification of a referential constraint causes the database to check that every foreign key value has a matching parent key value.

```
                                Edit Check Pending

Type sequence, press Enter.

Sequence:   1-99, *HLD

                    ----------Constraints------------------Verify        Elapsed

Seq    Status          Cst    File          Library       Time          Time

1      RUN             EMP    DEP           EPPROD        00:01:00      00:00:50

1      READY           CONSTDEP             EPPROD        00:02:00      00:00:00

*HLD   CHKPND          FORT   STYBAK        EPPROD        00:03:00      00:00:00

*HLD   CHKPND          CST88  STYBAK        EPPROD        00:10:00      00:00:00

*HLD   CHKPND          CS317  STYBAK        EPPROD        00:20:00      00:00:00

*HLD   CHKPND          KSTANSTYBAK          EPPROD        02:30:00      00:00:00

                                Bottom

       F3=Exit         F5=Refresh     F12=Cancel     F13=Repeat all        F15=Sort by

       F16=Repeat position to         F17=Position to       F22=Display constraint name
```

Figure P.F.C.3. Edit Check Pending Constraints Display

The status field has one following values:

- RUN indicates the constraint is being verified.
- READY indicates the constraint is ready to be verified.
- INVAP indicates the access path associated with the constraint is invalid. Once the access path has been rebuilt, the constraint will be automatically verified by the system.
- HELD indicates the constraint is not be verified. You must change the sequence to a value from 1 to 99 to change this state.

- CHKPND indicates there has been a attempt to verify the constraint, but the constraint is still in check pending. You must change the sequence to a value from 1 to 99 to change this state.

The Constraint column contains the first five characters of the constraint name. If the name is followed by a > if it exceeds five characters. You can display the whole long name by putting the cursor on that line and press the F22 key.

The verify time column shows the time it would take to verify the constraint if there were no other jobs on the system. The elapsed time column indicates the time already spent on verifying the constraint.

Physical File Constraint Considerations and Limitations

- A file must be a physical file.
- A file can have a maximum of one member, MAXMBR(1).
- A constraint can be defined when the file has zero members. A constraint cannot be established until the file has one, and only one, member.
- A file can have a maximum of one primary key but may have many parent keys.
- There is maximum of 300 constraint relations per file. This maximum value is the sum of:
    - The unique constraints.
    - The primary key constraints.
    - The referential constraints whether participating as a parent or a dependent, and whether the constraints are defined or established.

- Constraint names must be unique in a library.
- Constraints cannot be added to files in the QTEMP library.

REFERENTIAL INTEGRITY

Introducing Referential Integrity and Referential Constraints

Referential integrity is a broad term encompassing all the mechanisms and techniques that ensure the validity of referenced data in a relational database. Referential constraints are a mechanism to enforce a level of validity desired in your database.

Database users want referential integrity implemented in their database management system for several reasons:

- To ensure data values between files are kept in a state that meets the rules of their business. For example, if a business has a list of customers in one file and a list of their accounts in another file, it does not make sense to allow the addition of an account if an associated customer does not exist. Likewise, it is not reasonable to delete a customer until all their accounts are deleted.
- To be able to define the relationships between data values.
- To have the system enforce the data relationships no matter what application makes changes.
- To improve the performance of integrity checks done in a HLL or SQL level by moving the checking into the database.

Referential Integrity Terminology

A discussion of referential integrity requires an understanding of several terms. These terms are in an order that may help you understand their relationship to each other.

Primary Key. A field or set of fields in a database file that must be unique, ascending, and cannot contain null values. The primary key is the primary file access path. The primary key can become the parent key in the parent file.

Unique Key. A field or set of fields in a database file that must be unique, ascending, and can contain null values.

Parent Key. A field or set of fields in a database file that must be unique, ascending, and may or may not contain null values. The parent key can be the same as the primary key or unique key. It can also be a superset of the primary key.

Foreign Key. A field or set of fields in which each non-null value must match a key value of the related parent file.

However, the attributes (data type, length, and so forth) must be the same as the primary or parent key.

Referential integrity. The state of a database in which the values of the foreign keys are valid. The enforcement of referential integrity prevents the violation of the "non-null foreign key must have a matching parent key" rule.

Referential constraint. The constraint defines a relationship between the records identified by parent keys and the records identified by foreign keys. Because a dependent file is always dependent upon the parent file, the referential constraint is defined from the dependent file's perspective. The constraint cannot be defined from the parent file's perspective.

Parent file. The file in a relationship containing the parent key or primary key.

Dependent file. The file in a relationship containing the foreign key.

