filePro® Structured Query Language

fPSQL

For Single and Multi-user Computer Systems

Chapter 1

Chapter 2 Quick Reference Guide

fP Technologies, Inc.
8383 Craig Street, Suite 270
Indianapolis, In 46250
800-847-4740

Program Version Number: 5.0
Manual Version: April 17, 2003

www.fptech.net

Chapter 3 TABLE OF CONTENTS

FPSQL Quick Reference Page

FPSQL Quick Reference Guide

(Version 5.0)

This guide provides a quick reference to the commands, syntax, and procedures for using FPSQL as described in the FPSQL User Manual (Version 5.0)

Main Screen Command Keys

[U]	Update the current query
[N]	Clear the current query, type in a new query
[S]	Save query to query file
[L]	Load query from query file
[H]	Send text of query statement to default printer
[RETURN]	Execute the query
[P]	Execute the query, send default output to the printer
[X]	Exit FPSQL

Query Editing Keys

The query editor uses the usual filePro editing keys:

Windows	Typical UNIX/LINUX	
[ESC]	[ESC] [ESC]	Record current query
[Fl]	[Fl]	Insert a character space
[F2]	[F2]	Delete a character
[F3]	[Ctrl] [D]	Insert a blank line
[F4]	[Ctrl] [U]	Delete a line
[F7]	[Ctrl] [E]	Move to end of line
[PgUp]	[Ctrl] [P]	Move cursor up a screen
[PgDn]	[Ctrl] [N]	Move cursor down a screen
[Home]	[Home]	Move cursor to top of screen, to top of query
[Tab]	[Tab]	Move cursor right 8 characters
[Shift Tab]	[Esc Tab]	Move cursor left 8 characters
[Back Space]	[Back Space]	Delete character to left of cursor
[Ctrl] [End]	[Ctrl] [O]	Delete rest of line

Plus four additional keys:

[F5]	[Ctrl] [R]	Toggle Insert/typeover modes
[F6]	[Ctrl][F]	Display filePro files
[F8]	[Ctrl][Z]	Copy/Paste a line
[F10]	[Ctrl][X]	Help

Direct Execution

You may also execute a query directly from your operating system "fp" subdirectory by entering at the command line:

fpsql queryfilename

FPSQL will run the query and exit when finished.

Queries are normally saved to and loaded from the "fp\sql" directory, unless it has been given a file name including a "\". In that case, FPSQL will use the path specified. You can use the FPSQL environmental variable to override the default directory.

Help:

While editing a query (in [N]ew or [U]pdate modes), the HELP key ([Fl0] or [Ctrl][X]) will bring up a list of subjects that help is available for. Select the subject you want by using the arrow keys, or typing the first few letters of the subject.

Load/Save Queries:

Queries are normally stored in the "fp/sql" directory. However, you may load/save from any directory by giving the full pathname to the file when prompted for a file name.

Query File Directory

[L]oad or [S]ave query, then [F1O] or [Ctrl][X], will show you all existing queries in the default directory. You may select which query you want simply by moving the highlight over the name (use arrow keys) and pressing ENTER. You may also type the first few letters of the name and the highlight will move automatically.

filePro File Directory

To display filePro files: in [N]ew or [U]pdate modes, press [F6] or [Ctrl][F]

Wide-screen Terminals: (UNIX/LINUX only)

If a query requires more than 80 columns, and your terminal can support it, FPSQL will switch into 132-column mode to display the results of the query, and switch back to 80 columns when the query is finished.

NOTE: Two new termcap entries have been defined to permit the use of wide-screen terminals:

PW	Set terminal to wide (132 column) mode
PX	Set terminal to normal (80 column) mode

Passwords:

Password security is currently implemented by using the creation password. In order to query any data within a file, you must know the creation password for that file.

Query Statement:

The general form of a query is:

SET options
SELECT fields
FROM files
WHERE conditions
GROUP BY fields
HAVING conditions
ORDER BY fields

The SELECT and FROM clauses are required, the others are optional.

In many cases, the SET, GROUP BY, and HAVING clauses are not needed.

NOTE: Reserved words may be entered as either upper or lower case. They are shown in capital letters here to make them stand out.

SET Clause:

The SET clause sends query output to file, printer, or screen, provides some control of print formatting, and enables querying qualified files.

SET OUTPUT SCREEN

Sends query output to the screen. (Default}

SET OUTPUT PRINTER

SET OUTPUT SPOOLER
Sends the output to the system's default printer.

SET OUTPUT PRINTER *'operating system command'*
SET OUTPUT SPOOLER *'operating system command'*
Sends the output to the operating system command specified *(UNIX/LINUX* only).
Example: SET OUTPUT PRINTER 'lp -copies 2'

SET OUTPUT *'filename'*
Sends the output to the file specified.

SET OUTPUT . . . WITH QUERY
Includes the text of the query in the output.
Example: SET OUTPUT PRINTER 'lp -copies 2' WITH QUERY

SET LINES *nn*
Sets the number of lines per page to *nn.*
Example: SET LINES 66

SET LINES *mm,nn*

Sets the number of lines to print per page to *mm* and the number of lines per page to *nn.*
Example: SET LINES 60,66

SET LINES 0 *(zero)*

A special case, which tells FPSQL not to generate any headings. This is useful if the output is to be used as a merge file to another program.

SET TITLE
SET TITLE ON

Includes a default title, in addition to the column headings, in the output.

SET TITLE OFF

No title will be printed. (Default)

SET TITLE ‘text’

Specifies what the title should be.

SET QUALIFIER qualifier name

Only one qualified file can be used in a query.

Default LINES values:

To the screen:	The height of the screen
To the printer:	60,66
To a file:	60,66

SELECT Clause: (required)

SELECT *fields:* Tells FPSQL what is to be displayed

"Fields" may be one or more of the following, separated by commas:

A real field, referenced by name: FirstName, LastName

A real field, referenced by number: @1, @29

A system maintained field: @RN, @CD

An associated field group: @A0, @B7

Any of the above fields, preceded by the file name:
Personnel.FirstName,managers.Department
maillist.@5, zipcodes.@RN

NOTE: This is in case the same field name appears in more than one file.

An asterisk, meaning all fields: *, maillist.*

A text literal, enclosed in single quotes: 'This is some text.','Some more text'

A numeric literal, (use without quotes): 1, 2, 3

An expression, using any combination of the above fields:
Salary * Hours_Worked, @TD – Last_Raise

An aggregate function: (AVG, MIN, MAX, SUM, COUNT)
MIN(Salary), AVG(@TD – BirthDay), COUNT (*)

The MID function: MID(birthdate,7,2), MID(lastname,1,1)

Using aggregate functions

Requires the use of the GROUP BY clause, or FPSQL will use an implied group of the entire file (grand totals.)

To specify fields by name

Since filePro allows a field name to contain any character, which is not allowed in ANSI SQL, you can use the following methods:

- You can use the underscore ("_") to designate any non-alphanumeric character in the field name. For example, for the field "Code (A, or B)" you could use "Code_A_or_B_"

- You may truncate the name at any place where you could use the underscore. For example, in the field above, you could have used just "Code" instead. If the name is not unique, FPSQL will use the first field that matches the given name.

"Alias"

A field/expression can be followed by an "alias" which will be used as the column heading instead of the name of the field, or the text of the description:

SELECT Hourly_Salary * Hours_Worked 'Weekly Salary'

FPSQL will use 'Weekly Salary' as the column heading in the output.

Please Note: FPSQL does not support 5.0 Blob or Memo Fields

FROM Clause: (required)

FROM *files:* Tells FPSQL which files are to be used in the query.

"Files" may be one or more of the following, separated by commas:

A filePro file name: maillist, zipcodes

A filePro file name, followed by an "alias"
personnel employees, personnel managers

An alias is needed if the file is to be used more than once in the same query, and that alias would then be prefixed to the field names, as described in the SELECT clause.

The order in which the files are listed is not important.

WHERE clause:

WHERE conditions: Tells FPSQL which records are to be selected.

The general form of a condition is:

compare op compare op compare ...

where "op" can be AND or OR.

Compares can be grouped by placing them within brackets:

[compare op compare] op [compare op compare]

The general form of a comparison is: field rel field
NOT field rel field

Where "rel" is a relation: < <= = >= > <>

and "field" is the same as in the SELECT clause, (except that aggregate functions are not permitted), or a subquery.

Other comparisons are:

field IS BETWEEN field AND field
field IS NOT BETWEEN field AND field
field IS IN (field, field, field, . . .)
field IS NOT IN (field, field, field, . . .)
field IS LIKE 'text'
field IS NOT LIKE 'text'
field is NULL
field is NOT NULL

NOTES:

a IS BETWEEN b AND c is the same as b <= a AND a <= c

a IS IN (b, c, d) is the same as a = b OR a = c OR a = d

a IS NULL is the same as a =''

The IS LIKE comparison allows wild-card matching:

"_" represents any one character
"%" represents any string of zero or more characters.

IS LIKE '%text%' is the same as filePro's contains (CO) relation

Examples:

LastName IS BETWEEN ‘A’ AND ‘M’ AND State IS IN (‘NY’,’NJ’,’CT’)

Hourly_Salary * Hours_Worked > 750

Employees.id = project.manager_id

[salary < 5000 OR hours < 20]AND NOT[age < 18 OR citizen = ‘N’

“Exists” Subclause:

Used with “where” clause and a subquery:

Example:

```
select * from ssalesmn
where exists
        select name
        from ssalesmn
        where sales > 2800
```

If the subquery is true (generates output) the outer query is executed. If the subquery is not true, outer query is not executed.

GROUP BY Clause:

GROUP BY fields: Tells FPSQL that subtotal records are to be generated instead of detail records.

“Fields” is the same as described in the SELECT clause, except that aggregates are not permitted.

Only one level of subtotals (the innermost) is generated.

GROUP BY cannot be used with ORDER BY.

HAVING Clause:

HAVING conditions: Tells FPSQL which groups are to be selected.

"Conditions" is the same as described under the WHERE clause, except that aggregate functions are permitted.

The HAVING clause cannot be used without the GROUP BY clause.

Examples:

```
        GROUP BY Department
HAVING AVG(Salary) > 20000
```

```
        GROUP BY State
        HAVING COUNT(*) > 10 OR AVG(Amount_Due) > 1000
```

ORDER BY clause:

ORDER BY fields: Tells FPSQL how to sort the output.

"Fields" is the same as described in the SELECT clause, including aggregate functions. The major key is listed first, followed be minor keys, and the innermost key listed last.

ORDER BY does not use existing filePro indexes when sorting.

If no ORDER BY clause is specified, then the output will be sorted by the fields in the GROUP BY clause, if any. If neither clause is used, the output is in no specific order.

FPSQL (Version 5.0) versus ANSI X3.135-1986 SQL:

In FPSQL:

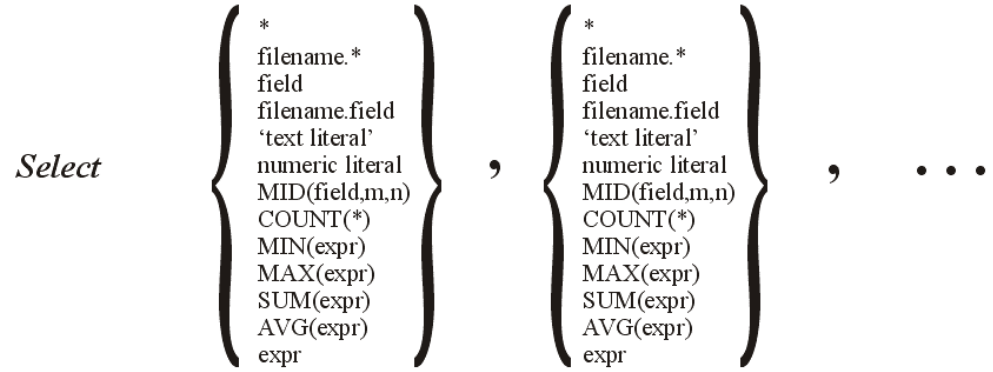
- SELECT DISTINCT clause not implemented.
- Password security is based on the creation password.
- FPSQL is case insensitive in sorts and comparisons, just like the rest of filePro.
- SET clause added.
- filePro's system-maintained fields can be used: @RN, @CD, @TD, . . .
- filePro's additional field types can be used: MDY, HMS, $, . . .
- Associated fields can be used: @A0, @Al, @A2, . . .
- Fields can be referenced by number: @1, @2, @3, . . .
- The MID function added: MID(field,l,2)