Check pending. The state that occurs when the database does not know with certainty that a particular dependent file contains only valid data relative to its associated parent key.

Delete rule. A definition of what action the database should take when there is an attempt to delete a parent record.

Update rule. A definition of what action the database should take when there is an attempt to update a parent record.

A Simple Referential Integrity Example

A database contains an employee file and a department file. Both files have a department number field named DEPTNO. The related records of these database files are those for which employee.DEPTNO equals department.DEPTNO.

Creating a Referential Constraint

The files involved in a referential constraint relationship must be physical files with a maximum of one member. A referential constraint is a file-level attribute. You can create a constraint definition for a file before the member exists. You can create the member for each file later.

Certain conditions must be met before you can create a referential constraint:

- There must be a parent file with a key capable of being a parent key. If the parent file has no primary key or unique constraint, the systems tries to add a primary key constraint to the file.
- There must be a dependent file with certain attributes that match those of the parent file:

* Sort sequence (SRTSEQ) must match for data types CHAR, OPEN, EITHER, and HEX.
* CCSID must match for each SRTSEQ table unless one or both CCSIDs is 65535.
* Each sort sequence table must match exactly.

- The dependent file must contain a foreign key that matches the following attributes of the parent key:
  * Data type
  * Length
  * Precision (if packed, zoned, or binary)
  * CCSID (unless either is 65535)
  * REFSHIFT (if data type is OPEN, EITHER, or ONLY)

CONSTRAINT RULES

Referential constraints allows you to specify certain rules for the system to enforce. Referential constraint rules apply to parent key deletes and updates.

Constraint rules are set with the Add Physical File Constraint (ADDPFCST) command. Specify the delete rule with the DLTRULE parameter and the update rule with the UPDRULE parameter.

Delete Rules

There are five possible values for the DLTRULE parameter. The delete rule specifies the action to be taken when a parent key value is deleted. Null parent key values are not affected by the delete rule.

- *NOACTION (This is the default value.)

  Record deletion in a parent file will not occur if the parent key value has a matching foreign key value.

- *CASCADE

  Record deletion in a parent file causes records with matching foreign key values in the dependent file to be deleted when the parent key value matches the foreign key value.

- *SETNULL

  Record deletion in a parent file updates those records in the dependent file where the parent non-null key value matches the foreign key value. For those dependent records that meet the preceding criteria, all null capable fields in the foreign key are set to null. Foreign key fields with the non-null attribute are not updated.

- *SETDFT

  Record deletion in a parent file updates those records in the dependent file where the parent non-null key value matches the foreign key value. For those dependent records that meet the preceding criteria, the foreign key field or fields are set to their corresponding default values.

- *RESTRICT

  Record deletion in a parent file will not occur if the parent key value has a matching foreign key value.

*Note: The system enforces a delete *RESTRICT rule immediately when the deletion is attempted. The system enforces other constraints at the logical end of the operation. The operation, in the case of other constraints, includes an trigger programs that arc run before or after the delete.*

Update Rules

There are two possible values for the UPDRULE parameter. The UPDRULE parameter identifies the update rule for the constraint relationship between the parent and dependent files. The update rule specifies the action taken when parent file updating is attempted.

- *NOACTION (This is the default value.)

  Record update in a parent file does not occur if there is a matching foreign key value in the dependent file.

- *RESTRICT

  Record update in a parent file does not occur if there is a non-null parent key value matching a foreign key value.

*Note: The system enforces an update *RESTRICT rule immediately when the update is attempted. The system enforces other constraints at the logical end of the operation.*

If you are performing inserts, updates, or deletes on a file that is associated with a referential constraint and the delete rule, update rule, or both is other than RESTRICT, you must use journaling. Both the parent and dependent files must be journaled to the same journal. In addition, you are responsible for starting the journaling for the parent and dependent files with the Start Journal Physical File (STRJRNPF) command.

If you are inserting, updating, or deleting records to a file that is associated with a referential constraint that has a delete rule, update rule, or both rules, other than *RESTRICT, commitment control is required.

If you have not started commitment control, it will automatically be started and ended for you.

Defining the Parent File

A parent file must be a physical file with a maximum of one member. You can create the parent file in a constraint with the Create Physical File (CRTPF) command specifying a unique, ascending, and non-null key--the primary key. You can also use an existing file as a parent file. The primary access path to such a file is potentially a parent key. Potentially

means the user must explicitly change the access path into a parent key through SQL or the Add Physical File Constraint (ADDPFCST) command.

Alternately, a unique key access path can be used as the parent key. A primary key is a unique key with special attributes.

If, after the file is created, neither the primary nor unique key will suffice as the parent key, you have the following options:

- Delete the file and recreate it with the appropriate keys.
- Add a unique or primary key constraint to the created file.
- You can add a primary key to a file with the Add Physical File Constraint (ADDPFCST) command specifying *PRIKEY for the TYPE parameter. You must also specify the key field or fields with the KEY parameter.

  If a primary key already exists for the file, the ADDPFCST command with TYPE(*PRIKEY) will fail because a file can have only one primary key. You must first remove the existing primary key with the Remove Physical File Constraint (RMVPFCST) command, then you can add a new primary key.

- You can add a unique constraint to a file with the Add Physical File Constraint (ADDPFCST) command specifying *UNQCST for the TYPE parameter. You must also specify the key field or fields with the KEY parameter. You can also add a unique constraint with SQL ALTER TABLE.

  If the parent file does not have an existing keyed access path that can be shared for the primary key or unique constraint, the system creates one.

Defining the Dependent File

A dependent file must be a physical file with a maximum of one member. You can create the file as you would any physical file or use an existing file.

It is not necessary that the dependent file have a keyed access path when you create the actual constraint. If no existing access paths meet the foreign key criteria, an access path is added to the file by the system.

Verifying Referential Constraints

During the creation of a constraint with the ADDPFCST command, the system verifies that every non-null foreign key value has a matching parent key value. It is not uncommon to add a referential constraint to existing files that contain large amounts of data. ADDPFCST can take several hours to complete when a very large number of records are involved. The files are locked exclusively during the add process. You should take the time factor and file availability into account before creating a referential constraint.

Referential Integrity Enforcement

The I/O access for files associated with established and enabled constraints varies depending on whether the file contains the parent key or foreign key in the constraint relationship. Referential integrity enforcement is performed system-wide on all parent and dependent file I/O requests. The database enforces constraint rules for I/O requests from application programs and system commands such as the INZPFM command.

Foreign Key Enforcement

The delete and update rules specified at constraint creation apply to parent key changes. To maintain referential integrity, the database enforces a no-action rule for foreign key updates and inserts. This rule must be enforced on foreign key updates and inserts to ensure that every non-null foreign key value has a matching parent key value.

If a matching parent key does not exist for the new foreign key value, a referential constraint violation is returned and the dependent record is not inserted or updated.

PARENT KEY ENFORCEMENT

Delete Rules

When a record is deleted from a parent file, a check is performed to determine if there are any dependent records (matching non-null foreign key values) in the dependent file. If any dependent records are found, the action taken is dictated by the delete rule:

- **No Action**--if any dependent records are found, a constraint violation is returned and no records are deleted.
- **Cascade**--dependent records that are found will be deleted from the dependent file.
- **Set Null**--null capable fields in the foreign key are set to null in every dependent record that is found.
- **Set Default**--all fields of the foreign key are set to their default value when the matching parent key is deleted.
- **Restrict**--Same as no action except that enforcement is immediate.

If part of the delete rule enforcement fails, the entire delete operation fails and all associated changes are rolled back. For example, a delete cascade rules causes the database to delete ten dependent records but a system failure occurs while deleting the last record. The database will not allow deletion of the parent key and the deleted dependent records are re-inserted.

If a referential constraint enforcement causes a change to a record, the associated journal entry will have an indicator value noting the record change was caused by a referential constraint.

For example, a dependent record deleted by a delete cascade rule will have a journal entry indicator indicating the record change was generated during referential constraint enforcement.

Update Rules

When a parent key is updated in a parent file, a check is performed to determine if there are any dependent records (matching non-null foreign values) in the dependent file. If any dependent records are found, the action taken is dictated by the update rule specified for the constraint relationship:

- No Action--if any dependent records are found, a constraint violation is returned and no records are updated.
- Restrict--same as no action except enforcement is immediate.

CHECK PENDING

Check pending is the condition of a constraint relationship when potential mismatches exist between parent and foreign keys. When the system determines that referential integrity may have been violated, the constraint relationship is marked as check pending.

For example:

- A restore operation where only data in the dependent file is restored and this data is no longer synchronized (a foreign key does not have a parent) with the parent file on the system.
- A system failure allowed a parent key value to be deleted when a matching foreign key exists. This can only occur when the dependent and parent files are not journaled.

A foreign key value does not have a corresponding parent key value. This can happen when you add a referential constraint to existing files that have never before been part of a constraint relationship

Check pending status is either *NO or *YES.