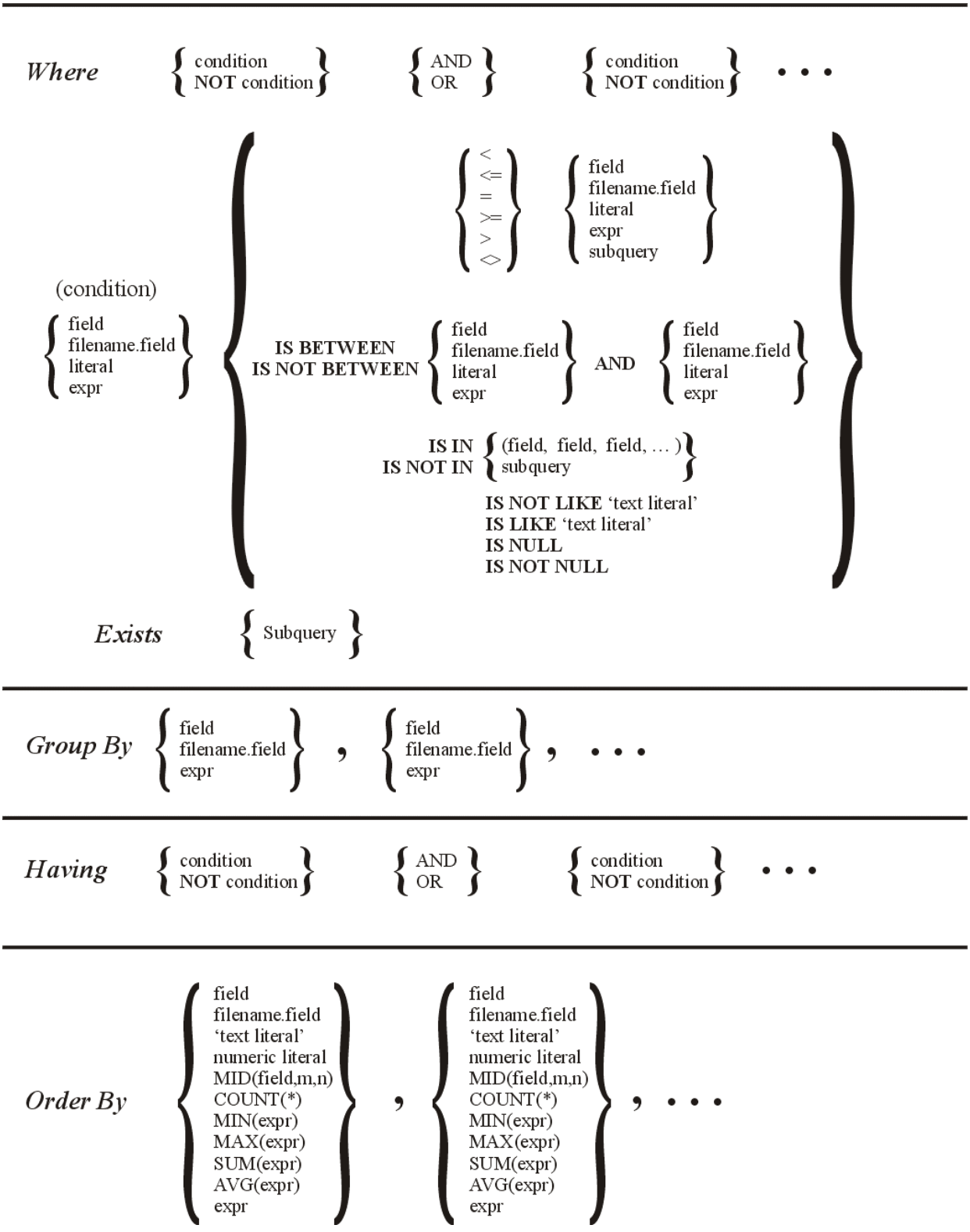
FPSQL SYNTAX DIAGRAMS

The following diagrams represent the syntax of all the clauses, except "set", in a query statement.

Key to diagrams:

Large brackets { }	enclose a list of allowable entities:
	fields, files, numbers, expressions, aggregate functions, etc., for use in the clause.
Comma (,)	means that the following entity is optional.
Ellipsis (. . .)	means that preceding may be repeated any number of times.




Where
condition
NOT condition
AND
OR
condition
NOT condition
• • •
(condition)
field
filename.field
literal
expr
<
<=
=
>=
>
<>
field
filename.field
literal
expr
subquery
IS BETWEEN
IS NOT BETWEEN
field
filename.field
literal
expr
AND
field
filename.field
literal
expr
IS IN
IS NOT IN
(field, field, field, …)
subquery
IS NOT LIKE 'text literal'
IS LIKE 'text literal'
IS NULL
IS NOT NULL
Exists
Subquery
Group By
field
filename.field
expr
,
field
filename.field
expr
, • • •
Having
condition
NOT condition
AND
OR
condition
NOT condition
• • •
Order By
field
filename.field
'text literal'
numeric literal
MID(field,m,n)
COUNT(*)
MIN(expr)
MAX(expr)
SUM(expr)
AVG(expr)
expr
,
field
filename.field
'text literal'
numeric literal
MID(field,m,n)
COUNT(*)
MIN(expr)
MAX(expr)
SUM(expr)
AVG(expr)
expr
, • • •

filePro® Structured Query Language

fPSQL

For Single and Multi-user Computer Systems

Users Manual

fP Technologies, Inc.
8383 Craig Street, Suite 270
Indianapolis, IN 46250
800-847-4740

Program Version Number: 5.0
Manual Version: March 30, 2000

www.fptech.net

TABLE OF CONTENTS

Chapter Page

Chapter Page

Chapter page

Appendixes

CHAPTER 1:
GETTING STARTED

In this chapter:

- Make sure you have everything
- Installing FPSQL
- Starting the FPSQL program
- How FPSQL works with filePro files
- How to use this manual
- Organization of this manual
- Style conventions used in this manual

Make sure you have everything

Your FPSQL package should contain the following:

- Program diskette, tape or CD
- Installation instructions
- User manual (see notes)
- Quick reference guide (see notes)
- Warranty, software license agreement including the software registration card

Check to see that both the program diskette and the installation instructions refer to your computer and/or operating system.

The user manual is a complete reference for FPSQL (filePro SQL). It is also organized for easy learning.

The quick reference guide puts all the commands and capabilities of FPSQL at your fingertips.

Be sure to fill out and mail in your software registration card so that you may have free use of the Customer Technical Support during the warranty period.

Notes: A printable ".pdf" file is furnished on the Program diskette, Tape or CD.

Installing FPSQL

See the installation instructions in the package (it should refer to your computer and/or operating system). For most systems, installation involves little more than inserting the FPSQL diskette m the floppy drive, typing “install” and following the prompts on the screen. FPSQL will install itself in the proper filePro directories and add a FPSQL menu selection to the main menu. Certain demo files, which will enable you to duplicate the sample queries used in the manual, are also installed.

Starting the FPSQL program

Once installed, FPSQL can be started from the filePro main menu. It will be in the “Runtime Programs” column, menu selection “S”. You may also start FPSQL from your operating system “fp” subdirectory: simply enter “fpsql”.

How FPSQL works with filePro and filePro Plus

FPSQL will work with file Pro 16/16+ files, Profile 16 files, filePro Plus files and non-filePro files that you have defined to filePro.

How to use this manual

For learning FPSQL, you should read the chapters in order and do the exercises (“quiz queries”) at the ends of the chapters: the answers are in Appendix E. A good idea is to have FPSQL up and running on your computer so that you can gain hands-on experience duplicating the sample queries in the manual and experimenting with your own variations of them. You might also try to compose the example queries before you look at the query output in the manual. Once you are familiar with the program, use the manual as a reference.

Organization of this manual

Chapter 2	explains what FPSQL is and what it can do.
Chapter 3	gives you the mechanics of using FPSQL: entering, executing, saving, loading your queries; viewing files, etc.
Chapter 4	documents the files used in all the example queries in the manual.

Chapters 5 through 10 introduce:

- "clauses" the building blocks used to construct queries
- query statement syntax
- comparative, aggregate, and mathematical operations you can use in queries
- "subqueries" and "variable" queries, for finding the answers to more complex questions.

Chapter 11 shows you how a query can use data from more than one file at a time.

Chapter 12 covers the "set" clause which enables you to redirect and control the output of your queries.

Appendix A documents some common on-screen error messages and their causes.

Appendix B is a list of "reserved words" in FPSQL.

Appendix C contains diagrams of FPSQL syntax.

Appendix D documents the differences between FPSQL and the American National Standards Institute (ANSI) standard for SQL.

Appendix E contains answers to the "quiz queries".

Style conventions used in this manual

Keys and commands described in the text

References to keys and commands will give the Windows version followed by the UNIX/LINUX version.

Example: press [F5] or [Ctrl][R]

Note that keyboard buttons to be pressed are shown inside square brackets.

If more than one key is to be pressed sequentially, they are separated by commas:

press [Esc] or [Esc],[Esc]

If they are to be pressed simultaneously, there is no comma:

press [Ctrl][Break] or [Break]

Keys and commands shown on the screen

The screens reproduced in this manual show the Windows version of FPSQL. Your version of FPSQL will show, on your screen, the keys and commands appropriate for your computer's operating system.

Text to be entered

In a sentence, text to be entered will have quotes ("") around it:

Example: Then type "FPSQL" and press [Return].

Or, as when used in an example, text may be indented and on a separate line:

FPSQL [Return]

Prompts and messages

These will be marked with a 8 and reproduced as they appear on the screen:

8 | Aggregate functions not permitted |

CHAPTER 2:
WHAT IS FPSQL?

In this chapter:

- What is SQL?
- What is FPSQL?
- A sample query
- The general form of a query
- Reserved words

What is SQL ?

SQL (usually pronounced “sequel”) means Structured Query Language. It is a relational inquiry and data manipulation language using English reserved words. SQL enables you to quickly manipulate your data and view the results. It’s easy enough for non-programmers to learn and use, yet powerful enough to satisfy data processing professionals.

What is FPSQL?

filePro’s implementation of SQL enables you to query the data but not manipulate it. FPSQL is especially easy and powerful. It has a simple-to-use screen editor, detailed online Help, easy file-handling, and flexible control over query output, making FPSQL the state-of-the-art in SQL implementations. With FPSQL, you can look at your data in relationships limited only by your imagination.

A Sample query

To get an idea of what FPSQL is like, let’s look at a sample query. Using FPSQL’s demo file, “sproduct” (see Chapter 4), the following is how you would ask FPSQL for a list of products whose prices are higher than $99. The list should include the product number, description, and prices with the lowest price first. You would type in the query as follows:

```
select product_number, description, price
from sproduct
where price > 99
order by price
```

NOTE:

When typing in queries you’re not restricted to any set format of lines, spacing, or upper and lower case (except for file names in UNIX/LINUX). As long as the elements are in the right order and the syntax is correct, the query will be executed.

This query, when executed, would produce the following result:

```
Product_number          description            price
     ------------------------------------------------------
     67-8901            Humidity Meter          99.50
     78-9012            Barometer Kit          199.60
     23-4567            Metal Tracker          219.00
```

The general form of a query

Queries are constructed of clauses containing "reserved" words (words which have a special meaning to FPSQL). The following table shows the clauses, what type of information they contain, and the order in which they are used in a query statement. A query must have the "select" and "from" clauses; the others are optional:

CLAUSE		FUNCTION
"set..."	options	redirects output to printer, file, screen; controls printer and print formatting
"select..."	data (fields)	tells FPSQL what to display
"from..."	files	tells FPSQL where to get the data (fields)
"where..."	conditions	tells FPSQL what conditions apply in selecting the data (fields)
"group by..."	data (fields)	tells FPSQL on what data (field) to subdivide the records. Generates subtotals
"having..."	conditions	tells FPSQL what conditions apply in selecting which groups to display
"order by..."	data (fields)	tells FPSQL what data (fields) to sort the output by

Reserved words

These are words which have specific meaning to FPSQL and, therefore, whenever possible, should not be used in naming files or fields in queries. The clauses we just looked at contain the reserved words: “set”, “select”, “from”, “where”, “group by”, “having”, “order by”. Here is a complete list:

and	group	not	select
asc	having	null	separator
avg	help	off	set
between	in	on	spooler
by	insert	or	start
count	into	order	sum
desc	is	output	title
edit	like	printer	unique
end	lines	query	unlock
exists	max	records	where
fields	mid	restart	width
from	min	screen	with

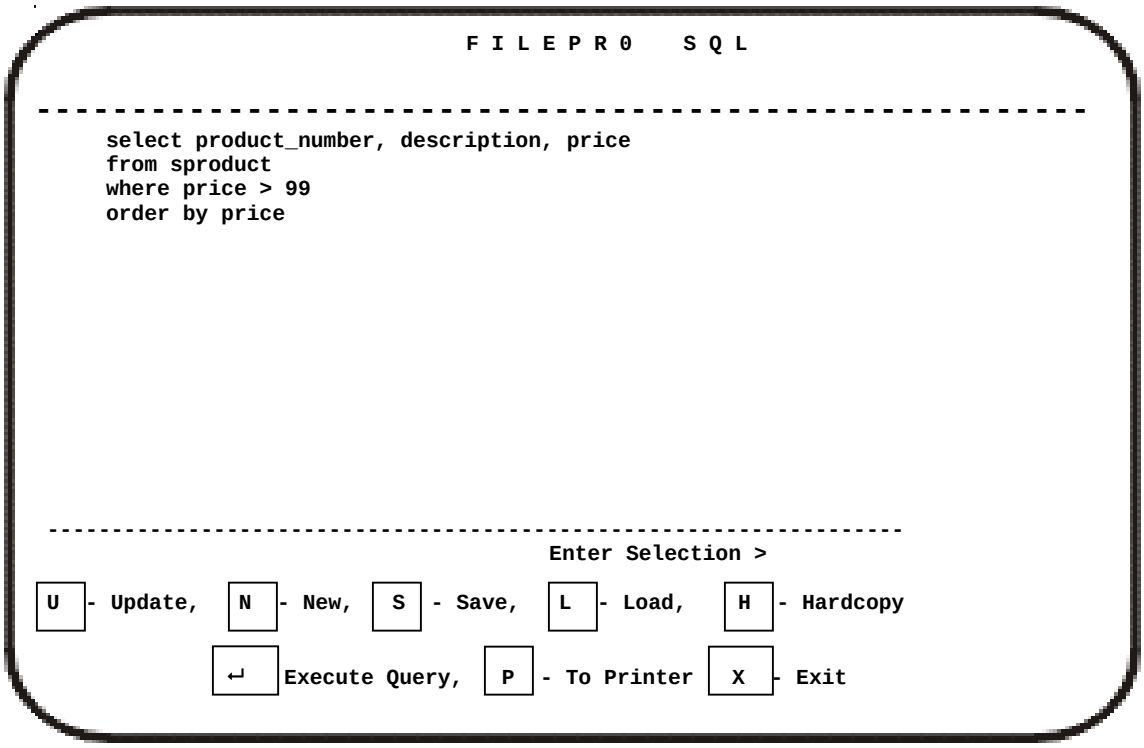
CHAPTER 3:
USING FPSQL

In this chapter:

- The main screen and commands
- Typing in a query
- Including comments in a query
- Editing a query
- Executing a query
- Passwords
- Viewing query output
- Redirecting query output to printer or file
- Controlling printed output formatting
- Saving a query
- Loading a saved query
- Viewing the query file directory
- Viewing field headings, lengths, and edits
- Using online Help

The main screen and commands

Below is the FPSQL main screen, shown with a sample query typed in.