To get a constraint relationship out of check pending, you must disable the relationship, correct the key (foreign, parent, or both) data, and then re-enable the constraint. The database will then re-verify the constraint relationship.

When a relationship is in check pending, the parent and dependent files are in a situation that restricts their use. The parent file I/O restrictions are different than the dependent file restrictions. Check pending restrictions do not apply to constraints that are in the established/disabled state (which are always in check pending status).

Dependent File Restrictions in Check Pending

A dependent file in a constraint relationship that is marked as check pending cannot have any file I/O operations performed on it. The file mismatches between the dependent and parent files must be corrected and the relationship taken out of check pending before I/O operations are allowed. Reading of records from such a file is not allowed because the user or application may not be aware of the check pending status and the constraint violation.

Parent File Restrictions in Check Pending

The parent file of a constraint relationship marked as check pending can be opened but is limited in the types of I/O allowed:

- Read is allowed
- Insert is allowed
- Delete is not allowed
- Update is not allowed

Check Pending and the ADDPFCST Command

The ADDPFCST command causes the system to verify that every non-null foreign key value has a matching parent key valueIf the database marks the newly created constraint relationship as check pending, the constraint is disabled so the data can be corrected. Once the data has been corrected, the constraint should be re-enabled so the database can again check that each record in the dependent file meets the constraint definition.

You can identify the records that are in violation of the constraint with the Display Check Pending Constraint (DSPCPCST) command. In this circumstance with very large numbers of records, the DSPCPCST command can also take a long time to process.

* ***Enabling and Disabling a Constraint***

The Change Physical File Constraint (CHGPFCST) command enables or disables one or more referential constraint relationships. Always specify the dependent file when changing a constraint; you cannot disable or enable a constraint by specifying the parent file. You must have a minimum of object management authority (or ALTER privilege) to the dependent file to enable or disable a constraint.

While a constraint is being enabled or disabled, the parent and dependent files are locked, both members are locked, and both access paths are locked. The locks are removed when the enable or disable is complete.

Attempting to enable an enabled constraint or disable a disabled constraint does nothing but cause the issuance of an informational message.

An established/disabled or check pending constraint relationship can be enabled. The enabling causes re-verification of the constraint. If verification finds mismatches between the parent and foreign keys, the constraint is marked as check pending.

Disabling a constraint relationship allows all file I/O operations for both the parent and the dependent, providing the user has the correct authority. The entire infrastructure of the constraint remains. The parent key and foreign key access paths are still maintained. However, there is no referential enforcement performed for the two files in the disabled relationship. All remaining enabled constraints are still enforced.

Disabling a constraint can allow file I/O operations in performance critical situations to run faster. Always consider the trade-off in this kind of a situation. The file data can become referentially invalid and the database will eventually take the time to reverify the relationship after the constraint is enabled again.

Note: Users and applications must be cautious when modifying files with a constraint relationship in the established/disabled state. Relationships can be violated and not detected until the constraint is re-enabled.

The Allocate Object (ALCOBJ) command can allocate (lock) files while a constraint relationship is disabled. This allocation prevents others from changing the files while this referential constraint relationship is disabled. A lock for exclusive use should be requested so that other users can read the files. Once the constraint is enabled again, the Deallocate Object (DLCOBJ) command unlocks the files.

When you enable or disable multiple constraints, they are processed sequentially. If a constraint cannot be modified, you receive a diagnostic message and function proceeds to the next constraint in the list. When all constraints have been processed, you receive a completion message listing the number of constraints modified.

Delete File (DLTF)

The DLTF command has an optional keyword that specifies how referential constraint relationships are handled. The RMVCST keyword applies to the dependent file in a constraint relationship. The keyword specifies how much of the constraint relationship of the dependent file is removed when the parent file is deleted:

***RESTRICT**

If a constraint relationship is defined or established between a parent file and dependent file, the parent file is not deleted and the constraint relationship is not removed. This is the default value.

***REMOVE**

The parent file is deleted and the constraint relationship and definition are removed. The constraint elationship between the parent file and the dependent file is removed. The dependent file's corresponding foreign key access path or paths and constraint definition are removed.

***KEEP**

The parent file is deleted and the referential constraint relationship definition is left in the defined state. The dependent file's corresponding foreign key access path and constraint definition are not removed.

Remove Physical File Member (RMVM)

When the member of a parent file in a constraint relationship is removed, the constraint relationship is put in the defined state. The foreign key access path and referential constraint definition are not removed. The parent key access path is removed because the parent member was removed, but the parent constraint definition remains at the file level.