FPSQL main screen with sample query

Main screen command keys

Press	In Order To:
[U]	update the current query
[N]	clear the current query, type in new query
[S]	save query to query file
[L]	load query from query file
[H]	send text of query statement to default printer
[Return]	execute query
[P]	execute query, send output to default printer
[X]	exit FPSQL

Typing in a query

Query statements are typed in the query statement area (the area between the horizontal rules) on the FPSQL main screen (see above). When FPSQL starts, you will see the main screen; the query statement area will be blank. The cursor will be in the "Enter Selection" field. Press [N] for "New" and the cursor will move to the first space of the first line. The main screen command options will be replaced by:

▶ F10 - Help, F6 - Display Fields, ESC - Record, BREAK - Cancel

These command options are covered later in this chapter.

Insert and typeover modes

There are two typing modes: "insert" and "typeover". You can toggle from one to the other with the [F5] or [Ctrl][R] keys. When you type in insert mode, existing characters to the right of the cursor are moved to the right to make room for the new characters. Also, the word "Insert" appears in the upper left corner of the screen. When you type in "typeover" mode, the new characters replace the existing characters; there is no word displayed in the upper left corner.

Rules for typing

There are almost no rules for the way your query statement is entered in the query statement area. As long as everything in your query statement is in the correct order, the query will be executed. You may type in upper or lower case (except for file names in UNIX/LINUX). You may break lines wherever you wish, use extra line returns and extra spaces between words. You can type as many as 79 characters on one line but, at the end of each line, you must press [Return} or [Enter] to move the cursor to the next line.

Editing

You may use the arrow keys to move the cursor left/right one character, or up/down one line. You may use the [Del] and [Back Space] keys to remove characters. Text editing is similar to filePro.

Long Queries - more than one screen

If your query statement is longer than one screen, you can scroll up and down a line at a time using the up and down arrow keys: [□], [□]; or a screen at a time using the [PgUp], [PgDn] keys or [Ctrl][P] (up), [Ctrl][N] (down).

NOTE: A complete list of editing keys and functions is in the "Editing a query" section, which follows the next two sections.

Recording the query

When you are finished and ready to execute or save your query to a query file, you must first "record" it. Simply press [Esc] or [Esc][Esc]. You will go back to the main screen, which will display your query, and the cursor will be in the "Enter Selection" field.

Including comments in a query

It may be helpful at times to add explanatory notes or comments to a query to make it easier for others to understand or for your own future reference. You may insert comments before, within, or after a query statement by preceding the comment with the "#" sign. FPSQL will regard everything from the "#" to the end of the line as a comment and not to be executed. If your comments require an additional line, start the additional line with the "#" again, as in the following example:

```
select name, sales, commission, salary*9 #total of
#monthly salary Jan. through Sept., 1987
from ssalesmn
order by salary
```

Editing a query

The editing keys are the same ones you use in filePro depending on your computer system:

FPSQL Editing Keys

The query editor uses the usual filePro editing keys:

Windows **Press:**	Typical UNIX/LINUX **Press:**	**In Order To:**
[ESC]	[ESC] [ESC]	Record current query
[Fl]	[Fl]	Insert a character space
[F2]	[F2]	Delete a character
[F3]	[Ctrl] [D]	Insert a blank line
[F4]	[Ctrl] [U]	Delete a line
[F7]	[Ctrl] [E]	Move to end of line
[PgUp]	[Ctrl] [P]	Move cursor up a screen
[PgDn]	[Ctrl] [N]	Move cursor down a screen
[Home]	[Home]	Move cursor to top of screen, to top of query
[Tab]	[Tab]	Move cursor right 8 characters
[Shift Tab]	[Esc Tab]	Move cursor left 8 characters
[Back Space]	[Back Space]	Delete character to left of cursor
[Ctrl] [End]	[Ctrl] [O]	Delete rest of line

Plus four additional keys:

[F5]	[Ctrl] [R]	Toggle Insert/typeover modes
[F6]	[Ctrl][F]	Display filePro files
[F8]	[Ctrl][Z]	Copy/Paste a line
[F10]	[Ctrl][X]	Help

The copy/paste key works as follows:

1. Position the cursor on the line you wish to copy.

2. Press [F8] or [Ctrl][Z]. The word "copy" will appear in the upper left corner of the screen.

3. Position the cursor on the line were you want the copied line to appear.

4. Press [F8] or [Ctrl][Z] again. The copied line will appear. The word "copy" will disappear. Any characters present on the line where you insert the copied line will move down to the next line.

To edit a query that has already been recorded (using [Esc] or {Esc][Esc]), you must first press [U] for "Update". The cursor will move to the first space of the first line in the query statement area and the main screen command options will be replaced with:

▶ F10 - Help, F6 - Display Fields, ESC - Record, BREAK - Cancel

Now you can edit the query.

Editing exercise

You might like to try the following to get used to editing query statements:

Type in the following query exactly as shown below:

```
#lists products with prices above $99
select product_number, description, price
from sproduct where price > 99 order by price
```

Then, use the editing keys to make it look like this:

```
select

     product_number, description, price

from

     sproduct

where

     price > 99

order by

     price

#lists products with prices above S99
```

Hint: use the copy/paste key ([F8] or [Ctrl][Z]) to move the comment line.

Provided there are no typos, either version of this query will work. However, note how much easier it is to scan and comprehend the second version. This flexibility in typing queries, and the ability to include comments, makes it easier for you to compose queries, and for others to read and understand them.

Executing a query

(Refer to the FPSQL main screen at the beginning of this chapter.) After you have typed in the query and pressed [Esc] or [Esc][Esc] to save it, press [Return] to execute it. If you have not used the "set" clause (see Chapter 12) to redirect the output to a printer or file, you will see the output on your screen.

Screen Display

While FPSQL is executing the query, you may see briefly on your screen:

```
▶ Sorting Keys . . .
                          Records Read: n
                          Selected: n
```

The number of records read will be updated as the query execution continues.

This screen is followed by,

```
▶ Writing Index . . .
                              Records Written: n
                              Selected: n
```

The number of records written is updated as the query execution continues. The next screen will contain the results of the query.

Direct Execution

You may also execute a query directly from your operating system "fp" sub-directory by entering at the command line:

```
fpsql queryfilename
```

FPSQL will run the query and exit when finished.

Queries are normally saved to and loaded from the "fp\sql" directory, unless it has been given a file name including a "\"'. In that case, FPSQL will use the path specified. You can use the FPSQL environmental variable to override the default directory.

No Output

When there is nothing in the data that meets your query's specifications, the following message is displayed:

```
▶          No Output for this Query
           Output for this Query

                Press ↵
```

Passwords:

FPSQL uses creation passwords for file security. In order to query a password-protected file you must know the creation password for the file. When you execute the query, either from FPSQL or your operating system command line, you will see the following:

```
▶          Enter Password for filename
```

FPSQL will allow you three tries to enter the password correctly.

Once you have entered it correctly, and as long as you remain in the FPSQL program, you will not be asked for it again when you query that file. However, each time you execute FPSQL and want to query the file, you will have to enter the password again (one time).

Viewing query output

The output from a query may be as little as one field or as many fields as will fit into 32000 columns across and an unlimited number of screens deep. Ultimately, it is the capacity of your computer system that sets limits on the FPSQL's output.

Output of more than one screen

If the output is more than one screen (23 lines) deep, you will see the following prompt at the bottom of the screen:

▶ **Press any key to continue >**

Press any key for the next screen.

(You can return to the main screen at any time when viewing output by pressing [Ctrl][Break] or [Break].)

At the bottom of the last screen you will see:

▶ **Press any key to continue >** **End of Query**

Pressing ANY KEY will take you back to the main screen, which will still have your query on it.

If, while viewing output of more than one screen, you want to go back to a screen already passed, you must execute the query again.

Output wider than one screen - horizontal scrolling

When your query output is wider than 80 columns and you are not using a "wide screen" terminal (see next section), or, if you are using a "wide-screen" terminal and the output is wider than 132 columns, you can scroll horizontally to view the output. The command options at the bottom of the screen will look like this:

▶ **PGUP** **PGDN** **□** **□** **- Scroll, ↵ -Continue**

The [PgDn] key will move you one screen to the right.

The [PgUp] key will move you one screen to the left.

The [ ] key will display the next field to the left.

The [ ] key will display the next field to the right.

"Wide-screen" terminals

If your terminal has a "wide-screen" capability, FPSQL will switch into 132-column display mode to display output that is more than 80 columns wide. When you are finished with the query, FPSQL will automatically return to 80-column mode. If the output is wider than 132 columns, FPSQL will allow you to scroll horizontally (see previous section).

Note: Two new termcap entries have been defined to permit use of wide-screen terminals:

PW	Set terminal to wide (132-column) mode
PX	Set terminal to normal (80-column) mode

Redirecting query output to printer or file

You can send the results of your query directly to a printer or file by using the "set" clause. The "set" clause is optional; without it, query output will go to your screen when you execute it using [Return]. If you press [P] instead, the output will go to the default printer (a "set" clause, if present, will override the "P" command.) In a query statement the "set" clause must always precede the "select" clause. More than one "set" clause can be used at a time.

To send query output to the printer, type:

set output printer

To send query output to a file, type:

set output 'filename'

See Chapter 12 for all the "set" clause commands and syntax.

Controlling printed output formatting

With the "set" clause you can also control the formatting of your printed query output. For example, you can specify page lengths and bottom margins. You can include or delete headings and titles. See Chapter 12 for all the "set" clause commands and syntax.

Saving a query

(Refer to the FPSQL main screen at the beginning of this chapter.) One of the options displayed at the bottom of the main screen is "S -Save". To save your query, press [S]. You will then see the following prompt:

```
▶ Save...
  Enter Query Name:
  Or press F10 for a list existing queries
```

Queries are normally saved to the "fp\sql" directory. However, you may save queries to another directory by giving the full path name when prompted for the query file name.

Type in the name you wish to give the query. Use the file naming conventions that apply to your computer's operating system. Then press [Return]. FPSQL will save the file and immediately take you back to the main screen which will still have your query on it.

If you had loaded a query (loading queries is covered in the next section) at some time prior to your "save" operation - let's say the query file name was "sales.09", when you go to save the query you will see an additional prompt:

```
▶ Save...
  Enter File Name:
  Or Press ↵ For 'sales.09'  F10  for a list of
  existing queries
```

This convenient feature allows you to use the last file name loaded for saving a query on the assumption, as may, often be the case, you loaded a query, modified it, and wish to save it again.

Loading a saved query

Another option displayed at the bottom of the FPSQL main screen is “L - Load”. Press [L] to load a query from a query file into the query statement area. The main screen command options will be replaced by the prompt:

```
▶ Load...
  Enter File Name: Or press [F10] for a list existing
  queries
```

At this time you may type in the query name or look at the query file directory. After you type in the query name press [Return]. You will then see the query in the query statement area and the main screen command options will have been restored.

Queries are normally loaded from the “fp\sql” directory. However, you may load queries from another directory by giving the full path name at the prompt.

If, when you try to load a query, the query statement area already contains a query - a newly composed one, or one that had been previously loaded and modified, the following message will appear:

```
▶ ┌──────────────────────────────────┐
  │ [WARNING] Query has been modified.
  │ Do you want to save it first?
  └──────────────── Press [Y] or [N]
```

If you press [N], the main screen command options will be replaced by the prompt:

```
▶ Load...
  Enter File Name: Or press [F10] for a list existing
  queries
```

At this time you may type in another query name or look at the query file directory. After you type in the query name press [Return]. The query in the query statement area will be replaced by the new one.

If you press [Y], and the query, on the screen is brand new, the main screen command options will be replaced by the prompt:

```
▶ Save...
  Enter File Name:
  Or press [F10] for a list existing queries
```

At this time you may save the query with a new name or look at the query file directory. After you type in the query name press [Return]. The query will be saved and the prompt will change to "Load...". Now you can load another query.

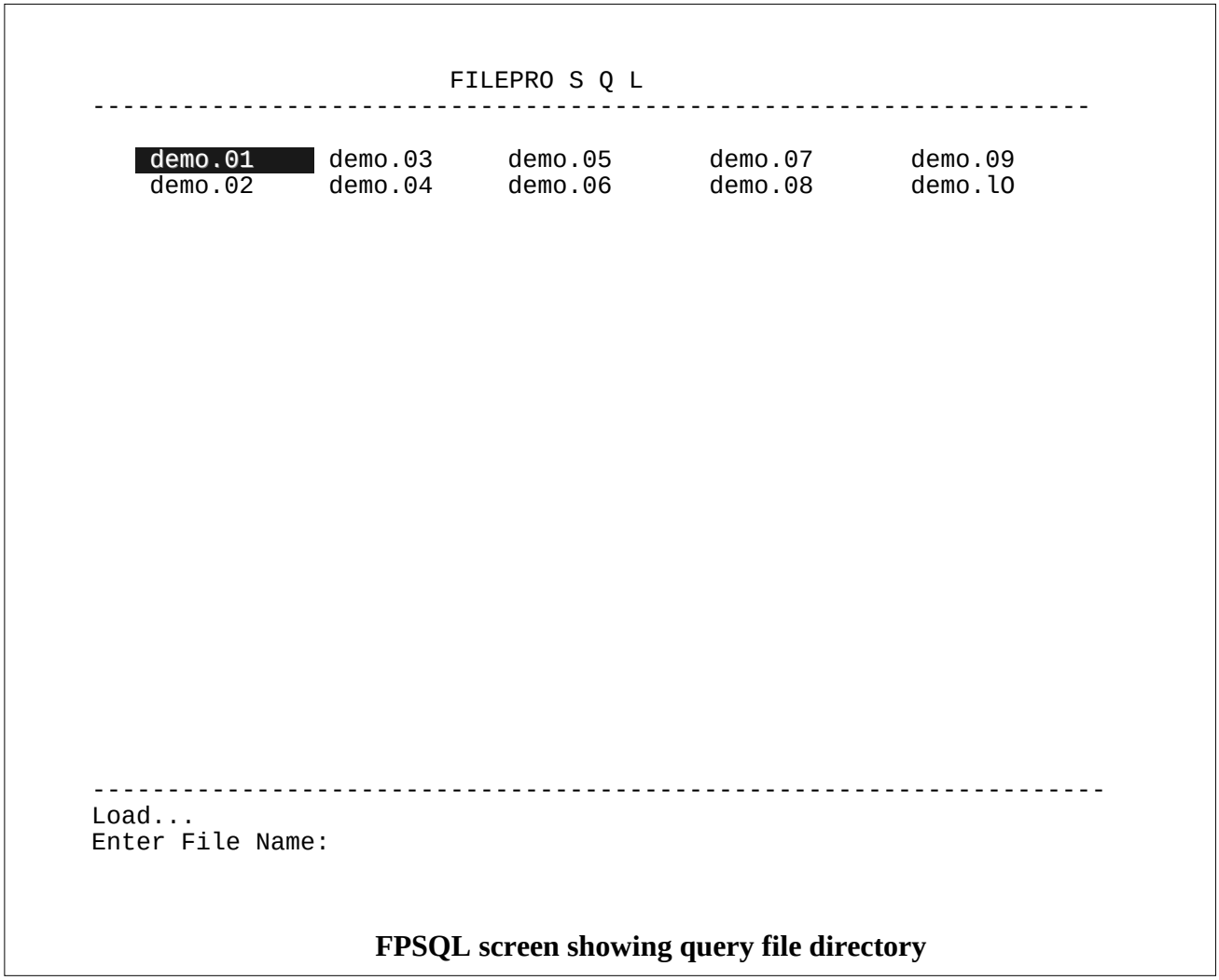
If you press [Y] and the query on the screen is one that had been loaded and modified, the main screen command options will be replaced by the prompt:

```
▶ Save...
  Enter File Name:
  Or Press [↵] For 'filename'  [F10] for a list of
  existing queries
```

At this time you may save the query under a new name, or by its original name (just press [Return]), or look at the query file directory. After typing in the query name press [Return]. The query will be saved and the prompt will change to "Load..." Now you can load another query.

Viewing the query file directory

There will be times when you will want to see the query file directory. All you have to do is press [S] ("Save") or [L] ("Load"), and, as you have learned in the last section, when you see the "Save..." or "Load..." prompt, press [F10]. Whatever is in the query statement area will be temporarily replaced by a wide format listing of all your query files, with the current query file name highlighted in reverse video, as in the sample screen on the next page.



FPSQL screen showing query file directory

You may select a file by any of three methods:

- type in the full file name and press [Return]
- type in a partial file name until the highlight moves to the desired file name and then press [Return]
- move the highlight to the desired file name with the directional arrow keys [□] [□] [□] [↓] and then press [Return]

As soon as you press [Return], you will return to the main screen where the query selected will be displayed.

Viewing field headings, lengths, and edits

To compose query statements you need to know your file names, field headings, and the nature of the data in the fields (field length and field edit type). You can quickly and easily check them while in FPSQL. Follow the procedure below.

From the main screen select “Update” or “New” mode (press [U] or [N]). The main screen command options will be replaced by:

```
▶ F10 – Help,  F6 – Display Fields,
   ESC – Record,  BREAK - Cancel
```

At this point press [F6] and you will see the following prompt:

```
▶ Enter File Name:
  Press F10 for a list of filePro files
```

Type in the file name. If the name uses the maximum file name length allowed by your computer system, a “Return” will automatically be entered when you type the last character of the file name (same as in filePro). If the file name is shorter than the maximum, you will have to press [Return].

The next screen you see will contain a list of that file's field numbers and headings. If, at the prompt, you had typed in "sproduct", one of the filePro demo files (see Chapter 4), you would see the following screen:

```
                              F I L E P R O    SQL

------------------------------------------------------------------------
1-      Product Number
2-      Description
3-      Price
4-      Quantity On Hand
5-      Reorder Point
6-      Total Sales
7-      Total Quantity




------------------------------------------------------------------------
File: sproduct        Enter Selection >
   F5  - Field Lengths and Edits, F6  - Different File
        PGUP  PGDN  - Scroll Fields,  H  - Hardcopy, ↵  Return
```

FPSQL screen (with Windows command keys) displaying field headings for file, "sproduct"

Commands

[F5]	toggles between displaying field headings and displaying field lengths and edits to
[F6]	view a different file (asks for file name)
[PgDn] [PgUp]	to scroll down/up list of fields to print
[H]	headings, lengths, and edits on the default printer

If you want to see the field lengths and edits, press [F5], and you would see the following:

```
                    F I L E P R O   S Q L

------------------------------------------------------------------
1-    (8, *)
2-    (25, LOWUP)
3-    (8, .2)
4-    (5, .O)
5-    (5, .0)
6-    (lO, .2)
7-    (6, .0)




------------------------------------------------------------------
File: sproduct        Enter Selection >

         F5 - Field Lengths and Edits,  F6   Different File

   PGDN   PGUP - Scroll Fields,  H -  Hardcopy,  ↵  - Return

Enter File Name:
```

FPSQL screen (with Windows command keys) displaying field lengths and edits for file, "sproduct"

Pressing [F5] again toggles you back to the field headings.

If you want to see a different file, press [F6}. You will then see the prompt:

```
▶ Enter File Name:
  Press F10 for a list of filePro files
```

Enter the name of the file you want to see.

The hardcopy command, "H", will produce a printout from your printer which includes headings, lengths, and edits in one list, as shown on the next page:

```
Sep 3, 1987  14:10      File Name: sproduct
Page 1

      Number -------------Field Heading-------  Len     -
-Type--
 key segment:
          1  Product Number                      8
 *
          2  Description                        25
 LOWUP
          3  Price                               8
 .2
          4  Quantity on Hand                    5
 .0
          5  Reorder Point                       5
 .0
          6  Total Sales                        10
 .2
          7  Total Quantity                      6
 .0



Key segment record length: 87
There is no data segment.
```

FPSQL printout of filePro file headings, lengths, and edits
To return to the FPSQL main screen, press [Return].

Using online Help

FPSQL provides you with a full complement of online Help screens. All major topics are covered. All you have to do to access online Help is to go into the "Update" or "New" modes (press [U] or [N] while at the main screen). A new set of command options will be displayed:

▶ **F10 – Help, F6 - Display Fields,**
ESC - Record BREAK - Cancel

Press [F10] to see a list of Help subjects, as shown on the next page:

```
                        F I L E P R O    SQL

------------------------------------------------------------
      Editor
      SET
      SELECT
      FROM
      WHERE
      GROUP BY
      HAVING
      ORDER BY
      Comments
      Field
      Literal
      Aggregate
      Condition




What do you want help on? SET
```

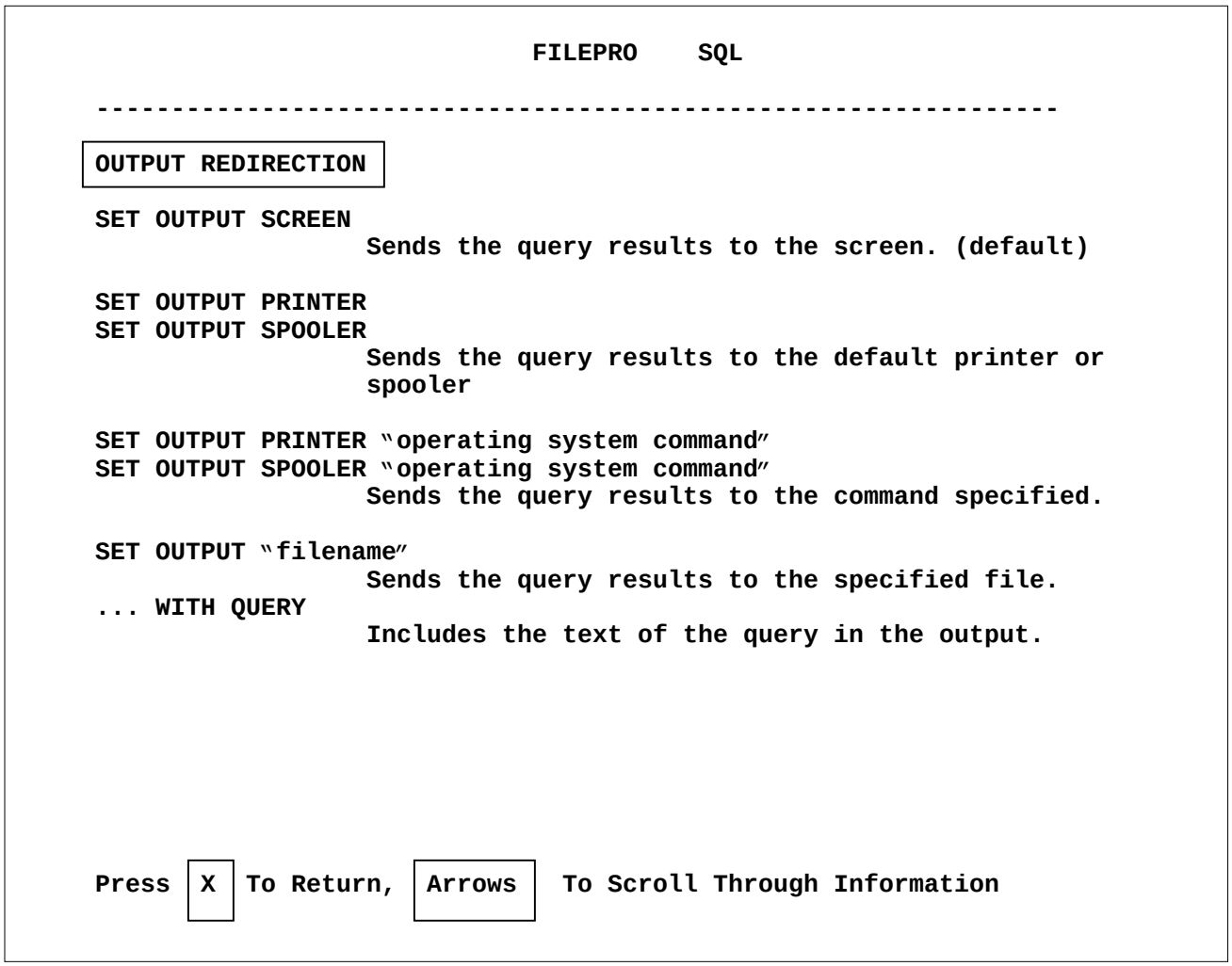
FPSQL Help subject screen

You can select a topic by any of three methods:

- Type in the complete subject name
- Type in the first few characters of the subject name until the highlight moves to the subject you want, then press [Return]
- Move the highlight to the subject you want with the arrow keys, [□] [□] [□] [↓], then press [Return]

To get back to the FPSQL main screen from here, press [Ctrl][Break] or [Break].

Having selected a topic, for example: “SET”, you will then see the first Help screen for the subject, as shown below:



FPSQL Help screen for "SET"

Pressing [X] will take you back to the Help subject screen (with the list of subjects). If there are multiple screens of information on the subject, press [↓] to see the next screen, and [ ] to see a previous screen. At the bottom of the last Help screen the prompts are:

▶ **Last Page. Press RETURN To Continue**

[Return] takes you back to the Help subject screen where you can press [Ctrl][Break] or [Break] to take you back to the FPSQL main screen.

CHAPTER 4:
THE FPSQL DEMO FILES

In this chapter:

The FPSQL demo files

All the queries shown in this manual use filePro demonstration files which are automatically installed in your "\filepro" directory when you install the FPSQL program. The presence of these files in your own system gives you the ability to try out the queries presented in the manual. You can try them as is, or experiment with modifications. Either way, they should make your learning FPSQL easier and more enjoyable.

The essential ingredients in composing a query, besides the clauses and reserved words, of course, are the file and field names. It also helps to know the type of data that a field contains (specified by the field edit) and the length of the field.

To help you understand the queries composed for the demo files, the field parameters from each file are presented here. This is the same information you could print out yourself from within FPSQL (see "viewing field headings, lengths, and edits" in the previous chapter).

“sproduct” demo file

Number	Field Heading	Len	Type
key segment:			
1	Product Number	8	*
2	Description	25	LOWUP
3	Price	8	.2
4	Quantity on Hand	5	.0
5	Reorder Point	5	.0
6	Total Sales	10	.2
7	Total Quantity	6	.0

“sclient” demo file

Number	Field Heading	Len	Type
key segment:			
1	Client Number	6	*
2	Name	25	UPLOW
3	Address	25	LOWUP
4	City	16	UPLOW
5	State	2	*
6	Zip	5	*
7	Phone Number	14	*
8	Tax Rate	4	.2
9	Company Name	20	LOWUP
10	Total Sales	10	.2
11	Balance Due	10	.2

“sinvoice” demo file

Number	Field Heading	Len	Type
key segment:			
1	Customer Number	6	*
2	Invoice Number	5	.0
3	Invoice Date	8	MDY/
4	salesman	10	UPLOW
5	A1) Items	8	*
6	A1)	8	*
7	A1)	8	*
8	A1)	8	*
9	A2) Quantities	5	.0
10	A2)	5	.0
11	A2)	5	.0
12	A2)	5	.0
13	A3) Line Item Totals	9	.2
14	A3)	9	.2
15	A3)	9	.2
16	A3)	9	.2
17	Subtotal	10	.2
18	Tax Amount	8	.2
19	Shipping	10	.2
20	Payment	10	.2
21	Balance Due	10	.2

“ssalesmn” demo file

Number	Field Heading	Len	Type
key segment:			
1	Name	10	UPLOW
2	Territory	3	ALLUP
3	Manager	10	UPLOW
4	Sales	10	.2
5	Commision	8	.2
6	Comm Rate	5	.2
7	Salary	8	.2
8	Hired Date	8	MDY/

CHAPTER 5:
"SELECT" AND "FROM" CLAUSES

In this chapter:

- The "select" clause
- Typing in field names
- The "from" clause
- Typing in file names
- Field names - what can be a field
- File names - what can be a file
- Quiz query #1

SET (options)
SELECT (fields)
FROM (files)
WHERE (conditions)
GROUP BY (fields)
HAVING (conditions)
ORDER BY (fields)

The “select” clause

The “select” clause is required in every query statement. It tells FPSQL what data is to be displayed. The data can be real fields or the results of arithmetic operations on fields. For example, you can tell FPSQL to add the “commission” fields to the “salary” fields to create, in the query output, brand new fields containing the sum. In fact, as you will learn further on, you can even tell FPSQL apply the heading, “earnings”, to this new field when it displays the query output. ***Please Note: FPSQL does not support 5.0 Blob or Memo Fields***

Typing in field names

When typing in field names, put them after the word “select” and separate them with commas (,). Put them in the order in which you want them to be displayed in the query output. The number of fields you may use is virtually unlimited.