When the member of a dependent file in a constraint relationship is removed, the constraint relationship is put in the defined state. The parent key access path and

constraint definition are not removed. The foreign key access path is removed because the dependent member was removed, but the referential constraint definition is not removed.

Save/restore

If the parent file is restored either to the same library or to a different library, the system cross reference file is used to locate the dependent file of a defined referential constraint. An attempt is made to establish the constraint relationship.

If the dependent file is restored, the TOLIB is used to determine constraint relationships:

- If both the parent and dependent files are in the same library, the referential constraint relationship is established with the parent file in the TOLIB.
- If the parent and dependent files are in different libraries, the referential constraint relationship of the duplicated dependent file is established with the original parent file.

The order of the restore of dependent and parent files within a constraint relationship, does not matter (parent restored before dependent or dependent restored before parent), the constraint relationship is established.

REFERENTIAL CONSTRAINT CONSIDERATIONS AND LIMITATIONS

- A parent file must be a physical file.
- A parent file can have a maximum of one member, MAXMBR(1).
- A dependent file must be a physical file.
- A dependent file can have a maximum of one member, MAXMBR(1).
- A constraint can be defined when both or either of the dependent and parent files have zero members. A constraint cannot be established unless both files have a member.
- A file can have a maximum of one primary key but may have many parent keys.

- There is maximum of 300 constraint relations per file. This maximum value is the sum of:
  - * The referential constraints whether participating as a parent or a dependent, and whether the constraints are defined or established.
  - * The unique constraints.
- Only externally described files are allowed in referential constraints. Source files are not allowed. Program described files are not allowed.
- Files with insert, update, or delete capabilities are not allowed in *RESTRICT relationships.
- Constraint names must be unique in a library.
- Constraints cannot be added to files in the QTEMP library.

Constraint Cycles.

A constraint cycle is a sequence of constraint relationships in which a descendent of a parent file becomes a parent to the original parent file.

Constraint cycles are not prohibited in the DB2/400 database. However, you are strongly urged to avoid using them.

HANDLING DATABASE FILE ERRORS IN A PROGRAM

Error conditions detected during processing of a database file cause messages to be sent to the program message queue for the program processing the file or cause an inquiry message to be sent to the system operator message queue. In addition, file errors and diagnostic information generally appear to your program as return codes and status information in the file feedback area. (For example, the COBOL/400 language sets a return code in the file status field, if it is defined in the program.) For more information about handling file errors in your program, see your high-level language guide.

If your programming language allows you to monitor for error messages, you can choose which ones you wish to monitor for. The following messages are a small sample of the error messages you can monitor (see your high-level language guide and the CL Reference manual for a complete list of errors and messages you can monitor):

MessageIdentifier	Description
CPF5001	End of file reached
CPF5006	Record not found
CPF5007	Record deleted
CPF5018	Maximum file size reached
CPF5025	Read attempted past *START or *END
CPF5026	Duplicate key
CPF5027	Record in use by another job
CPF5028	Record key changed
CPF5029	Data mapping error
CPF502B	Error in trigger program
CPF502D	Referential constraint violation
CPF5030	Partial damage on member
CPF5031	Maximum number of record locks exceeded

CPF5032	Record already allocated to job
CPF5033	Select/omit error
CPF5034	Duplicate key in another member's access path
CPF503A	Referential constraint violation
CPF5040	Omitted record not retrieved
CPF5072	Join value in member changed
CPF5079	Commitment control resource limit exceeded
CPF5084	Duplicate key for uncommitted key
CPF5085	Duplicate key for uncommitted key in another access path
CPF5090	Unique access path problem prevents access to member
CPF5097	Key mapping error

Note: To display the full description of these messages, use the Display Message Description (DSPMSGD) command.

If you do not monitor for any of these messages, the system handles the error. The system also sets the appropriate error return code in the program. Depending on the error, the system can end the job or send a message to the operator requesting further action.

If a message is sent to your program while processing a database file member, the position in the file is not lost. It remains at the record it was positioned to before the message was sent, except:

- After an end-of-file condition is reached and a message is sent to the program, the file is positioned at *START or *END.
- After a conversion mapping message on a read operation, the file is positioned to the record containing the data that caused the message.

Anyone who stops learning is old,

Whether twenty or eighty.

Anyone who keeps learning

Stays young.

The greatest thing in your life is

To keep your mind young.

75
R_Saripalle@yahoo.co.in.

... *Henry Ford*

CBSI COMPLETE BUSINESS SOLUTIONS (INDIA) PRIVATE LIMITED

SYSTEMS INTEGRATION PARTNERS