Example: ***SELECT field, field, field, ...***

The “from” clause

The “from” clause is also required in every query statement. It tells FPSQL which files are to be accessed for the query. The fields named in the “select” clause must be in the file(s) named in the “from” clause.

Typing in file names

When typing in the file names, put them after the word “from” and separate them with commas (,). Their order in the query statement is not important. The number of files you may use is virtually unlimited.

Example: ***FROM file, file, file, ...***

As you will see in the next section, a field used in the "select" clause, as well as in other clauses in FPSQL, can be much more than just the name of a real field in a filePro file.

Field names - what can be a field

A real field

A field used in the "select" clause can be a real field.

Example:

select product_number, description, price
from sproduct

```
product_number     description         price
-----------------------------------------
01-2345            Soldering Iron      14.00
90-1234            Car Antenna         26.45
89-0123            Lite Dimmer          9.35
***  QUERY OUTPUT TRUNCATED ***
```

NOTE: In several cases in this manual, the output of example queries has been truncated. This was done in the interests of brevity, and only when the complete output is not needed to illustrate the point.

Multiple-word field names

To select a field name that has more than one word, you substitute an underscore (_) for the space between words, as in "**product_number**". above.

Field name order

As mentioned before, you can put the field names in any order you wish to see them displayed in the output, as in the following example.

Example: ***select price, description, product_number***
from sproduct

```
 Price  description        Product_Number
 ---------------------------------------
 14.00  Soldering Iron             01-2345
 26.45  Car Antenna                90-1234
  9.35  Lite Dimmer                89-0123
***  QUERY OUTPUT TRUNCATED ***
```

Partial Name of real fields

You don't have to type in all the words in a multi-word field name. You can use the first word of the field name, truncating it in the same place you would use the underscore, as described previously. In this example you would use "product" instead of "product_number", and get the same results.

Example:

select product, description, price
from sproduct

```
product_number     description        price
-----------------------------------------
01-2345            Soldering Iron      14.00
90-1234            Car Antenna         26.45
89-0123            Lite Dimmer          9.35
***  QUERY OUTPUT TRUNCATED ***
```

NOTE: Make sure there are no other fields with the same first words in their names as the one you use. When executing the query, FPSQL grabs the first field name it finds that matches.

Substituting your own column headings

If you want the resultant field in the query output to have a heading which more concisely describes its content, such as "earnings" instead of "commission+salary*9. for example, you can tell FPSQL to use the alias, "earnings" as the column heading in the output.

Example: ***select name, commission+salary*9 'earnings'***
from ssalesmn

```
name       earnings
--------------------
Macmillan  18055.38
Winston    10927.03
Mifflin    16200.00
*** QUERY OUTPUT TRUNCATED ***
```

A real field referenced by its number

Example:

select @1, @2, @3
from sproduct

```
product_number     description        price
-----------------------------------------
01-2345            Soldering Iron     14.00
90-1234            Car Antenna        26.45
89-0123            Lite Dimmer         9.35
```

A system-maintained field

Example: ***select @RN, @CD, @TD***

An associated field group

Example:

select @Al, @A2, @A3
from sinvoice

```
Al)  Items  A2) Quantities  A3) Line Item Totals
-----------------------------------------------
01-2345              10                   140.00
90-1234               5                   132.25
89-0123               3                    28.05
***  QUERY OUTPUT TRUNCATED ***
```

Associated Fields

Associated fields are fields which are grouped together in a record because they contain the same kind of information. Associated fields enable you to make multiple entries in the same field in a record.

The demo file, "sinvoice" (see Chapter 4) contains three associated field groups: @Al "Items", @A2 "Quantities", and @A3 "Line Item Totals".
Each associated field group in the sinvoice file consists of four "subfields", allowing each record to contain four "Items", four "Quantities", and four "Line Item Totals". You can have as many as 10 associated field groups with 16 subfields in each. (see your filePro Reference manual for more about associated fields.)

When you use an associated field in a query, FPSQL uses all the subfields in computing the answer.

Fields preceded by a file name, as in "filename.field".

Example:

select sclient.address, invoice_number
from sclient, sinvoice

Use this form when you are querying two or more files, such as "sclient" and "sinvoice", which contain the identical field name, "address". The name**,** "sclient.address" tells FPSQL precisely which file and field to access. (See Chapter 11, "Joining Files".)

An asterisk, meaning "all fields"

Example:

select * from ssalesmn

```
Name     Territory Manager     Sales    Commission Comm Rate Salary     Hired Date
-------------------------------------------------------------------------------
Macmillan NYC      Knopf         692.30    5538.40  8.00  2000.00        01/15/80
Winston   NYC      Knopf        2822.79   12702.56  4.50  1200.00        06/01/87
Mifflin   NYC      Knopf                            7.20  1800.00        02/10/82
Hall      PA       Jovanovich   3746.83   26227.81  7.00  1700.00        11/20/83
Wiley     OH       Jovanovich    794.78    5166.07  6.50  1500.00        08/12/85
McGraw    NYC      Knopf                            7.50  1850.00        03/13/81
Reinhart  NYS      Knopf                            6.70  1600.00        10/17/84
Holt      NJ       Jovanovich    311.05    1866.30  6.00  1400.00        03/14/86
Haughton  NYS      Knopf                            5.00  1300.00        11/11/86
```

NOTE: When you use "*" to select all fields, they are displayed in the order in which they were defined in the file.

An aggregate function applied to a field

(Aggregate functions, MIN, MAX, AVG, SUM, and COUNT are described in Chapter 7)

Example: ***select max(price), count(*)***

The "mid" function

(The "mid" function in FPSQL is similar to the "mid" function in filePro processing. In FPSQL it gives you the ability to create, for purposes of querying data, a field within a field, such as the month in an MDY field. The "mid" function requires the use of real fields only. It can be used in the "select'' and "order by" clauses. (See Chapter 10, "Order by Clause".)

"Mid" specifies a character position and a character range in the field's character string.

Example: ***select mid(invoice_date,l,2)***

In the example above, the "**1**" means the first character in the string. The "**2**" means two characters, starting with the first. In an 8-character MDY/ field, the specified characters, "1,2" would be the first two, i.e., the month. To select the year in an MDY/ field, specify the seventh character and a range of two characters:

Example: List just the years of the dates-of-hire for the salesmen.

select mid(hired_date,7,2)
from ssalesmn

```
mid(hired_date,7,2,)
--------------------
80
87
82
*** QUERY OUTPUT TRUNCATED ***
```

Example: List the first initials of the names of the salesmen.

select mid(name,1,1)
from ssalesmn

```
mid(name, 1, 1)
---------------
M
W
M
*** QUERY OUTPUT TRUNCATED ***
```

A text literal

Example: ***select 'text'***

A numeric literal

Example: ***select 19.95***

NOTE: Text literals are enclosed in single quotes(''). Numeric literals are not.

Literals are primarily useful in the "where" clause. However, both numeric and text literals can be used in the "select" clause. If they are, they will appear in the output as column headings, and they will take up column space.

Example:

select name, sales, ' (write comments here) '
from ssalesmn

```
Name       Sales          (write comments here)
           -------------------------------------------
           Macmillan    692.30
           Winston     2822.79
           *** QUERY OUTPUT TRUNCATED ***
```

An expression using fields

(Expressions are covered in Chapter 6)

Example: Display the salesmen's names, their commissions plus the total (monthly) salary for the first nine months of the year.

select name, commission+salary*9
from ssalesmn

```
name        commission+salary*9
--------------------------------
Macmillan                18055.38
Winston                  10927.03
Mifflin                  16200.00
Hall                     15562.28
***  QUERY OUTPUT TRUNCATED ***
```

"commission+salary*9" is the expression. The plus sign(+) is for addition, the asterisk (*) is for multiplication. These and other "arithmetic operators" are covered in Chapter 6.

How fields are displayed in output

The width of a field as displayed in query' output is determined by one of the following:

1. The length of the field name or alias used in the "select" clause
2. The length of the field itself, if it is longer than the field name

Also, FPSQL inserts a single blank space between output fields.

You can place literals full of blanks between field names in your "select" clause to spread the output fields apart. You can use aliases in place of long field names in your "select" clause to narrow the output fields.

How field names are justified in output

Names of fields with numerical values are right-justified when displayed in output. All other field names are left-justified.

File names - what can be a file

Files are the object of the "from" clause, and they can be used in two ways.

A filePro file

Example: ***from ssalesmn***

A filePro file with an "alias"

As you will see later in this manual (Chapter 11, "Joining Files"), some queries require that you access the same file more than once in the query. To do this you use an "alias" as follows:

Examples: ***from ssalesmn territory, ssalesmn manager***
from ssalesmn x, ssalesmn y

The file is "ssalesmn", the aliases are "territory" and "manager", or "x" and "y" You can use any name for an alias. What's important is that you use it consistently throughout the query statement whenever you refer to the file name.

A filePro qualified file

To execute a query on a qualified file, you must first use the "set" clause (see Chapter 12) to give FPSQL the qualifier name.

Quiz query #1 **(answer in Appendix E)**

See if you can compose a query statement to accomplish the following:

Produce a list of salesmen showing their dates of hire, commissions, and salaries, in that order.

(refer to Chapter 4 if you need to see demo files and field headings, or use the online file directories; their use is covered in Chapter 3)

CHAPTER 6:
THE “WHERE” CLAUSE

In this chapter:

- The “where” clause
- Condition - general form
- Comparison - general form
- Literals
- Comparison operators
- Grouping comparisons
- Examples using comparison operators
- Expressions
- Quiz query #2

SET (options)
SELECT (fields)
FROM (files)
WHERE (conditions)
GROUP BY (fields)
HAVING (conditions)
ORDER BY (fields)

The “Where” Clause

As we know, the “select” clause specifies the fields we want to display. However, “select” (and “from”) used without any other clauses retrieves the specified fields from ALL the records. To really unleash the analytical power of FPSQL, you need to be able to extract the fields from just the records you want. This is what the “where” clause does. It tells FPSQL exactly which fields, out of all of them, to display by specifying the conditions the fields must meet in order to be selected.

The following query retrieves the fields specified from all the records.

Example:

select product_number, description, price
from sproduct

```
product_number  description           price
--------------------------------------------
01-2345         Soldering Iron        14.00
90-1234         Car Antenna           26.45
89-0123         Lite Dimmer            9.35
78-9012         Barometer Kit        199.60
67-8901         Humidity Meter        99.50
56-7890         Thermo Reader         12.00
45-6789         DigiClock             12.00
34-5678         Battery Pak           15.10
23-4567         Metal Tracker        219.00
12-3456         Touch Button Phone    29.99
```

Let's say you're interested in items with high prices - greater than $99, for example. You would then type in the query with a "where" clause specifying that the condition to be met is that the price field contain a number greater than 99. FPSQL then retrieves the specified fields from all the records whose price field contains a number greater than 99.

Example:

select product_number, description, price
from sproduct where price > 99

```
product_number  description             price
-------------------------------------------
78-9012         Barometer Kit          199.60
67-8901         Humidity Meter          99.50
23-4567         Metal Tracker          219.00
```

Note the use of the "greater than" symbol (>). This and other "comparison operators" are covered later on in this chapter.

Condition - general form

The general form of a condition is:

comparison op comparison op comparison...

A "comparison" is two things compared. In the previous example, "price" is compared with "99". They would make a comparison. "Op" is a boolean operator, either "and", or "or". You can use as many comparisons as you wish.

Example: ***where hired > '01/01/80' and salary < 1800 and territory = 'nys'***

Comparison - general form

The general form of a comparison is:

field rel field...

"Field" can be any of the kinds of fields used in the "select" clause (see the previous chapter), except the aggregate functions, which are not allowed in the "where" clause. (Aggregate functions are covered in the next chapter.)

"Field" may also be a "subquery" (see Chapter 9).

"Field" may also be a "literal" (see next section).

"Rel" is the relation: < <= = >= > <> (see "comparison operators" following "literals", on the next page).

Literals

In comparisons, you may want to compare a field name to a character string or a number.

Examples: ***where name = 'Holt'***
where price > 99

A character string used in a comparison is a "text literal". A number may be a "text literal" or a "numeric literal". If the number is used to identify rather than quantify, e.g., zip codes, catalog numbers, invoice numbers, it is a text literal. If the number is used as a numerical value, e.g., price, quantity, sales, commission rate, it is a numeric literal.

Example of text literals: ***where territory = 'nys'***
and product = '89-0123'

Note the use of single quotes to enclose the text literals.

Example of numeric literals: ***where price = 29.99***
and quantity on hand = -12

As shown in the example, above, FPSQL allows you to use negative numbers as numeric literals.

Comparison operators

These define the relation between fields in a comparison:

Operator	Relation
<	field is less than field
<=	field is less than or equal to field
=	field is equal to field
>=	field is greater than or equal to field
>	field is greater than field
<>	field is not equal to field

Note: When using the above operators with dates, “greater than” means “later than”. “Less than” means “earlier than”.

Note: When comparing a field name to the text contents of a field, a “text literal”, you must determine if there are any empty spaces in the field preceding the text string. The empty spaces must be added to the literal between the first single quote and the first character of the literal:

Example: ***where product_number = ‘ 89-0123’***

Note: FPSQL does not require the complete text literal to make a comparison. However, the more characters you include in literal, the closer the match.

Example:
```
select name from ssalesmn
where name = ‘H’

name
-----------
Hall
Haughton
Holt
```

Example:

```
select name from ssalesmn
where name = 'Ha'

name
-----------
Hall
Haughton
```

Example:

```
select name from ssalesmn
where name = 'Hau'

name
-----------
Haughton
```

IS BETWEEN field A is between field B and field C

Note: same as field A >= field B and field A <= field C

Example: ***where zip is between 10036 and 10500***

IS NOT BETWEEN field A is not between field B and field C

IS IN field A is in (field B, field C, field D, . . .)

Note: same as field A = field B or field A = field C or field A = field D

Sets of fields used with "is in" require inclusion within parentheses, "( )".

Example: ***where territory is in ('nj', 'pa', 'oh')***

IS NOT IN field A is not in (field B, field C, field D, . . .)

IS LIKE field is like ‘text literal’

IS NOT LIKE field is not like ‘text literal’

Note: The “is like” comparison operator allows “wild card” matching of individual characters and strings:

Use the underscore (_) to represent any single character.

Example: ***where territory is like ‘ny_’***

Use the percent sign (%) to represent any string of zero or more characters.

Example: ***where company name is like ‘first%’***

“is like ‘%text%’“ is the same as filePro’s Contains (CO) relation.

IS NULL field = ““

IS NOT NULL field <> “”

NOTE: You **cannot** use the identical system-maintained fields from different files, as in the following.

Example: ***where customer.@rn = cust2.@rn***

Accessing different files in the same query is covered in Chapter 11.

Grouping comparisons

Comparisons can be grouped by placing them within square brackets:

[comparison op comparison] op [comparison op comparison]

Example: ***where [salary < 1800 or comm_rate < 7.5] and not [territory = ‘ny_‘ or hired < ‘01/01/83’]***

Examples using comparison operators

Example: Let's say you want the name and telephone numbers of customer contacts from companies in New York State that owe you more than $1,000.

select balance_due, name, state, phone_number
from sclient

where state = 'ny' and balance_due >= 1000

```
balance_due name                 state  phone_number
-------------------------------------------------------
2655.32     Bill Smith           NY     (212) 555-1223
1111.34     G Can                NY     (914) 555-4040
```

Example: Let's say someone left you a poorly written phone message and you can't make out all the letters of the name of the person who called, or all the numbers in the phone number. You can make out that it's a customer contact from one of your clients. The letters of what you can make out are "e gel a". You want to know the person's name and phone number.

select name, phone_number
from sclient
where name is like '%e_gel_a%' and
phone_number is like '%555-3_1_'

```
name                  phone_number
-----------------------------------------------
Ellen Eigelvar        (613) 555-3218
```

Example: You want to see a list of companies and their balances due if the balances due are between $300 and $1,000. Included are contact names and phone numbers.

select company, balance, name, phone
from sclient
where balance is between 300 and 1000

```
company                        balance name           phone
-------------------------------------------------------------------
Pooh Enterprises                915.41 Zeb Wellman    (212) 555-4456
Bingo Enterprises               648.32 John Bingo     (201) 555-9876
Canary Card Co                  400.00 Carl Bird      (212) 555-6565
First National Corp.            598.92 Ellen Eigelvar (613) 555-3216
Boomer Instruments              335.49 John Boomer    (412) 555-0971
Mallard Industries              371.69 Ed Murphy      (514) 323-3233
```

Example: You want to see a list of companies (with cities, states, and phone

numbers) located in Pennsylvania, Ohio, and New Jersey only.

select company_name, city, state, phone_number
from sclient
where state is in ('PA', 'OH', 'NJ')

```
Company_Name          City         State   Phone_Number
-------------------------------------------------------------
Bingo Enterprises     Bingolo Blvd   NJ      (201) 555-9876
First National Corp.  Cincinnati     OH      (613) 555-3218
Boomer Instruments    Pittsburgh     PA      (412) 555-0971
```

Expressions

Unlike comparisons, expressions do not compare fields. They combine fields (containing numeric values), arithmetic operators, and numbers to derive and express a resultant value.

Example: ***where payment < (subtotal+tax_amount)*.25***

In the example, "(subtotal+tax_amount)*.25" is the expression. It represents the number derived by adding together the values in the "subtotal" and "tax amount" fields and then multiplying the sum by .25.

Arithmetic operators

Operator	Function
+	Add
-	Subtract
*	Multiply
/	Divide

Expressions can only use fields containing numeric values, and dates, if appropriate.

Example: **`where @TD >= invoice_date+60`**

When calculating expressions, FPSQL performs multiplication and division before addition and subtraction. Use parentheses to group the elements of the expression so that the calculations are performed as desired.

Example:

select name. sales-commission*9
from ssalesmn

```
name            sales-commission*9
-------------------------------------
Macmillan                    193.88
Winston                     1679.52
***  QUERY OUTPUT TRUNCATED ***
```

Example:

select name, (sales-commission)*9
from ssalesmn

```
name            (sales-commission)*9
-------------------------------------
Macmillan                    5732.28
Winston                     24261.84
***  QUERY OUTPUT TRUNCATED ***
```

The "Exists" Subclause

The "exists" subclause tests whether a condition exists or doesn't exist (is true or not true). "Exists" is always used in the "where" clause, and the condition it tests is expressed in a subquery (see Chapter 9. "Subqueries and Variable Queries").

If the subquery generates output (the condition is true), the outer query will be executed. If the subquery generates no output (the condition is not true), the outer query will not be executed.

Example:

```
select *                #this is the "outer" query
from ssalesmn
where exists

                        select name  #this is the "subquery"
                        from salesmn
                        where sales >2800
```

```
Name        Territory Manager   Sales     Commission Comm_Rate Salary     Hired_Date
-------------------------------------------------------------------------------

Macmillan  NYC      Knopf        692.30    5538.40      8.00   2000.00    01/15/80
Winston    NYC      Knopf       2822.79   12702.56      4.50 1200.00      08/01/87
Mifflin    NYC      Knopf                              7.20 1800.00      02/10/82
Hall       PA       Jovanovich  3746.83   26227.81      7.00 1700.O0      11/20/83
Wiley      OH       Jovanovich   794.78    5166.07      6.50 1500.00      08/12/85
McGraw     NYC      Knopf                              7.50 1850.00      03/13/81
Reinhart   NYS      Knopf                              6.70 1600.00      10/17/84
Holt       NJ       Jovanovich   311.05    1866.30      6.00 1400.00      03/14/86
Haughton   NYS      Knopf                              5.00 1300.00      11/11/86
```

NOTE: When you use "*" to select all fields, they are displayed in the order in which they were defined in the file.

This query displays the entire contents of the salesmn file because the condition in the subquery is true; at least one of the salesmen has sales greater than $2800 (the subquery generates output). If no salesman had sales greater than $2800 (the subquery generated no output), there would be no output for this query.

Quiz query #2 **(answer in Appendix E)**

From the invoice file, you want to see a list of companies (with the sum of their subtotals and tax amounts, their payments, invoice numbers) whose payments are greater than 25% of the amounts owed.

CHAPTER 7:
AGGREGATE FUNCTIONS
AND THE "GROUP BY" CLAUSE

In this chapter:

- Aggregate functions
- "Group by" clause
- Quiz query #3

SET (options)
SELECT (fields)
FROM (files)
WHERE (conditions)
GROUP BY (fields)
HAVING (conditions)
ORDER BY (fields)

Aggregate functions

The aggregate functions are:

"min"	minimum
"max"	maximum
"avg"	average
"sum"	sum
"count"	count (number of records)

"Avg" and "sum" can only be applied to fields with numeric values.

Example:

select sum(salary)
from ssalesman

```
sum(salary)
-----------
   14350.00
```

"Min" and "max" can be applied to any type of field. With dates, "min" is earliest, "max" is latest.

Example:

select min(hired)
from ssalesman

```
min(hired)
----------
 01/15/80
```

"Count" counts the number of records in the query output.

Example:

select count(*)
from sproduct

```
count (*)
-----------
       10
```

The query output shows that there are 10 records in sproduct.

Example:

select manager, count(*)
from ssalesmn
where manager = 'knopf'

```
manager   count (*)
------------------
knopf              6
```

Manager Knopf has six salesmen reporting to him.

Example:

select count(*), avg(sales), avg(commission),
avg(sales)-avg(commission), avg(salary)*9
from ssalesmn

```
count(*) avg(sales) avg(commission) avg(sales) –avg(commisslon) avg(salary)*9
-------------------------------------------------------------------------
       9   929.7500         57.2233                   872.5267    14350.0000
```

Queries using aggregate functions, except "count", should be composed to include only fields that share some common characteristic. It makes no sense to include salesmen's name with average salary, sales, and commission because each record has a different salesman's name.

Example:

select name, avg(sales), avg(commission),
avg(sales)-avg(commission) avg(salary)*9
from ssalesmn

```
name    avg(sales) avg(commission) avg(sales)-avg(commission) avg(salary)*9
------------------------------------------------------------------------
Haughton  929.7500         57.2233                  872.5267    14350.0000
```

The values in the output above, are the averages for all the salesmen in the file, not for salesman Haughton. Finding these same kinds of averages for a manager is meaningful because there are more than one record which have the same manager.

Example:

select manager, avg(sales), avg(commission),
avg(sales)-avg(commission), avg(salary)*9
from ssalesmn where manager = 'knopf'

```
manager   avg(sales) avg(commission)      avg(sales)-avg(commission) avg(salary)*9
-------------------------------------------------------------------------------
Knopf       585.8483         30.4017                      555.4467     14625.0000
```

As you will see later in this chapter, you can find the averages for both managers in one query by using the "group by" clause. Without the "group by" clause aggregate functions apply to the entire set of selected records.

Aggregate functions are used in the "select" and "having" clauses (see Chapter 8, "Having Clause"), and commonly used with the "group by" clause, but not used in the "where" clause. However, you can usually achieve the same effect in the "where" clause by using a "subquery" (see Chapter 9, "Subqueries and Variable Queries".)

“Group by” clause

The “group by” clause enables you to perform aggregate functions on groups of records. The effect of the “group by” clause is to first group the records by some shared characteristic and then perform the aggregate functions for each group.

In the following example the records are grouped by manager, creating two groups (one for each manager). In each group the records all have the same manager (the “shared characteristic”). The aggregate functions are then performed for each group.

Example: For each manager, find the average sales, average commission, average sales minus average commission and average salary through September (9 months)

select manager, avg(sales), avg(commission),
avg(sales)-avg(commission), avg(salary)*9
from ssalesmn
group by manager

```
       manager     avg(sales) avg(commlssion) avg(sales)-avg(commission) avg(salary)*9
-------------------------------------------------------------------------------
Jovanovich    167.5533        110.8867                 1506.6867       13800.0000
       Knopf          585.8483         30.4017                   555.4467    14625.0000
```

You may use more than one field in the “group by” clause. The records grouped by the first field named will be subdivided and grouped by the second field named, and so on. The aggregate functions are performed for each of the subdivisions.

Example:

select manager, territory, avg(salary)
from ssalesmn
group by manager, territory

```
manager     territory avg(salary)
-----------------------------------
Jovanovich  NJ          1400.0000
Jovanovich  OH          1500.0000
Jovanovich  PA          1700.0000
Knopf       NYC         1712.5000
Knopf       NYS         1450.0000
```

Quiz query #3 **(answer in Appendix E)**

You want to find out, for each manager, the average salary, number of salesmen, and the earliest date of hire - from the "ssalesmn" file.

CHAPTER 8:
"HAVING" CLAUSE

In this chapter:

- The "having" clause
- When used with the "where" clause
- Quiz query #4

SET (options)
SELECT (fields)
FROM (files)
WHERE (conditions)
GROUP BY (fields)
HAVING (conditions)
ORDER BY (fields)

The "having" clause

The "having" clause applies to groups formed with the "group by" clause. It can only be used with the "group by" clause. It enables you to specify the conditions to be met in selecting the groups to output.

The conditions you can specify with the "having" clause are the same as for the "where" clause (see Chapter 6) except that you have the added capability of using the aggregate functions (see Chapter 7).

Example: Show the territories, and their managers, whose salesmen's average salary is between 1000 and 1600.

select territory, avg(salary), manager
from ssalesmn
group by territory
having avg(salary) between 1000 and 1600

```
territory avg(salary)  manager
---------------------------------
NJ          1400.0000  Jovanovich
NYS         1450.0000  Knopf
OH          1500.0000  Jovanovich
```

NOTE: In the "ssalesmn" file all the salesmen in a given territory have the same manager, otherwise, this query wouldn't work.

When used with the "where" clause

When the "having" clause is used with the "where" clause the query is evaluated as follows:

1. "Where" clause is applied to select records.

2. Groups specified by the "group by" clause are formed.

3. "Having" clause is applied to further select the groups.

Example: Find the territories (with their managers), in the NY/NJ area, in which the salesmen's average salary is between $l,000 and $1,600.

select territory, avg(salary) , manager
from ssalesmn
where territory is like 'N%'
group by territory
having avg(salary) between 1000 and 1600

```
territory avg(salary) manager
--------------------------------
NJ          1400.0000 Jovanovich
NYS         1450.0000 Knopf
```

Quiz query #4 (answer in Appendix E)

From the ssalesmn file, show the territories, with salesmen's average salaries, that have exactly two salesmen in them.

CHAPTER 9:
SUBQUERIES AND VARIABLE QUERIES

In this chapter:

- Subqueries
- Multiple levels of nesting
- Variable queries
- Quiz query #5

Subqueries

A subquery is a query within a query. It enables you to use the results of the "inner" query (subquery) as input to the "outer" query - all in the same query statement. Suppose, for example, you wanted to find the name of the salesman with the highest earnings through September (the 9th month). Using what you have learned so far, you might try:

select name, max(commission+salary*9)
from ssalesmn

```
name       max(commission+salary*9)
-----------------------------------
Haughton                   18055.38
```

FPSQL supplies a name. and the maximum of the sums of commissions plus salaries multiplied by 9, just as you requested. However, this query doesn't give FPSQL a way to connect a name with the amount, so it displays any name from the name field. Actually, the name of the salesman who makes the maximum amount is Macmillan, not Haughton.

You might try using two distinct queries to find the answer. First, you would find the maximum amount:

select max(commission+salary*9)
from ssalesmn

```
max(commission+salary*9)
------------------------
                18055.38
```

Once you had this amount, you would then compose and execute another query to find the name that goes with the amount:

select name
from ssalesmn
where commission+salary*9 = 18055.38

```
name
-----------
Macmillan
```

By nesting a sub-query within a query you can accomplish the above in one query statement:

```
select name, commission+salary*9
from ssalesmn
where commission+salary*9 =
                                    select max(commission+salary*9)
                                    from ssalesmn
```

```
name        commission+salary*9
------------------------------
Macmillan                18055.38
```

This query works by first evaluating the "inner" query, ("select max(commission+salary*9)..."") to get a value for "commission+salary*9". The value is then used as a condition in the "where" clause of the "outer" query ("where commission+salary*9 = ...') to select the name of the salesman earning that value.

Multiple levels of nesting

Queries can be nested to any number of levels.

Example: Who is the salesperson in the NY/NJ area who has the second highest sales?

```
select name, territory, sales
from ssalesmn
where sales =

        select max(sales)
        from ssalesmn
        where [territory is like 'NY%'
        or
                territory is like 'NJ%']
        and
        [sales <>

                        select max(sales)
                        from ssalesmn
                        where [territory is like 'NY%'
                        or
                                territory is like 'NJ%']
                ]
```

```
name      territory sales
--------------------------
Macmillan NYC       692.30
```

Working outward, the innermost query finds the highest sales in the NY/NJ territories. The next query outward does the same except that it compares its result to the result of the innermost query in a "not equal to" comparison ("sales <> select max(sales)..."). Since the result of the outer query is not allowed to be equal to the maximum sales (already selected by the inner query), it represents the next highest sales.

Note the use of square brackets to enclose the grouped comparisons.

If there are any more elements of a query following a subquery, a semicolon (;) is used to mark the end of the subquery. This enables FPSQL to differentiate between the end of a subquery and the rest of the query statement.

Example: List the product number, description, quantity-on hand for each product whose quantity-on-hand is less than the average quantity-on-hand. List the smaller quantities first.

```
select product, description, price, quantity
from sproduct
where quantity <
                    select avg(quantity)
                    from sproduct;
order by quantity
```

```
product   description           price quantity
-------------------------------------------
67-8901   Humidity Meter         99.50       24
34-5678   Battery Pak            15.10       26
12-3456   Touch Tone Phone       29.99       27
78-9012   Barometer Kit         199.60       58
56-7890   Thermo Reader          12.00       74
23-4567   Metal Tracker         219.00       76
90-1234   Car Antenna            26.45       82
```

NOTE: Without the semicolon, the "order by" clause would be applied to the inner query, and the outer query would not be sorted as in the following example:

Example:

```
select product, description, price, quantity
from sproduct
where quantity <
                    select avg(quantity)
                    from sproduct
order by quantity
```

```
product   description              price    quantity
-------------------------------------------------------
90-1234   Car Antenna               26.45          82
78-9012   Barometer Kit            199.60          58
67-8901   Humidity Meter            99.50          24
56-7890   Thermo Reader             12.00          74
34-5678   Battery Pak               15.10          26
23-4567   Metal Tracker            219.00          76
12-3456   Touch Tone Phone          29.99          27
```

Example: (See Chapter 10 for more information on the "order by" clause.)

Variable queries

Variable queries contain subqueries which are computed multiple times with different records, giving variable results which affect the outer query.

Suppose, for example, you were doing a comparative study of salary and commission levels and you wanted to know which salesmen made more than 25% of the total paid to everyone else with the same manager.

Example: Find the managers who have at least 3 salesmen and then list the name. salary+commission, for each salesman in the group whose salary+commission is more than 25% of the total of the remaining salesmen's salary+commission.

```
select name, salary+commission , manager
from ssalesmn x
where manager is in
            select manager
            from ssalesmn
            group by manager
            having count(*) >= 3;
```

```
and salary+commission >
            select sum(salary+commission) * .25
            from ssalesmn
            where ssalesmn.manager = x.manager and
            ssalesmn.name <> x.name
            group by manager

name        salary+commission manager
-------------------------------------
Macmillan         2055.38 Knopf
Hall              1962.28 Jovanovich
Wiley             1551.66 Jovanovich
Holt              1418.66 Jovanovich
```

This query works as follows:

1. It finds the managers who have at least three salesmen. ("where manager is in . . .")

2. For EACH salesman working for one of these managers it computes:

   a. the total of salary plus commission for everyone else with the same manager, leaving out the salary plus commission of the current salesman being computed. ("where ssalesmn.manager = x.manager and ssalesmn.name <> x.name")

   b. whether or not the current salesman earns more than 25% of this total. ("salary+commission > select sum(salary+commission)*.25 from ssalesmn... "). If he does, he is selected (by the outer query).

Quiz query #5 (answer in Appendix E)

List the territories, their salesmen's average salaries and managers, where the territories' average salary is greater than the average salary for all the salesmen.

CHAPTER 10
"ORDER BY" CLAUSE

In this chapter:

- The "order by" clause
- The sorting sequence
- Quiz query #6

SET (options)
SELECT (fields)
FROM (files)
WHERE (conditions)
GROUP BY (fields)
HAVING (conditions)
ORDER BY (fields)

The “order by” clause

The “order by” clause tells FPSQL which fields to sort the records by.

Example: ***order by name***
order by product_number
order by invoice_date

The fields you may use are the same as in the “select” clause (see Chapter 5), including aggregate functions (see Chapter 7).

If you don’t use an “order by” clause, FPSQL sorts the records by the fields specified in the “group by” clause, if any. If neither clause is used, the records are displayed in no specific order.

You **cannot** use the “order by” and “group by” clauses together in a query.

The sorting sequence

The default sorting sequence in the “order by” clause is “ascending”:

for text: A to Z
for numbers: 0 to 9
for alphanumeric: 0 to 9, then A to Z (ASCII order)
for dates: earlier to later

You can specify a “descending” sequence by adding the word, “desc” after the field for which you want the descending order.

Example: ***order by salary desc***

Example: List the product descriptions, numbers, and prices with the highest prices first.

select description, product_number, price
from sproduct
order by price desc

```
description                product_number     price
-------------------------------------------------
Metal Tracker               23-4567          219.00
Barometer Kit               78-9012          199.60
Humidity Meter              67-8901           99.50
***  QUERY OUTPUT TRUNCATED ***
```

You can sort by more than one field and have a different sorting sequence, ascending or descending, for each field.

Example: list the managers, their territories, salesmen's names, and salaries. Order by descending manager, ascending territory and descending name.

select manager, territory, name, salary
from ssalesmn
order by manager desc, territory, name desc

```
manager       territory       name          salary
-------------------------------------------------
Knopf         NYC             Winston      1200.00
Knopf         NYC             Mifflin      1800.00
Knopf         NYC             McGraw       1850.00
Knopf         NYC             Macmillan    2000.00
Knopf         NYS             Reinhart     1600.00
Knopf         NYS             Haughton     1300.00
Jovanovich    NJ              Holt         1400.00
Jovanovich    OH              Wiley        1500.00
Jovanovich    PA              Hall         1700.00
```

When you compose a query with more than one field in the "order by" clause, list the major sorting key first, followed by the minor keys, with the innermost key last. The order in which you put them in your query statement is the order in which FPSQL will sort them, as well as display them.

Quiz query #6 **(answer in Appendix E)**

Display the managers, territories, names, and dates of hire for all the salesmen. Order by manager, territory, and names.

CHAPTER 11:
JOINING FILES

In this chapter:

- Joining files
- Quiz query #7

Joining files

So far, all the queries we have looked at access a single file for the data. However, a query statement may specify fields from a virtually unlimited number of files. All you have to do is list them, in any order, in the "from" clause. FPSQL determines the most efficient method by which to perform the selection and qualification of the fields.

NOTE: Be sure that all the fields specified in the "select" clause are contained by the files specified in the "from" clause.

Example: ***from ssalesmn, sinvoice, sclient***

When two or more files use the same field name such as the "name" field in the "ssalesmn" and "sclient" files, you have to differentiate them by prefixing the file name with the field name, in the "select" clause.

Example: For each salesman find the customer_number and client name.
Order by salesman's name, customer_number.

```
select ssalesmn.name, customer_number, sclient.name
from sclient, ssalesmn, sinvoice
order by ssalesmn.name, customer_number
```

If we execute this query we get the following results.

```
ssalesmn.name customer_number sclient.name
-------------------------------------------------
Hall          1               Zeb Wellman
Hall          I               Bill Smith
Hall          1               John Bingo
Hall          1               Carl Bird
Hall          1               G Can
Hall          1               Ellen Eigelvar
Hall          1               John Boomer
Hall          1               John Jones
Hall          1               Ed Murphy
Hall          2               Zeb Wellman
Hall          2               Bill Smith
Hall          2               John Bingo
***  QUERY OUTPUT TRUNCATED ***
```

The complete output contains every possible combination of salesman's name, customer_number, and client name. This voluminous output is of little use. The real power of FPSQL in joining files is in the ability to qualify the results so that you can list the customer numbers and client names for each salesman. This is done by adding a "where" clause that selects, from all these possible combinations, only those that meet the conditions specified.

Example:

```
select ssalesmn.name, customer_number, sclient.name
from sclient, ssalesmn, sinvoice
where ssalesmn.name = sinvoice.salesman
and customer_number = client_number
order by ssalesmn.name, customer_number
```

```
ssalesmn.name customer_number sclient.name
-------------------------------------------------------
Hall          4               Ellen Eigelvar
Hall          7               Zeb Wellman
Hall          8               John Bingo
Hall          9               G Can
Holt          3               John Boomer
Macmillan     6               Bill Smith
Wiley         2               John Jones
Wiley         5               Carl Bird
Wiley         8               John Bingo
Winston       1               Ed Murphy
Winston       6               Bill Smith
```

By specifying that the name field in ssalesmn be the same as the "salesman" (name) field in "sinvoice" and that the sinvoice file's "customer_number" be the same as the sclient file's "client number", we are eliminating all the "possible" relations between these files by specifying the ones that really do exist. Each invoice has a salesman's name on it and each invoice has a customer number that is the same as a client number in the sclient file.

The following example contains multiple files and subqueries:

Example: For each sales manager show the total sales of the product with the maximum sales.

```
select manager, @Al, sum(@A3)
from ssalesmn, sinvoice
where
        salesman = name
and
        @al =
                select product
                from sproduct
                where total_sales =
                                        select max(total_sales)
                                        from sproduct;
                ;
group by manager
order by sum(@a3) desc
```

```
manager         @Al) Items                sum(@a3)
-----------------------------------------------
Knopf           78-9012                  1796.40
Jovanovich      78-9012                   998.00
```

Note the use of the semicolons (;) to mark the ends of both subqueries.

Quiz query #7 (answer in Appendix E)

List the products, their descriptions, product numbers, quantities, prices, total sales (quantities*prices) in each territory. Include salesmen's names. Sort by territory, salesperson and product number. Hint: you will need to use all the FPSQL demo files for this one.

CHAPTER 12:
"SET" CLAUSE

In this chapter:

- The "set" clause
- Redirecting query output
- Controlling printed output formatting
- Multiple "set" commands
- Quiz query #8

SET (options)
SELECT (fields)
FROM (files)
WHERE (conditions)
GROUP BY (fields)
HAVING (conditions)
ORDER BY (fields)

The “set” clause

The “set” clause must precede the “select” clause. The “set” clause enables you to redirect query output to printer, file, or screen, to control printed output formatting, and to enable querying qualified files. If a “set” clause in a query conflicts with the main screen command option “P” (to printer), the “set” clause overrides command “P”.

Redirecting query output

set output screen

Query output is sent to the screen when no other set clause is used. Even when the main screen command, *“P”,* is used to print out the query results, the results will still be displayed on your screen.

set output printer

set output spooler

You can send output to the default printer.

set output printer *‘operating system command’*

set output spooler ***‘****operating system command’*

Example: ***set output printer ‘lp -copies 2’***

Example: ***set output spooler ‘lp -copies 2’***

set output *'filename'*

You can send query output to a file in the current directory by specifying the file name. If you want to send it to another directory, use the path name. Use **single quotes** around the name in either case.

Example: ***set output '\wp\sales.jan'***

set output… with query

You can include the text of your query statement in the output by adding "with query" to the "set" clause This works with files as well as printers.

Example: ***set output printer with query***
set output '\wpksales.jan' with query

Controlling printed output formatting

set lines nn (page length)

You can set the output for the length of the paper in your printer by specifying the number of lines that will fit on a page. At the standard 6 lines per inch an 11" page will accommodate 66 lines from the top edge of the paper to the bottom edge. Without any additional numbers specifying the number of lines to actually print on a page (see next section) FPSQL will *print* on every line of the page.

Example: ***set lines 66***

set lines *mm,nn* (bottom margin**)**

If, as in the previous example, the number of lines on a page is set at 66, you can create a 6-line bottom margin by setting the number lines to be *printed* on a page at 60. (You can "split" the margin between top and bottom of the page by positioning the top edge of the paper three lines above the print head before you start to print.)

When typing the "set" clause, precede the number of lines on a page with the number of lines to be printed on a page; separate the numbers with a comma.

Example: ***set lines 60,66***

These settings can be used when sending output to a file, as well.

set lines 0 (zero) (to delete default heading)

Normally, the top of each page contains the field headings and a horizontal divider. You can delete this default heading from the output. This is especially useful if the output will be input for another program, such as a word processor. Simply set lines to zero.

set title
set title on (to include default title)

You can include a default title with the output. The default title is the date, time, and page number on the first line printed.

Example: ***set title on***

```
Sep 30, 1987 13:48                                    Page : 1
```

set title 'text' (your own title)

You can add your own title to the output. FPSQL inserts it between the date/time and page number..

Example: ***set title 'Sales for January'***

```
Sep 30, 1987 13:48      Sales for January      Page : 1
```

set title off (delete title)

No title will be printed (this is the default)

Default lines values

Output to the screen conforms to the height of the screen.

Output to the printer defaults to 60,66.

Output to a file defaults to 60,66.

Querying qualified files

set qualifier *qualifier name*

Only one qualified file can be used in a query.

Multiple "set" clauses

You can combine a single "set output..." with a single "set lines..." with a single "set title…" in any order, separated by a space or line return.

Example: ***set output printer set lines 72,84 set title***

You cannot, however, use more than one "set output…" or more than one "set lines…" or more than one "set title…" together. You cannot, for instance, set output to both printer and file in the same query.

Quiz query #8 **(answer in Appendix E)**

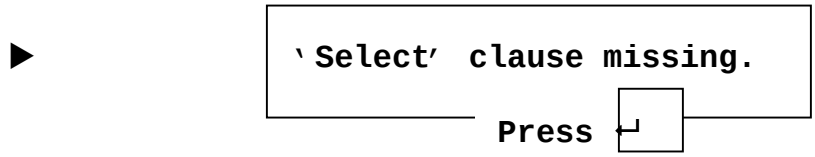
Use the "set" clause to send your query output to a file named "contract". Later, the contents of "contract" will be printed on legal size paper (14 inches long). Create a one inch bottom margin (lines per inch = 6). Give the output a title of "Assets Sorted By Location".

APPENDIX A:

ERROR MESSAGES

The error messages in this appendix are listed in alphabetical order. This is not a complete list; many of the error messages are so specific as to need no elaboration. Listed with each error message are the likely causes to look for.

Error messages appear in the middle of the query statement area in a small window:



The query will not be executed until the problem is corrected. To clear the error message from the screen, press [Return}. In many cases the cursor will be positioned in the query statement at the source of the problem.

Bad comparison.

Incorrect boolean operator symbols.

Error in aggregate function.

Open parenthesis, "(" missing.
Comma missing before, after "from"
"COUNT" used with something other than (*).

'From' clause missing.

A problem with text preceding the "from" clause.
Comma missing between fields.
Extra comma between fields and/or, after last field.

Mismatched brackets.

Close bracket, "]", missing, or extra bracket: "[" or "]"

Mismatched parentheses.

Close parenthesis, ")", missing, or extra parenthesis: "(" or ")"

No or invalid map: *'path/filename'***.**

File name misspelled.

Number of lines per page has already been set.

"Set lines..." used more than once.

Output has already been redirected.

"Set output..." used more than once.

'Select' clause missing.

"Select" spelled wrong.
A problem with text preceding "select".
"#" missing from commentary text.

Syntax error.

Incorrect order of reserved words, clauses.
Misspelling of reserved word.
Commas missing or extra commas present.
Quotes around text missing or extra quotes present.
Underscore missing from multiple-word field name.

Title has already been set.

"Set title…" used more than once.

Unknown field *'field name'***.**

Incorrect spelling of field name.
File name missing from "from" clause.

Unknown field *'filename.fieldname'***.**

Comma missing before or after file name in "from" clause.
Period missing in prefix.

APPENDIX B:

RESERVED WORDS

The following words have special meaning to FPSQL and cannot be used for file or field names. Queries using these words for file or field names will result in syntax errors.

and	group	not	select
asc	having	null	separator
avg	help	off	set
between	in	on	spooler
by	insert	or	start
count	into	order	sum
desc	is	output	title
edit	like	printer	unique
end	lines	query	unlock
exists	max	records	where
fields	mid	restart	width
from	min	screen	with

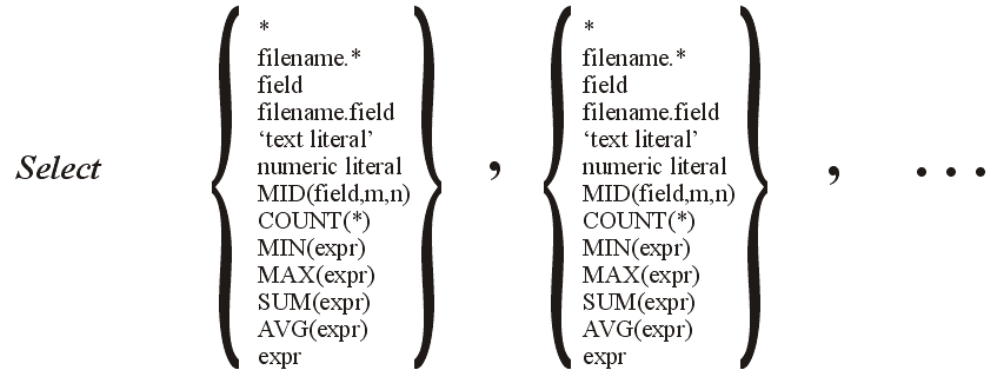
APPENDIX C:

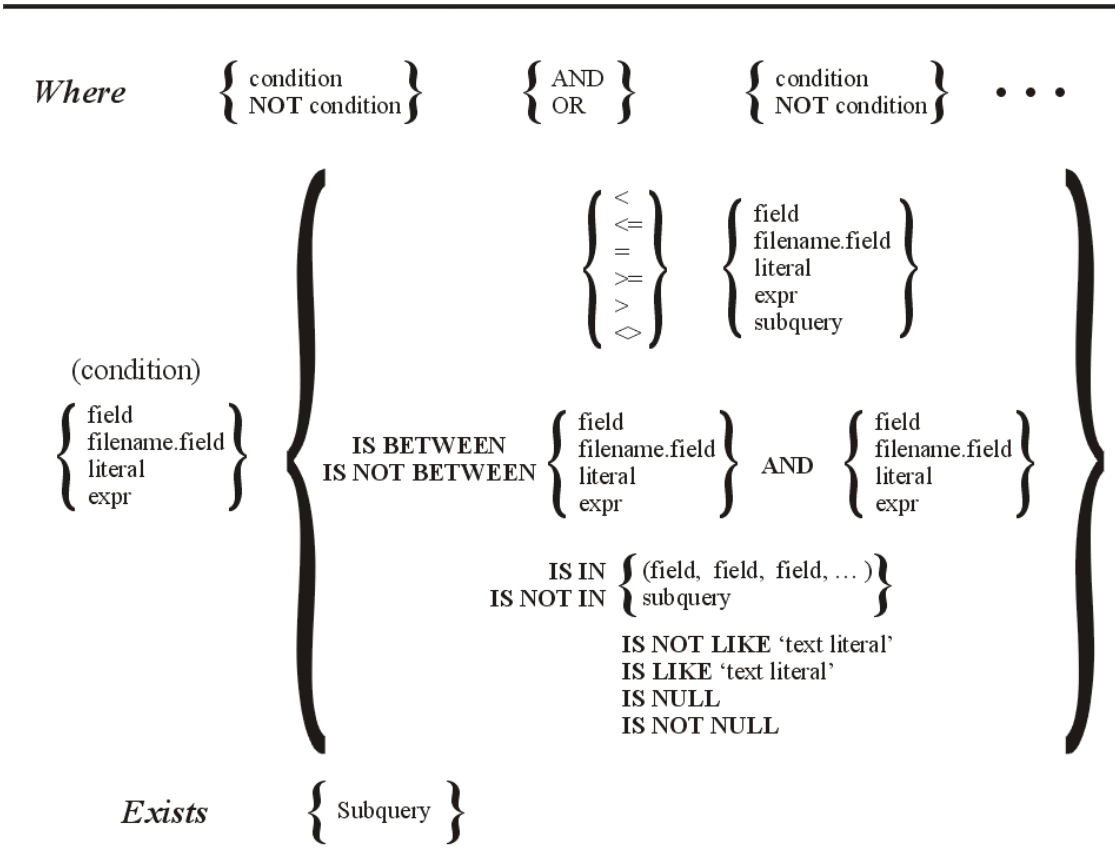
FPSQL SYNTAX DIAGRAMS

The following diagrams are also in the "FPSQL Quick Reference Guide" included in the FPSQL software package. The diagrams represent the syntax of all the clauses, except, "set", in a query statement.

Key to diagrams:

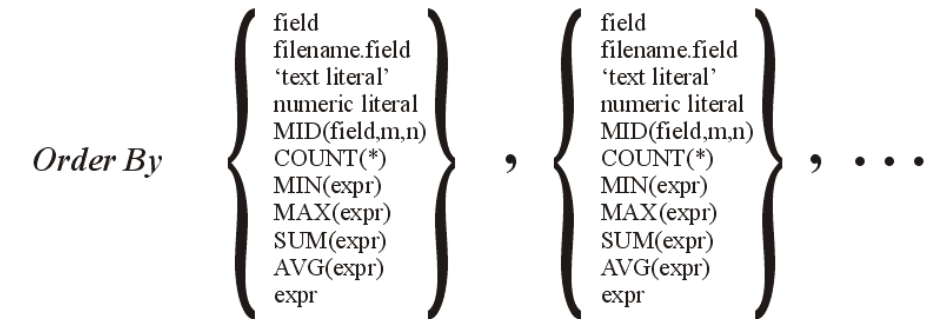
Large brackets { }	enclose a list of allowable entities:
	fields, files, numbers, expressions, aggregate functions, etc., for use in the clause.
Comma (,)	means that the following entity is optional.
Ellipsis (. . .)	means that preceding may be repeated any number of times.




Where
condition
NOT condition
AND
OR
condition
NOT condition
(condition)
field
filename.field
literal
expr
<
<=
=
>=
>
<>
field
filename.field
literal
expr
subquery
IS BETWEEN
IS NOT BETWEEN
field
filename.field
literal
expr
AND
field
filename.field
literal
expr
IS IN
IS NOT IN
(field, field, field, ...)
subquery
IS NOT LIKE 'text literal'
IS LIKE 'text literal'
IS NULL
IS NOT NULL
Exists
Subquery


Group By
field
filename.field
expr
,
field
filename.field
expr
, . . .


Having
condition
NOT condition
AND
OR
condition
NOT condition
. . .


Order By
field
filename.field
'text literal'
numeric literal
MID(field,m,n)
COUNT(*)
MIN(expr)
MAX(expr)
SUM(expr)
AVG(expr)
expr
,
field
filename.field
'text literal'
numeric literal
MID(field,m,n)
COUNT(*)
MIN(expr)
MAX(expr)
SUM(expr)
AVG(expr)
expr
, . . .

APPENDIX D:

FPSQL VERSUS ANSI STANDARD SQL

The following is a list of differences between FPSQL, version 5.0, and the American National Standards Institute (ANSI) X3.135-1986 SQL standard.

In FPSQL:

- The "select distinct" clause is not implemented.
- Password security is based on the creation password.
- There is no case sensitivity in sorts and comparisons (just like the rest of filePro).
- A "set" clause has been added.
- filePro's system-maintained fields can be used.

  Example: **@RN, @CD, @TD**
- filePro's additional field types can be used.

  Example: **MDY, HMS, $**
- Associated fields can be used.

  Example: **@A0, @Al, @A2**
- Fields can be referenced by number.

  Example: **@1, @2, @3**
- The "mid" function has been incorporated.

  Example: **mid(field,1,2)**

APPENDIX E:

ANSWERS TO QUIZ QUERIES

Answer to #1:

select name, hired, commission, salary
from ssalesmn

```
name         hired     commission    salary
-----------------------------------------
Macmillan   01/15/80        55.38   2000.00
Winston     06/01/87       127.03   1200.00
Mifflin     02/10/82                1800.00
***  QUERY OUTPUT TRUNCATED ***
```

Answer to #2:

select customer, subtotal+tax_amount, payment, invoice_date, invoice_number
from sinvoice
where payment > (subtotal+tax_amount).25*

```
customer subtotal+tax     payment invoice_date invoice_number
-------------------------------------------------------------
7             1405.41      500.00  08/20/87                 9
8              255.85      100.00  02/19/87                11
4              788.92      200.00  05/05/87                14
2              209.68      150.00  07/30/87                16
1             1156.69      800.00  09/20/87                17
```

Answer to #3:

select manager, avg(salary), count(*), min(hired)
from ssalesmn
group by manager

```
manager     avg(salary) count(*) min(hired)
-----------------------------------------
Jovanovich    1533.3333         3   11/20/84
Knopf         1625.0000         6   01/15/80
```

Answer to #4:

```
select territory, avg(salary)
from ssalesmn
group by territory
having count(*) = 2
```

```
territory avg(salary)
----------------------
NYS          1450.0000
```

Answer to #5:

```
select territory, avg(salary), manager
from ssalesmn
group by territory
having avg(salary) >
                          select avg(salary)
                          from ssalesmn
```

```
territory avg(salary) manager
---------------------------------
NYC          1712.5000 Knopf
PA           1700.0000 Jovanovich
```

Note, in the example, above, that the "having" clause can compare the results of two aggregate functions by using a subquery.

Subqueries in the "having" clause are evaluated in the same manner as they are in the "where" clause. In FPSQL subqueries can be used in both the "where" and "having" clauses at the same time.

Answer to #6:

select manager, territory, hired, name
from ssalesmn
order by manager, territory, hired

```
manager     territory hired     name
-------------------------------------------
Jovanovich  NJ        03/14/86 Holt
Jovanovich  OH        08/12/85 Wiley
Jovanovich  PA        11/20/83 Hall
Knopf       NYC       01/15/80 Macmillan
Knopf       NYC       03/13/81 McGraw
Knopf       NYC       02/10/82 Mifflin
Knopf       NYC       06/01/27 Winston
Knopf       NYS       10/17/84 Reinhart
Knopf       NYS       11/11/86 Haughton
```

Answer to #7:

```
select territory,
        ssalesmn.name 'Sales Rep',
        sinvoice.@A1 'Product',
        sinvoice.@A2 'Quan.',
        description 'Description',
        sproduct.price * sinvoice.@A2'Total Sales'

from sinvoice, scllent, sproduct, ssalesmn
where
        sinvoice.@l = sclient.@l
        and
        sinvoice.@A1 = sproduct.product
        and
        sinvoice.salesman = ssalesmn.name

order by territory, ssalesmn.name, sinvoice.@A1
```

```
Territory Sales Rep Product  Quan.  Description         Total Sales
------------------------------------------------------------------
NJ        Holt      O1-2345     2  Soldering Iron            28.00
NJ        Holt      23-4567     1  Metal Tracker            219.00
NJ        Holt      56-7890     3  Thermo Reader             36.00
NJ        Holt      89-0123     3  Lite Dimmer               28.05
NYC       Macmillan 78-9012     3  Barometer Kit            598.80
NYC       Macmillan 89-0123    10  Lite Dimmer               93.50
NYC       Winston   12-3456     1  Touch Tone Phone          29.99
NYC       Winston   45-6789     4  DigiClock                 48.00
NYC       Winston   58-7890     3  Thermo Reader             36.00
NYC       Winston   67-8901    15  Humidity Meter          1492.50
NYC       Winston   78-9012     5  Barometer Kit             998.00
NYC       Winston   78-9012     1  Barometer Kit             199.60
NYC       Winston   89-0123     2  Lite Dimmer               18.70
OH        Wiley     01-2345     2  Soldering Iron            28.00
OH        Wiley     12-3456     2  Touch Tone Phone          59.98
OH        Wiley     23-4567     1  Metal Tracker            219.00
*** QUERY OUTPUT TRUNCATED ***
```

Answer to #8:

```
set output 'contract' set lines 72,84 set title 'Assets Sorted by Location'
```

INDEX

A

B

D

E

F

R

S

T

V

W